A Python Toolkit for Kernel Estimation for SISO Linear Systems

Nithilasaravanan Kuppan

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

June 2022

Abstract

Any modern control-based problem, specifically complex feedback control system design, requires an efficient and robust system dynamics estimation methodology. It is not pragmatic to always assume that complete or even partial knowledge of the dynamics is available beforehand. Therefore its crucial to investigate and rigorously test out conventional and novel estimation and filtering techniques. This thesis discusses an updated filtering and estimation algorithm that does not require any knowledge of the system's structure, noise, or even the right initial conditions to estimate the system parameters and state. To provide a true account of the performance of this algorithm, both the state and parameter estimation ability of the novel forward-backward kernel-based algorithm was tested against a modified version of the Unscented Kalman Filter strengthened with Raunch-Tung-Striebel (RTS) smoother with varying levels of added noise. The thesis also thoroughly describes a nifty python-based code repository, developed to bridge the current lack of a user-friendly estimation and filtering toolkit that combines both conventional and innovative methods for Single Input Single Output (SISO) Linear Time-Invariant (LTI) system identification. This incisive library circumvents the requirement for a deep knowledge of various python libraries and instead, provides a practical approach to solving such a problem. This thesis, along with the developed repository strives to be a starting point for any estimation problem in the future.

Abrégé

Tout problème moderne basé sur le contrôle, en particulier la conception complexe d'un système de contrôle par rétroaction, nécessite une méthodologie efficace et robuste d'estimation de la dynamique du système. Il n'est pas pragmatique de toujours supposer qu'une connaissance complète ou même partielle de la dynamique est disponible à l'avance. Par conséquent, il est crucial d'étudier et de tester rigoureusement les nouvelles techniques d'estimation et de filtrage conventionnelles. Cette thèse traite d'un algorithme de filtrage et d'estimation mis à jour qui ne nécessite aucune connaissance de la structure du système, du bruit ou même des bonnes conditions initiales pour estimer les paramètres et l'état du système. Pour fournir un compte rendu fidèle des performances de cet algorithme, la capacité d'estimation de l'état et des paramètres du nouvel algorithme basé sur l'avant-arrière noyau a été testée par rapport à une version modifiée du filtre de Kalman sans parfum renforcée avec le Rauch-Tung-Striebel (RTS) adoucisseur avec différents niveaux de bruit ajouté. La thèse décrit également en détail un référentiel de code basé sur python, développé pour combler le manque actuel d'une boite à outils d'estimation et de filtrage convivial qui combine à la fois des méthodes conventionnelles et innovantes pour identification d'entrée unique et sortie unique, invariant de temps linéaire du système. Cette bibliothèque incisive contourne l'exigence d'une connaissance approfondie de diverses bibliothèques Python et fournit à la place une approche pratique pour résoudre un tel problème. Cette thèse, ainsi que le référentiel développé, s'efforce d'être un point de départ pour tout problème d'estimation à l'avenir.

Acknowledgments

It has been an incredible delight to work under the supervision of Professor Dr. Hannah Michalska throughout my master's at McGill University. She has been a great professor and a guiding figure throughout my research. I am grateful to her for giving me the opportunity to come to McGill University.

I would also like to sincerely thank my fellow researcher Manoj Krishna Venkatesan for his help in the development of several aspects of this thesis - without him, a lot of the ambitious goals illustrated in this thesis would not have been possible. Along with Manoj, I would also like to thank Israel Hernandez Quinto and Adharsh Mahesh Kumaar for their constant support and words of encouragement. This work would not have been possible without the research work done by Luis Medrano del Rosal, Shantanil Bagchi, Nikhil Jayam, and fellow students of the Systems and Control group under Dr. Michalska.

To all my friends - Manoj, Vishal, Adharsh, Rhythm, and Sai, I thank you for your friendship and support during all the hard times in my life and for being there for me. A special thanks to Shreya for the constant encouragement and for pushing me when required. Lastly, I would like to express my gratitude and respect to my parents for everything that they've done for me, are doing for me, and will keep doing for me.

Preface

This section is used to declare that the work presented in this document was completed and carried out by Nithilasaravanan Kuppan and is part of the collaborative work of a three-member team guided by Professor Hannah Michalska. The team includes Manoj Krishna Venkatesan, Israel Hernandez Quinto, and the author.

The theory of representing an n^{th} order system using kernels in Reproducing Kernel Hilbert Space (RKHS) was derived by Debarshi Patanjali Ghoshal, a former Ph.D. scholar in the research group. The theoretical background for the RKHS approaches for optimization is based on the research notes by Professor Hannah Michalska, which is duly acknowledged. The preliminary work for the estimation of a fourth-order system was worked up by Luis Medrano del Rosal and Adharsh Mahesh Kumaar, while the integration with Kalman filter and RTS was developed in conjunction with Manoj Krishna Venkatesan. Subsequently, the development of the Python Estimation Toolkits (PETs) library was also the result of a collaboration with Manoj Krishna Venkatesan. The development of an efficient version of UKF for parameter estimation and the establishment of a dedicated pipeline for such problems was proposed and carried out by the author.

Contents

List of Figures

List of Tables

List of Acronyms

RKHS	Reproducing Kernel Hilbert Space
LTI	Linear Time Invariant
LTV	Linear Time Variant
PETs	Python Estimation Toolkits
RMSE	Root Mean Square Error
MAD	Maximum Absolute Difference
SISO	Single Input Single Output
MIMO	Multi Input Multi Output
BLUE	Best Linear Unbiased Estimator
OLS	Ordinary Least Squares
IV	Instrument Variable
GLS	Generalized Least Squares
MRRLS	Modified Recursive Regularized Least Squares
RTS	Raunch-Tung-Striebel
UKF	Unscented Kalman Filter
EKF	Extended Kalman Filter
BMFLS	Biswas-Mahalanabis Fixed-Lag Smoother
AWGN	Additive White Gaussian Noise
SNR	Signal to Noise Ratio
JURTS	Joint Unscented Rauch-Tung-Striebel
JSON	JavaScript Object Notation
CLI	Command Line Interface

Chapter 1

Introduction

1.1 Motivation

In the history of automatic control, it has long been recognized that the knowledge about a system and its environment required to build a control system is rarely available *a priori*. It is possible that system definition equations are known but it is common for specific parameters to be unknown. Therefore, with the purpose of designing a control strategy coupled with the possibility to experiment on the system to identify missing elements, several techniques referred to as *system identification* techniques were engendered. One such powerful technique often used in *classical* control theory for system identification is frequency analysis.

However, the models used in *modern* control theory are mostly parametric in nature, in terms of the state equations. Without any precise information about the model, practically all real-world issues, from robotics to biological systems, require a strong, trustworthy algorithm to adequately predict the state variables and output signals. As a result, this thesis explores a mix of unique and enhanced versions of a traditional technique that can be used to successfully tackle such problems. This thesis also introduces a one-of-a-kind tool that was created to make system identification and filtering accessible to the general public, filling a gap in the market for a modular toolkit that is both easy to use and scalable to address a larger range of challenges in the future.

1.2 Background

Zadeh [1] defined the general identification problem as determining a system within a specified class of systems, based on input and output, to which the system under test is equivalent. This can be accomplished using a variety of approaches based on the problem's elements, such as the class of systems, the input signals, and the criterion. The models can be characterized as *non-parametric* representations such as impulse responses, transfer functions, etc or by *parametric* models such as state models comprising of the state, input, output and parameter vectors [2, 3]. Because of its generality, flexibility in input selection, and ability to describe systems in *canonical representations*, modal parameter estimation has been the most preferable option, leading to the development of multiple efficient algorithms [4–6].

In the absence of any external factors operating on the system, the state of a system is a minimal set of (state) variables that specifies enough about the system to predict its future behavior. The major issue in the state estimation task is to deliver reliable estimation while being computationally tractable under process and measurement uncertainty [7]. One of the earliest methods for forming an optimal estimate from noisy data is the method of least squares which was formalized by Carl Friedrich Gauss in the late eighteenth century [8]. All the pioneering work done in the field of probability in the next few decades led to the development of *Markov process / Markov chain* - a random process with the property that the evolution over time of its probability distribution can be treated as an initial-value problem [9]. This led to the initiation of the concept of linear least squares extrapolation of stationary processes by A. N. Kolmogorov which was furthered by many [10]. Following this, a crucial optimal estimator called the *Wiener - Kolmogorov filter* was developed which used probability measures on function spaces to represent uncertain dynamics [11] [12]. The Kalman filter has become one of the most widely used algorithms for state estimates over time. By calculating a joint probability distribution over the variables for each time frame, the Kalman filter employs a sequence of noisy and imprecise observations (data) to estimate unknown states of the system that are more accurate than those based on a single observation [13]. The Extended Kalman filter (EKF) and Unscented Kalman filter (UKF) are nonlinear system expansions of the classical Kalman filter. The EKF linearizes the system around an estimated mean trajectory with previously estimated covariance, while the UKF is a non-linear transformation-based method for linearizing non-linear systems [14] [15]. The Kalman filter is frequently used due to its simplicity and ease of implementation. On the other hand, it necessitates a precise system model and statistical noise characteristics. These requirements cannot be met in complicated systems and can be extremely difficult to implement [16] [17].

To counter these issues of a traditional Kalman filter, this thesis talks about an updated estimation scheme that leverages the knowledge of the system characteristic equation and builds kernel representations of differential invariants, first presented in [18] and continued in [19–22]. A forward-backward double-sided kernel was developed in the Reproducing Kernel Hilbert Space (RKHS) for a general SISO LTI homogeneous system of order n. Because this kernel estimator works over a finite window, it is classified as a fixed interval smoother. This algorithm circumvents the need for *a priori* knowledge of the model structure, initial conditions, or the noise characteristics. The approach uses annihilator functions and repeated integration to estimate the system parameters without significant interference from system noise, which is then utilized to estimate the system state. The noisy measurement is filtered by the orthogonal projection onto the finite-dimensional subspace of RKHS spanned by the fundamental solutions of the system's characteristic equation, yielding the system state.

1.3 Thesis objectives and achievement

As previously stated, the objective of this thesis is to present the updated novel kernel-based estimation algorithm and explain the extended multiple regression algorithm used thorough-out this thesis, called *Modified Recursive Regularized Least Squares* (MRRLS). And in order to provide a true and detailed account of its state estimation capabilities, the projection method has been compared with a linear Kalman filter appended with a fixed interval smoother, called Raunch-Tung-Striebel (RTS). This thesis also features an augmented Unscented Kalman filter (with RTS) called *JURTS* that has been designed in such a way that it can simultaneously estimate the parameters and filter for the state from noisy input. Naturally, this enables a rigorous comparison between *JURTS* and the kernel estimator (with MRRLS and projection method).

Apart from the aforementioned experiments, the primary achievement of this thesis work is the creation of an agile and capable Python-based estimation library called *Python Estimation Toolkits* (PETs). The driving motivation for this toolkit was the lack of an easy-to-use, modular estimation library that does not require the user to be an expert in handling complex Python libraries. PETs includes all the estimation algorithms discussed in this thesis, which makes it a versatile tool that houses both conventional (Kalman-based) filters and novel (Kernel-based) filters in one nifty, customizable package. This python library has been crafted with the combined effort of Manoj Krishna Venkatesan (research partner) [23]; Dr. Hannah Michalska and the graduate students under her supervision have been conducting

research on this topic for many years, and PETs is a testament to their efforts. The details surrounding the construction, setup, and running of this repository have been incisively explained in Chapter 6.

Conclusively, the thesis along with the newly created estimation repository strives to be the starting point for estimation problems - setting up a pipeline of processes that can be followed for any estimation problem. Currently only SISO LTI systems are supported however theoretical research work by fellow lab-mates suggests that extending this logic (and hence, the code) to linear time-varying (LTV) systems, non-linear systems, and even MIMO systems is easily achievable [17, 24]. The modular structure allows the user to tinker with the code and augment additional libraries to add more functionality downstream, for instance, the Model Predictive Control based controller as shown in [25].

1.4 Thesis organization

The thesis has been bucketed into the following seven chapters -

- **Chapter 1** briefly introduces the problem statement along with the motivation and summarizes the literature associated with the problem of state and parameter estimation. It also presents the core objective of the thesis along with its structure
- **Chapter 2** provides an extensive account on a conventional state estimation methodology - Kalman filters. It discusses the algorithm, along with a powerful smoothing algorithm called Rauch–Tung–Striebel (RTS) and how its been adapted for the use-case presented in the thesis
- **Chapter 3** introduces the double sided kernel approach and the relevant equations/theorems. Furthermore, the chapter also presents the derivation of the updated kernel based multiple regression equations for a 4^{th} order SISO LTI system. The chapter ends with a description of the Modified Recursive Regularized Least Squares (MRRLS) algorithm used for parameter estimation and the projection method for state reconstruction
- **Chapter 4** methodically compares the state estimation capabilities of the projection method and the Kalman + RTS filter using systems of varying order and noise levels
- **Chapter 5** discusses a novel augmented Unscented Kalman (+RTS) filter called JU-RTS that circumvents the need for Kalman filters to have prior knowledge of the system dynamics to simultaneously predict the state and the parameter of the system. The

chapter also compares its joint estimation ability with the kernel-based approach

- **Chapter 6** extensively explains the Python Estimation Toolkits (PETs) that was developed for joint state and parameter estimation, leveraging four different types of both novel and conventional algorithms
- **Chapter 7** concludes the thesis with a short introduction to constrained filtering and summarizes the thesis by providing recommendations and possible future extensions on the presented research

Chapter 2

State Estimation of LTI systems - Kalman Filter Approach

The general problem of system identification in Linear Time Invariant (LTI) Systems is defined in this chapter followed by a specific case of state estimation over a finite horizon when the elements of matrix $A \in \mathbb{R}^{n\times n}$ are known, i.e., the system parameters are already known. Specifically, Kalman filter augmented with an optimal smoothing algorithm using Rauch-Tung-Striebel (RTS) has been explained in great detail.

2.1 Finite interval estimation problem for Linear time invariant (LTI) systems

Consider a general n^{th} order, strictly proper and minimal Single-input Single-output (SISO) LTI system in state space form evolving on a given finite time interval $[a, b] \subset \mathbb{R}$:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{2.1}$$

with $x \in \mathbb{R}^n$; column matrix consisting of state variables called **state vector**, $x(0) = x_0$, $\dot{x} = dx/dt$, $y \in \mathbb{R}$, $u \in \mathbb{R}$. The matrices $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times 1}, C \in \mathbb{R}^{1\times n}$ are the system,

input, and output matrix respectively which are represented as,

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{n-1} \end{bmatrix}, \tag{2.2}$$

(2.3)

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2.4}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{2.5}$$

The input-output equation for system (2.1) becomes

$$y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + ... + a_1y^{(1)}(t) + a_0y(t) = -b_{n-1}u^{(n-1)}(t) - ... - b_0u(t) \tag{2.6}$$

where -b_i for $i = 0, ..., n-1$ are the coefficients of the polynomial in the numerator of the rational transfer function for (2.1). The unknown parameters a_i and b_i for $i = 0, .., n-1$ need to be estimated from noisy observations of the system's output, $y_M(t)$ for $t \in [a, b]$.

The estimation problem is stated as follows. Given an arbitrary finite interval of time $[a, b]$:

(1) The dimension of the state vector of the LTI system is not known *a priori*. However, a set of possible dimensions for the system is assumed

(2) The system input function $u(t)$ is equal to zero (No input)

(3) The output of the system is observed as a single realization of a 'continuous' measurement process $y_M(t) := y(t) + \eta(t), t \in [a, b]$ in which η denotes additive white Gaussian noise with unknown intensity (variance) σ^2.

An implementable version of assumption (3) simply requires availability of an unrestricted number of output measurements over the observation horizon $[a, b]$.

Under the constraints of process and measurement noise, getting reliable state values poses a significant problem. Unmodeled dynamics, modeling approximations, and parameter uncertainties all contribute to process noise in the system. Based on the physical characteristics of the sensor and the measurement process, measurement noise is added. Given that the system parameters are known, there are numerous ways for predicting the state vector from a given noisy input signal [23].

One of the earliest methods to achieve convergence of a state irrespective of its initial condition was by using a Luenberger observer [26]. In this model, the error caused by the difference between the expected and measured output is sent back to the model. If the system is observable, the model's state converges to the system's actual state however, only deterministic systems are suitable for this strategy. Unfortunately, due to the induced process and measurement noise, the majority of the systems are stochastic. To circumvent this issue Rudolph Kalman came up with the Kalman filter [13], which uses the stochastic properties of the noise to perform accurate state estimation.

2.2 Kalman filters for state estimation

The Kalman filter is one of the most widely used tools in mathematics, named after Rudolph E. Kalman, who published his groundbreaking work on the linear filtering problem in 1960. [13]. The Kalman filter is primarily a set of mathematical equations that implements an *optimal recursive data processing algorithm* that is optimal in the sense that it minimizes the estimated *error* covariance - when some presumed conditions are fulfilled [27].

Kalman filter uses a series of measurements *(which might be corrupted by noise, biases, and device inaccuracies)* and combines it with prior knowledge about the system to produce estimates of unknown variables by estimating a joint probability distribution over the variables for each time frame. The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: *time update equations* and *measurement update equations*. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the *a priori* estimates for the next time step. The measurement update

equations are responsible for the feedback—i.e. for incorporating a new measurement into the *a priori* estimate to obtain an improved *a posteriori* estimate [28].

It can run in real-time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required. The time update equations can also be thought of as *predictor* equations, while the measurement update equations can be thought of as *corrector* equations. Indeed the final estimation algorithm resembles that of a *predictor-corrector* algorithm for solving numerical problems as shown below in figure 2.1.

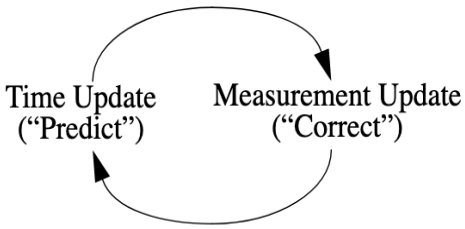


Fig. 2.1 The recursive Kalman filter cycle

2.2.1 Kalman filter algorithm

The following section provides a quick summary of the Kalman algorithm in terms of the equations used, without any of the derivations *(abridged from [28])*.

- **Model and Observation**: A **stochastic time-variant linear system** is described by the difference equation and the observation model:

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1} \tag{2.7}$$

$$z_k = H_k x_k + v_k \tag{2.8}$$

  where the control input u_k is a known deterministic vector. The initial state x_0 is a random vector with known mean $\mu_0 = E[x_0]$ and covariance $P_0 = E[(x_0 - \mu_0)(x_0 - \mu_0)^T]$.

  We assume that the random vector w_k captures uncertainties in the model and v_k denotes the measurement noise. Both are temporally uncorrelated (white noise), zero mean random sequences with known covariances $E[w_k w_k^T] = Q_k$, $E[v_k v_k^T] = R_k$ where Q_k is process noise covariance matrix and R_k is measurement noise covariance matrix.

- **Initialization**: Because the only information given is the initial state's mean, μ_0, and covariance, P_0, the initial optimal estimate x_0^a and the error covariance are

$$x_0^a = \mu_0 = E[x_0] \tag{2.9}$$
$$P_0 = E[(x_0 - x_0^a))(x_0 - x_0^a))^T] \tag{2.10}$$

- **Model Forecast Step/Predictor**: The *predictor* equations project the state and covariance estimates forward from time step $k-1$ to step k.

$$x_k^f = A_{k-1}x_{k-1}^a + B_{k-1}u_{k-1} \tag{2.11}$$
$$P_k^f = A_{k-1}P_{k-1}A_{k-1}^T + Q_{k-1} \tag{2.12}$$

- **Corrector Step/Update**: The primary task of the *updation* step is to compute the Kalman gain, K_k. The next step is to get the measurement z_k and then to generate an *a posteriori* state estimate. Lastly, an *a posteriori* error covariance estimate is generated.

$$K_k = P_k^f H_k^T (H_k P_k^f H_k^T + R_K)^{-1} \tag{2.13}$$
$$x_k^a = x_k^f + K_k(z_k - H_k x_k^f) \tag{2.14}$$
$$P_k = (I - K_k H_k)P_k^f \tag{2.15}$$

After each *predict* and *update* step, the whole process is repeated with previous *a posteriori* estimates to obtain the new *a priori* estimates. Figure 2.2 shows the high level diagram along with a summary of all the equations.

In the next section, the concept of *smoothing* has been explored which assists in further *fine tuning* of the estimates achieved from the Kalman filter.

2.3 Optimal smoothing and RTS

Optimal smoothing methods are derivatives of the same Kalman filter methods for solving the same class of problems. This section will illustrate the different types of *smoothers* - which are algorithmic implementations of smoothing methods, that are essentially extensions of Kalman filtering. Finally, this chapter will go through one of the methodologies that has been employed extensively in this thesis. [29].

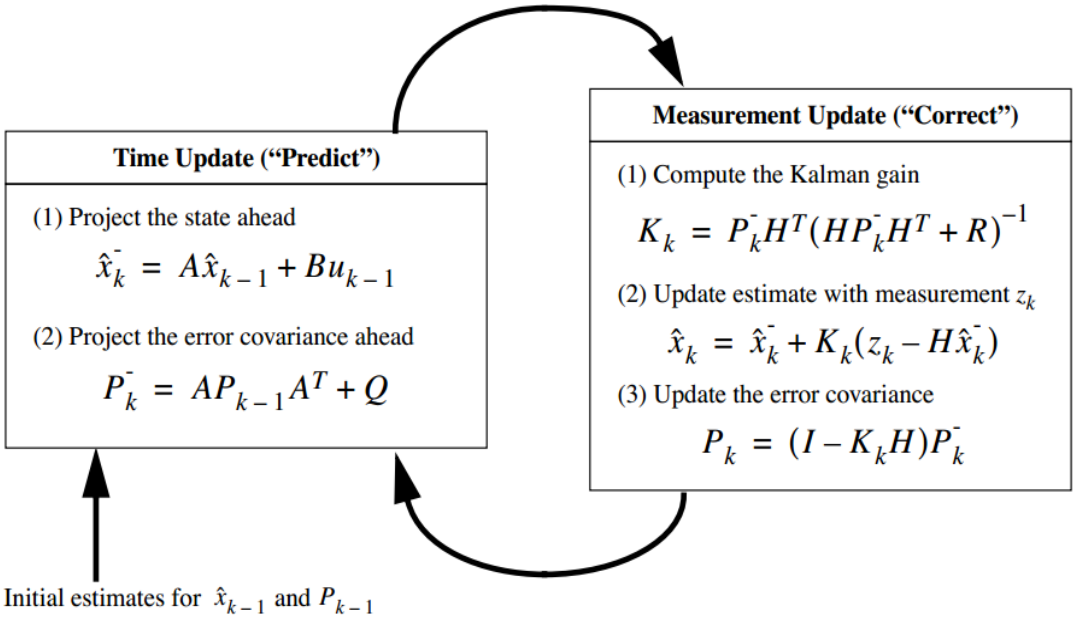


Fig. 2.2 Operation of a Kalman filter - A summary

2.3.1 Types of smoothers

There are three major classes of smoothing algorithms, bucketed by the importance of measurement data on the estimated state vectors [29–31].

Fixed-lag smoothing

Fixed-lag smoothers use all measurements made over a time interval $t_{start} \leq t_{meas} \leq t_{est} + \Delta t_{lag}$ for the estimate $\hat{x}(t_{est})$ at time t_{est}. This implies that the generated estimate at time t is for the value of x at time $t - \Delta t_{lag}$, where Δt_{lag} is a fixed time.

Fixed-lag smoothers are commonly used in communications to improve signal estimation, however the trade-off is that this method adds delay to the signal (thus the name). These approaches operate in real time, using all measurements up to the present moment, but produce an estimate in a deferred time frame.

One of the frequently used algorithms in this class of smoothers is the Biswas-Mahalanabis fixed-lag smoother *(BMFLS)* [32] which is a *state augmentation* filtering approach based on [33, 34]. It's a Kalman filter with an augmented state vector made up of the original state

vector's successive values throughout a discrete time window of fixed width. If $\Delta t_{lag} = l\Delta t$ is the time lag, which is l discrete time steps, then the augmented state vector at time t_k is of length $n(l+1)$ n-dimensional subvectors $x_k, x_{k-1}, x_{k-2}, ..., x_{k-l}$.

Fixed-point smoothing

Fixed-point smoothers generate an estimate $\hat{x}(t_{fixed})$ of x at a fixed time t_{fixed} based on all measurements $z(t_{meas})$ up to the current time $t(t_{start} \leq t_{meas} \leq t)$. Fixed-point smoothers act like a predictor when $t < t_{fixed}$, as filters when $t = t_{fixed}$ and as smoothers when $t > t_{fixed}$.

Fixed-point smoothing is beneficial for estimation problems where the system state is only of interest at a specific time t_{fixed}, which is typically the initial state. As a result, this thesis does not make use of this type of smoother.

Fixed-interval smoothing

Fixed-interval smoothers use all the measurements made at times t_{meas} over a fixed interval $t_{start} \leq t_{meas} \leq t_{end}$ to produce an estimated state vector $\hat{x}(t_{est})$ at time $t_{start} \leq t_{est} \leq t_{end}$ in the same fixed interval.

Fixed-interval smoothing can be performed at any time after the required measurements have been acquired, hence it is typically used to post-process measurements taken during a procedure. When compared to other recursive forms of filters/smoothers, fixed-interval smoothers have been shown to produce better results.

In the following subsection, a smoother developed by *Rauch, Tung* and *Striebel* has been described in detail which has been the *go-to* method due to its ease of implementation and computational efficiency [35].

2.3.2 Rauch-Tung-Striebel

Rauch-Tung-Striebel smoother [29, 31, 35, 36] is based on a *two-filter* model

(1) A **forward filter** running forward in time. The forward filter's estimate is based on all of the measurements taken up to that point in time, and the corresponding estimation uncertainty covariance quantifies the estimation uncertainty based on all of those data

(2) A **backward filter** running backward in time. The estimate from the backward filter at any given moment is reliant on all subsequent observations, and the corresponding

estimation uncertainty covariance quantifies the estimation uncertainty based on all those measurements

At each time t, the forward filter generates the covariance matrix $P_{[f]}(t)$ representing the mean-squared uncertainty in the estimate $\hat{x}_{[f]}(t)$ using all measurements $z(s)$ for $s \leq t$. Similarly, the backward filter generates the covariance matrix $P_{[b]}(t)$ representing the mean-squared uncertainty in the estimate $\hat{x}_{[b]}(t)$ using all measurements $z(s)$ for $s \geq t$. The optimal smoother combines $\hat{x}_{[f]}(t)$ and $\hat{x}_{[b]}(t)$, using $P_{[f]}(t)$ and $P_{[b]}(t)$ in a Kalman filter to minimize the resulting covariance matrix $P_{[s]}(t)$ of smoother uncertainty. $P_{[s]}(t)$ indicates how the smoother performs [29].

2.3.3 RTS Algorithm

The following section provides a quick summary of the Kalman algorithm in terms of the equations used, without any of the derivations, from [29, 31, 35, 36].

- **Forward pass**: Based on the equations given in the algorithm for Kalman filter, the standard filtered quantities, i.e., the smoothed means and corresponding covariances $\hat{x}_{k|k-1}, \hat{x}_{k|k}, P_{k|k-1}, P_{k|k}$ for $k = 0, ..., n$ are calculated and stored in memory
- **Backward pass**: $\hat{x}_{k|n}$ is computed using

$$\hat{x}_{k|n} = \hat{x}_{k|k} + A_k \left(\hat{x}_{k+1|n} - \hat{x}_{k+1|k} \right), \quad k = n-1, \ldots, 0 \tag{2.16}$$

where,

$$A_k = P_{k|k-1} \bar{F}_k^T P_{k+1|k}^{-1} \tag{2.17}$$

and

$$P_{k+1|k} = \bar{F}_k P_{k|k-1} \bar{F}_k^T \tag{2.18}$$

The error covariance can be found by

$$P_{k|n} = P_{k|k} + A_k \left(P_{k+1|n} - P_{k+1|k} \right) A_k^T \tag{2.19}$$

In the above equations, A_k is the *smoother gain matrix*, n is the final time step, $P_{k|n}$ is the corresponding *state error covariance* matrix, $\bar{F}_k$ is the state transition matrix and $\hat{x}_{k|n}$ is the

smoothed state at time step k.

2.3.4 Improvement over unsmoothed Kalman outputs [29]

For asymptotically exponentially stable dynamic systems, theoretical limits on the asymptotic improvement of smoothing over filtering were shown in [37]. The limit was determined to be a factor of two in mean-squared estimation uncertainty, however there is the possibility of larger improvement in unstable systems.

If $P_{[s]}$ is the covariance matrix of smoothing uncertainty and $P_{[f]}$ is the covariance matrix of filtering uncertainty for multidimensional problems, then for smoothing to be an improvement over filtering

$$P_{[s]} < P_{[f]}, or [P_{[f]} - P_{[s]} \text{ is positive - definite }] \tag{2.20}$$

In practice, this is done by comparing the covariance matrices after both the filter and smoother procedures have been implemented.

2.4 State estimation algorithm

The pseudo-code for leveraging Kalman filter + RTS smoother has been given below in Algorithm 1. This algorithm highlights a general case when the signal to be filtered Y_m of any order n is enveloped in some additive white noise to produce final state estimates x_E. This algorithm, as previously stated, assumes that the system dynamics (parameters) are known.

2.5 Conclusion

This chapter gives a theoretical introduction to a class of estimation problems as well as a succinct description of one of the more widely employed solutions. Because the primary goal of this thesis is to present the reader with a proper estimation framework, this thesis compares a more robust estimation approach that combines double-sided kernels and multiple regression under various situations of noise as well as previous information available to the algorithm. The next chapter will describe this developed method in detail.

Algorithm 1 Calculate state estimates using Kalman + RTS

Initialize: $a_0 = a_{k_n}$ where a_{k_n} are the parameters of the system dynamics of order n
Initialize: $x_{initial} = [y_0;\ y_0^{(1)};\ y_0^{(2)}; ...;\ y_0^{(n)}]$
Procedure Kalman Filter {with known parameters}
Initialize: $P_{initial}, Q, R$ and H matrices
Initialize:

$$A_k = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_{k_0} & -a_{k_1} & -a_{k_2} & \dots & -a_{k_{n-1}} \end{bmatrix}$$

Calculate state transition function $F = e^{A_k t}$
for $k = 0, 1, ..., N$ **do**
  while $Y_{m_k} \in Y_m$ **do**
    Predict I: $x_k^f = F_{k-1} x_{k-1}$ {State Extrapolation [f for forecast]}
    Predict II: $P_k^f = F_{k-1} P_{k-1} F_{k-1}^T + Q$ {Covariance Extrapolation}
    Set $z_k = Y_{m_k}$
    Update I: $K_k = P_k^f H^T (H P_k^f H^T + R_k^{-1})$ {Kalman Gain}
    Update II: $x_k = x_n^f + K_k(z_k - H x_k)$ {State Update}
    Update III: $P_k = (I - K_n H) P_k^f$ {Covariance Update}
    return x_k, P_k
  end while
end for
Append x_k to X and P_k to P {Stored as a batch}
Procedure Rauch-Tung-Striebel {Fixed Interval Smoothing}
Initialize $x_{\hat{k}|n} = X[-1]$ and $P_{k|n} = P[-1]$ {Backward filter}
for $k = N-1, ..., 1, 0$ **do**
  while $x_k \in X$ **do**
    Calculate $A_k = P_{k|k-1} \bar{F}_k^T P_{k+1|k}^{-1}$ {Smoother gain matrix}
    Calculate $P_{k+1|k} = \bar{F}_k P_{k|k-1} \bar{F}_k^T$ using 2.19 {Error covariance}
    Update: $\hat{x}_{k|n} = \hat{x}_{k|k} + A_k \left(\hat{x}_{k+1|n} - \hat{x}_{k+1|k} \right)$
    return x_k
  end while
end for
Append x_k to x_E and plot the results

Chapter 3

Parameter and State Estimation of LTI systems - Double sided Kernel Approach

Unlike the methods mentioned in the previous chapter, this chapter discusses the double-sided kernel approach [38] which does not require any knowledge of the underlying dynamics for the estimation of the states. This method employs a forward-backward integration to convert a high order differential equation into an integral form with no singularities in the time interval. This chapter starts with an overview of the method for a system defined in a finite time interval, as discussed in section 2.1. Later on, the chapter builds up on how double-sided kernels can be used for the joint estimation of both the state as well as the parameters on a finite interval $[a, b]$ *(specifically, for a* 4^{th} *order system).* As discussed in [39], [40] these methods were re-derived using *up-to-date* formulae and the same has been presented in this chapter as discussed in [24] with an important modification.

3.1 System differential invariance in kernel representation

The kernel representation of the n^{th} order SISO LTI system, as introduced by [22] ascertains the fact that the key to finite interval estimation approach is the integral representation of the controlled differential invariance of the system. The parameter estimation of a homogeneous system can be viewed as the identification of a differential invariant $\mathcal{I}$ ($\mathcal{I} \equiv 0$, $\equiv$ is 'equivalent

to') which does not change under the action of the system dynamics:

$$\begin{aligned}&\mathcal{I}(t, y(t), y^{(1)}(t), \cdots, y^{(n)}(t)) \\ &\quad = y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + \cdots + a_0 y(t) \equiv const. = 0 \ ; \ t \in [a, b]\end{aligned} \tag{3.1}$$

Equation 3.1 delivers additional information about the behaviour of the system on top of direct observation of the noisy signal y_M. To be essential, the *zero-input response* characterization (3.1) has to be put in a form, which does not depend on the initial or boundary conditions of the system, and that does not involve any time derivatives of the output as they cannot be measured directly. The definitions and theorems in this chapter will assist in realizing an improved characterization.

Definition 2.1.

A pair of smooth (class C^∞) functions (α_a, α_b), $\alpha_s : [a, b] \longrightarrow \mathbb{R}, s = a$ or b, is an annihilator of the boundary conditions for a system (2.1) if the functions α_s are non-negative, monotonic, vanish with their derivatives up to order $n - 1$ at the respective ends of the interval $[a, b]$; i.e.

$$\alpha_s^{(i)}(s) = 0; \quad i = 0, \ldots, n-1; \quad s = a, b; \quad \alpha_s^{(0)} \equiv \alpha_s \tag{3.2}$$

and such that their sum is strictly positive, i.e. for some constant $c > 0$

$$\alpha_{ab}(t) := \alpha_a(t) + \alpha_b(t) > c \ ; \ t \in [a, b] \tag{3.3}$$

A simplest example of such an annihilator for system (2.1) is the pair,

$$\begin{aligned}&\alpha_a(t) := (t-a)^n; \quad \alpha_b(t) := (b-t)^n; \quad t \in [a, b] \\ &\alpha_{ab}(t) := \alpha_a(t) + \alpha_b(t) > 0 \\ &\alpha_{ab}(s) = (b-a)^n; \quad s = a, b\end{aligned} \tag{3.4}$$

With the help of annihilators, the differential invariance representation of the system can be obtained without the knowledge of initial conditions. This differential invariance representation is used to derive a behavioral model through Reproducing Kernel Hilbert Spaces (RKHS) and annihilators. This model can then be used in the reconstruction of output trajectory

and its time derivatives. The following theorem describes such system representation.

Theorem 3.1.1 *There exist Hilbert-Schmidt kernels $K_{DS,y}, K_{DS,u}$, such that input and output functions u and y of system* (2.1) *satisfy*

$$y(t) = \alpha_{ab}^{-1}(t,n)\left[\int_a^b K_{DS,y}(n,t,\tau)y(\tau)\ \mathrm{d}\tau + \int_a^b K_{DS,u}(n,t,\tau)u(\tau)\ \mathrm{d}\tau\right] \tag{3.5}$$

with,

$$\alpha_{ab}^{-1}(t,n) = \frac{1}{(t-a)^n + (b-t)^n} \tag{3.6}$$

Hilbert-Schmidt double-sided kernels of equation (3.5) *are square integrable functions on $L^2[a,b] \times L^2[a,b]$ and are expressed in terms of the forward and backward kernels given below:*

$$K_{DS,y}(n,t,\tau) \triangleq \begin{cases} K_{F,y}(n,t,\tau), & \text{for} \quad \tau \leq t \\ K_{B,y}(n,t,\tau), & \text{for} \quad \tau > t \end{cases} \tag{3.7}$$

$$K_{DS,u}(n,t,\tau) \triangleq \begin{cases} K_{F,u}(n,t,\tau), & \text{for} \quad \tau \leq t \\ K_{B,u}(n,t,\tau), & \text{for} \quad \tau > t \end{cases} \tag{3.8}$$

The kernel functions $K_{DS,y}, K_{DS,u}$ are n - 1 times differentiable functions of t. The forward kernels $K_{F,y}(n,t,\tau), K_{F,u}(n,t,\tau)$ and backward kernels $K_{B,y}(n,t,\tau), K_{B,u}(n,t,\tau)$ in equation (3.7) *and* (3.8) *are given below:*

$$K_{F,y}(n,t,\tau) = \sum_{j=1}^{n}(-1)^{j+1}\binom{n}{j}\frac{n!(t-\tau)^{j-1}(\tau-a)^{n-j}}{(n-j)!(j-1)!} + \sum_{i=0}^{n-1}a_i\sum_{j=0}^{i}(-1)^{j+1}\binom{i}{j}\frac{n!(t-\tau)^{n-i+j-1}(\tau-a)^{n-j}}{(n-j)!(n-i+j-1)!} \tag{3.9}$$

$$K_{B,y}(n,t,\tau) = \sum_{j=1}^{n}\binom{n}{j}\frac{n!(t-\tau)^{j-1}(b-\tau)^{n-j}}{(n-j)!(j-1)!} + \sum_{i=0}^{n-1}a_i\sum_{j=0}^{i}\binom{i}{j}\frac{n!(t-\tau)^{n-i+j-1}(b-\tau)^{n-j}}{(n-j)!(n-i+j-1)!} \tag{3.10}$$

$$K_{F,u}(n,t,\tau) = \sum_{i=0}^{n-1}b_i\sum_{j=0}^{i}(-1)^{j+1}\binom{i}{j}\frac{n!(t-\tau)^{n-i+j-1}(\tau-a)^{n-j}}{(n-j)!(n-i+j-1)!} \tag{3.11}$$

$$K_{B,u}(n,t,\tau) = \sum_{i=0}^{n-1}b_i\sum_{j=0}^{i}\binom{i}{j}\frac{n!(t-\tau)^{n-i+j-1}(b-\tau)^{n-j}}{(n-j)!(n-i+j-1)!} \tag{3.12}$$

The proof for theorem 3.1.1, can be found in [21] (pp. 153-157). Equation (3.5) is the integral equation that eliminates the need for boundary conditions as they are annihilated during every integration operation by the presence of the annihilating factors α_a and α_b. Theorem 3.1.1 will help us in obtaining the time derivatives of the system output $y^{(k)}, k = 1, \ldots, n-1$ recursively, as shown in the following Theorem 3.1.2.

Theorem 3.1.2 *There exist Hilbert-Schmidt kernels $K_{F,k,y}, K_{F,k,u}, K_{B,k,y}, K_{B,k,y}, k = 1, \ldots, n-1$ such that the derivatives of the output function in* (2.1) *can be computed recursively as follows:*

$$y^{(k)}(t) = \frac{1}{(t-a)^n + (b-t)^n}\left[\sum_{i=1}^{k}(-1)^{i+1}\begin{pmatrix} p+i-1 \\ i \end{pmatrix}\frac{n!(t-a)^{n-i}y^{(k-i)}(t)}{(n-i)!}\right.$$

$$
\begin{aligned}
&+\sum_{i=p}^{n-1} a_i \sum_{j=0}^{i-p} (-1)^{j+1} \left(\begin{array}{c} p+j-1 \\ j \end{array} \right) \frac{n!(t-a)^{n-j} y^{(i-j-p)}(t)}{(n-j)!} \\
&+\int_a^t K_{F,k,y}(n,p,t,\tau) y(\tau) d\tau + \int_a^t K_{F,k,u}(n,p,t,\tau) u(\tau) d\tau \\
&+\sum_{i=p}^{n-1} b_i \sum_{j=0}^{i-p} (-1)^{j+1} \left(\begin{array}{c} p+j-1 \\ j \end{array} \right) \frac{n!(t-a)^{n-j} u^{(i-j-p)}(t)}{(n-j)!} \\
&-\sum_{i=1}^{k} \left(\begin{array}{c} p+i-1 \\ i \end{array} \right) \frac{n!(b-t)^{n-i} y^{(k-i)}(t)}{(n-i)!} \\
&-\sum_{i=p}^{n-1} a_i \sum_{j=0}^{i-p} \left(\begin{array}{c} p+j-1 \\ j \end{array} \right) \frac{n!(b-t)^{n-j} y^{(i-j-p)}(t)}{(n-j)!} \\
&+\int_t^b K_{B,k,y}(n,p,t,\tau) y(\tau) d\tau + \int_t^b K_{B,k,u}(n,p,t,\tau) u(\tau) d\tau \\
&-\sum_{i=p}^{n-1} b_i \sum_{j=0}^{i-p} \left(\begin{array}{c} p+j-1 \\ j \end{array} \right) \frac{n!(b-t)^{n-j} u^{(i-j-p)}(t)}{(n-j)!} \Bigg]
\end{aligned}
$$

where $p = n - k$ *and:*

$$
\begin{aligned}
K_{F,k,y}(n,p,t,\tau) = &\sum_{j=1}^{p} (-1)^{j+n-p+1} \left(\begin{array}{c} n \\ n-p+j \end{array} \right) \frac{n!(t-\tau)^{j-1}(\tau-a)^{p-j}}{(p-j)!(j-1)!} \\
&+\sum_{i=0}^{p-1} a_i \sum_{j=0}^{i} (-1)^{j+1} \left(\begin{array}{c} i \\ j \end{array} \right) \frac{n!(t-\tau)^{p-i+j-1}(\tau-a)^{n-j}}{(n-j)!(p-i+j-1)!} \\
&+\sum_{i=p}^{n-1} a_i \sum_{j=1}^{p} (-1)^{j+i-p+1} \left(\begin{array}{c} i \\ i-p+j \end{array} \right) \frac{n!(t-\tau)^{j-1}(\tau-a)^{n-i+p-j}}{(n-i+p-j)!(j-1)!}
\end{aligned}
\tag{3.13}
$$

$$
\begin{aligned}
K_{F,k,u}(n,p,t,\tau) = &\sum_{i=0}^{p-1} b_i \sum_{j=0}^{i} (-1)^{j+1} \left(\begin{array}{c} i \\ j \end{array} \right) \frac{n!(t-\tau)^{p-i+j-1}(\tau-a)^{n-j}}{(n-j)!(p-i+j-1)!} \\
&+\sum_{i+p}^{n-1} b_i \sum_{j=1}^{p} (-1)^{j+i-p+1} \left(\begin{array}{c} i \\ i-p+j \end{array} \right) \frac{n!(t-\tau)^{j-1}(\tau-a)^{n-i+p-j}}{(n-i+p-j)!(j-1)!}
\end{aligned}
\tag{3.14}
$$

$$\begin{aligned} K_{B,k,y}(n,p,t,\tau) = &\sum_{j=1}^{p} \begin{pmatrix} n \\ n-p+j \end{pmatrix} \frac{n!(t-\tau)^{j-1}(\tau-a)^{p-j}}{(p-j)!(j-1)!} \\ &+ \sum_{i=0}^{p-1} a_i \sum_{j=0}^{i} \begin{pmatrix} i \\ j \end{pmatrix} \frac{n!(t-\tau)^{p-i+j-1}(b-\tau)^{n-j}}{(n-j)!(p-i+j-1)!} \\ &+ \sum_{i=p}^{n-1} a_i \sum_{j=1}^{p} \begin{pmatrix} i \\ i-p+j \end{pmatrix} \frac{n!(t-\tau)^{j-1}(b-\tau)^{n-i+p-j}}{(n-i+p-j)!(j-1)!} \end{aligned} \tag{3.15}$$

$$\begin{aligned} K_{B,k,u}(n,p,t,\tau) = &\sum_{i=0}^{p-1} b_i \sum_{j=0}^{i} \begin{pmatrix} i \\ j \end{pmatrix} \frac{n!(t-\tau)^{p-i+j-1}(b-\tau)^{n-j}}{(n-j)!(p-i+j-1)!} \\ &+ \sum_{i+p}^{n-1} b_i \sum_{j=1}^{p} \begin{pmatrix} i \\ i-p+j \end{pmatrix} \frac{n!(t-\tau)^{j-1}(b-\tau)^{n-i+p-j}}{(n-i+p-j)!(j-1)!} \end{aligned} \tag{3.16}$$

3.2 Kernel representation of a 4^{th} order system

Consider a fourth order homogeneous LTI system described by (3.17) consisting of four unknown parameters that describe the system dynamics - a_0, a_1, a_2, a_3.

$$y^{(4)}(t) + a_3 y^{(3)}(t) + a_2 y^{(2)}(t) + a_1 y^{(1)}(t) + a_0 y(t) = 0, \quad \forall t \in [a,b] \tag{3.17}$$

The primary idea fuelling Theorem 3.1.1 and 3.1.2 was to decrease the order of the output derivatives in (3.17) until no derivatives appeared. Pre-multiplying the *annihilator functions* assisted in eradicating the influence of any pre-existing initial conditions that vanished together with their derivatives at the endpoints. Thus, by pre-multiplying (3.17) with $(\epsilon - a)^4$ and $(b-\zeta)^4$ (annihilator at $\epsilon = a$ and $\zeta = b$) to get

$$(\varepsilon - a)^4 y^{(4)}(t) + a_3(\varepsilon - a)^4 y^{(3)}(t) + a_2(\varepsilon - a)^4 y^{(2)}(t)_+ a_1(\varepsilon - a)^4 y^{(1)}(t) + a_0(\varepsilon - a)^4 y(t) = 0 \tag{3.18}$$

$$(b-\zeta)^4 y^{(4)}(t) + a_3(b-\zeta)^4 y^{(3)}(t) + a_2(b-\zeta)^4 y^{(2)}(t) + a_1(b-\zeta)^4 y^{(1)}(t) + a_0(b-\zeta)^4 y(t) = 0 \tag{3.19}$$

Summarizing the detailed derivation in [41], equations (3.18) and (3.19) are integrated four times on the intervals $[a, a+\tau]$ and $[b-\sigma, b]$. This results in an integration in the forward direction for the interval $[a, a+\tau]$ and in the backward direction for $[b-\sigma, b]$. This results

in the following equations for the forward and backward kernels -

$$\begin{aligned}\tau^4 y(a+\tau) &= \int_a^{a+\tau} \left[16\left(\varepsilon'''-a\right)^3 - a_3\left(\varepsilon'''-a\right)^4\right] y\left(\varepsilon'''\right) \mathrm{d}\varepsilon''' \\ &+ \int_a^{a+\tau}\int_a^{\varepsilon'''} \left[-72\left(\varepsilon''-a\right)^2 + 12a_3\left(\varepsilon''-a\right)^3 - a_2\left(\varepsilon''-a\right)^4\right] y\left(\varepsilon''\right) \mathrm{d}\varepsilon''\mathrm{d}\varepsilon''' \\ &+ \int_a^{a+\tau}\int_a^{\varepsilon'''}\int_a^{\varepsilon''} \left[96\left(\varepsilon'-a\right) - 36a_3\left(\varepsilon'-a\right)^2 + 8a_2\left(\varepsilon'-a\right)^3 - a_1\left(\varepsilon'-a\right)^4\right] y\left(\varepsilon'\right) \mathrm{d}\varepsilon'\mathrm{d}\varepsilon''\mathrm{d}\varepsilon''' \\ &+ \int_a^{a+\tau}\int_a^{\varepsilon'''}\int_a^{\varepsilon''}\int_a^{\varepsilon'} \left[-24 + 24a_3(\varepsilon-a) - 12a_2(\varepsilon-a)^2 + 4a_1(\varepsilon-a)^3 - a_0(\varepsilon-a)^4\right] y(\varepsilon)\mathrm{d}\varepsilon\mathrm{d}\varepsilon'\mathrm{d}\varepsilon''\mathrm{d}\varepsilon'''\end{aligned} \tag{3.20}$$

$$\begin{aligned}\sigma^4 y(b-\sigma) &= \int_b^{b-\sigma} \left[-16\left(b-\zeta'''\right)^3 - a_3\left(b-\zeta'''\right)^4\right] y\left(\zeta'''\right) \mathrm{d}\zeta''' \\ &+ \int_b^{b-\sigma}\int_b^{\zeta'''} \left[-72\left(b-\zeta''\right)^2 - 12a_3\left(b-\zeta''\right)^3 - a_2\left(b-\zeta''\right)^4\right] y\left(\zeta''\right) \zeta''\zeta''' \\ &+ \int_b^{b-\sigma}\int_b^{\zeta'''}\int_b^{\zeta''} \left[-96\left(b-\zeta'\right) - 36a_3\left(b-\zeta'\right)^2 - 8a_2\left(b-\zeta'\right)^3 - a_1\left(b-\zeta'\right)^4\right] y\left(\zeta'\right) \mathrm{d}\zeta'\mathrm{d}\zeta''\mathrm{d}\zeta''' \\ &+ \int_b^{b-\sigma}\int_b^{\zeta'''}\int_b^{\zeta''}\int_b^{\zeta'} \left[-24 - 24a_3(b-\zeta) - 12a_2(b-\zeta)^2 - 4a_1(b-\zeta)^3 - a_0(b-\zeta)^4\right] y(\zeta)\mathrm{d}\zeta\mathrm{d}\zeta'\mathrm{d}\zeta''\mathrm{d}\zeta'''\end{aligned} \tag{3.21}$$

In order to simplify the repeated integrations above, Cauchy's formula [42] is applied, which is defined as -

If f is a continuous function on the real line then, the n^{th} repeated integral of f at a can be given by,

$$f^{(-n)}(t) = \int_a^t \int_a^{\sigma_1} \cdots \int_a^{\sigma_{n-1}} f\left(\sigma_n\right) \mathrm{d}\sigma_n \cdots \mathrm{d}\sigma_2 \; \mathrm{d}\sigma_1 \tag{3.22}$$

which is equivalent to a single integration.

$$f^{(-n)}(t) = \frac{1}{(n-1)!}\int_a^t (t-s)^{n-1} f(s)\mathrm{d}s \tag{3.23}$$

Let $a+\tau = t$ in (3.20), $b-\sigma = t$ in (3.21) and apply Cauchy's formula in the forward direction for the interval $[a, a+\tau]$ and backward direction for the interval $[b, b-\sigma]$ to get,

$$\alpha_a(t)y(t) \triangleq \int_a^t K_{F,y}(t,s)y(s)ds; \quad \text{where } \alpha_a(t) = (t-a)^4 \tag{3.24}$$

$$\alpha_b(t)y(t) \triangleq \int_t^b K_{B,y}(t,s)y(s)ds; \quad \text{where } \alpha_b(t) = (b-t)^4 \tag{3.25}$$

with $K_{F,y}(t,s)$ as,

$$\begin{aligned} K_{F,y}(t,s) = &\Big[16(s-a)^3 - a_3(s-a)^4\Big] + \frac{(t-s)^1}{1!}\Big[-72(s-a)^2 + 12a_3(s-a)^3 - a_2(s-a)^4\Big] \\ &+ \frac{(t-s)^2}{2!}\Big[96(s-a) - 36a_3(s-a)^2 + 8a_2(s-a)^3 - a_1(s-a)^4\Big] \\ &+ \frac{(t-s)^3}{3!}\Big[-24 + 24a_3(s-a) - 12a_2(s-a)^2 + 4a_1(s-a)^3 - a_0(s-a)^4\Big] \end{aligned} \tag{3.26}$$

and $K_{B,y}(t,s)$ as,

$$\begin{aligned} K_{B,y}(t,s) = &\Big[16(b-s)^3 + a_3(b-s)^4\Big] + \frac{(t-s)}{1!}\Big[72(b-s)^2 + 12a_3(b-s)^3 + a_2(b-s)^4\Big] \\ &+ \frac{(t-s)^2}{2!}\Big[96(b-s) + 36a_3(b-s)^2 + 8a_2(b-s)^3 + a_1(b-s)^4\Big] \\ &+ \frac{(t-s)^3}{3!}\Big[24 + 24a_3(b-s) + 12a_2(b-s)^2 + 4a_1(b-s)^3 + a_0(b-s)^4\Big] \end{aligned} \tag{3.27}$$

The expressions for kernel representation of order $n = 1, 2, 3$ can be found in appendix A of [25].

3.3 Multiple regression equations [24]

Integrating (3.20) and (3.21) multiple times using the Cauchy's formula, yields the multiple regression equations. Shown below is the detailed derivation for the forward kernel, taken from [24].

$$\int_a^t (s-a)^4 y(s)ds = \int_a^t \int_a^s \left[16\left(\varepsilon'''-a\right)^3 - a_3\left(\varepsilon'''-a\right)^4\right] y\left(\varepsilon'''\right) \mathrm{d}\varepsilon''' ds$$

$$+ \int_a^t \int_a^s \int_a^{\varepsilon'''} \left[-72\left(\varepsilon''-a\right)^2 + 12a_3\left(\varepsilon''-a\right)^3 - a_2\left(\varepsilon''-a\right)^4\right] y\left(\varepsilon''\right) \mathrm{d}\varepsilon'' \mathrm{d}\varepsilon''' ds$$

$$+ \int_a^t \int_a^s \int_a^{\varepsilon'''} \int_a^{\varepsilon''} \left[96\left(\varepsilon'-a\right) - 36a_3\left(\varepsilon'-a\right)^2 + 8a_2\left(\varepsilon'-a\right)^3 - a_1\left(\varepsilon'-a\right)^4\right] y\left(\varepsilon'\right) \mathrm{d}\varepsilon' \mathrm{d}\varepsilon'' \mathrm{d}\varepsilon''' ds$$

$$+ \int_a^t \int_a^s \int_a^{\prime\prime\prime} \int_a^{\prime\prime} \int_a^{\varepsilon'} \left[-24 + 24a_3(\varepsilon-a) - 12a_2(\varepsilon-a)^2 + 4a_1(\varepsilon-a)^3 - a_0(\varepsilon-a)^4\right] y(\varepsilon) \mathrm{d}\varepsilon \mathrm{d}\varepsilon' \mathrm{d}\varepsilon'' \mathrm{d}\varepsilon''' ds \tag{3.28}$$

$$\begin{aligned}
&= \int_a^t \frac{(t-s)^1}{1!} \left[16(s-a)^3 - a_3(s-a)^4\right] y(s)ds \\
&+ \int_a^t \frac{(t-s)^2}{2!} \left[-72(s-a)^2 + 12a_3(s-a)^3 - a_2(s-a)^4\right] y(s)ds \\
&+ \int_a^t \frac{(t-s)^3}{3!} \left[96(s-a) - 36a_3(s-a)^2 + 8a_2(s-a)^3 - a_1(s-a)^4\right] y(s)ds \\
&+ \int_a^t \frac{(t-s)^4}{4!} \left[-24 + 24a_3(s-a) - 12a_2(s-a)^2 + 4a_1(s-a)^3 - a_0(s-a)^4\right] y(s)ds
\end{aligned} \tag{3.29}$$

Equation (3.29) forms the base equation, which on further integration(s) engenders the following formula of the forward kernel for a 4^{th} order system *(for the k^{th} order of integration)*,

$$
\begin{aligned}
&\frac{1}{(k-1)!}\int_a^t \alpha_a(t,s)(t-s)^{k-1}y(s)ds \\
&= \int_a^t \frac{(t-s)^k}{k!}\Big[16(s-a)^3 - a_3(s-a)^4\Big]y(s)ds \\
&+ \int_a^t \frac{(t-s)^{k+1}}{(k+1)!}\Big[-72(s-a)^2 + 12a_3(s-a)^3 - a_2(s-a)^4\Big]y(s)ds \\
&+ \int_a^t \frac{(t-s)^{k+2}}{(k+2)!}\Big[96(s-a) - 36a_3(s-a)^2 + 8a_2(s-a)^3 - a_1(s-a)^4\Big]y(s)ds \\
&+ \int_a^t \frac{(t-s)^{k+3}}{(k+3)!}\Big[-24 + 24a_3(s-a) - 12a_2(s-a)^2 + 4a_1(s-a)^3 - a_0(s-a)^4\Big]y(s)ds
\end{aligned}
\tag{3.30}
$$

Equation (3.30) can also be represented concisely as

$$
\frac{1}{(k-1)!}\int_a^t \alpha_a(t,s)(t-s)^{k-1}y(s)ds = \int_a^t K_{F_k,y}(t,s)y(s)ds \quad \text{for } k = 1,..,m \tag{3.31}
$$

Similarly, the expression for the backward kernel can be given as

$$
\begin{aligned}
&\frac{1}{(k-1)!}\int_t^b \alpha_b(t,s)(t-s)^{k-1}y(s)ds \qquad\qquad \text{for } k = 1,2,3,4 \\
&= -\int_t^b \frac{(t-s)^k}{k!}\Big[16(b-s)^3 + a_3(b-s)^4\Big]y(s)ds \\
&\quad -\int_t^b \frac{(t-s)^{k+1}}{k+1!}\Big[72(b-s)^2 + 12a_3(b-s)^3 + a_2(b-s)^4\Big]y(s)ds \\
&\quad -\int_t^b \frac{(t-s)^{k+2}}{k+2!}\Big[96(b-s) + 36a_3(b-s)^2 + 8a_2(b-s)^3 + a_1(b-s)^4\Big]y(s)ds \\
&\quad -\int_t^b \frac{(t-s)^{k+3}}{k+3!}\Big[24 + 24a_3(b-s) + 12a_2(b-s)^2 + 4a_1(b-s)^3 + a_0(b-s)^4\Big]y(s)ds
\end{aligned}
\tag{3.32}
$$

And (3.32) can be re-written as

$$
\frac{1}{(k-1)!}\int_t^b \alpha_b(t,s)(t-s)^{k-1}y(s)ds = \int_t^b K_{B_k,y}(t,s)y(s)ds \quad \text{for } k = 1,..,m \tag{3.33}
$$

Summing up (3.31) and (3.33) gives,

$$\frac{1}{(k-1)!}\int_a^b \alpha_{ab}(t,s)(t-s)^{k-1}y(s)ds = \int_a^b K_{DS_k,y}(t,s)y(s)ds \tag{3.34}$$

where,

$$K_{DS_k,y}(t,s) \triangleq \begin{cases} K_{F_k,y}(t,s) : s \leq t \\ K_{B_k,y}(t,s) : s > t \end{cases} ; \alpha_{ab}(t,s) \triangleq \begin{cases} (t-a)^4 : s \leq t \\ (b-t)^4 : s > t \end{cases} \tag{3.35}$$

3.4 Parameter estimation for n$^{\text{th}}$ order system [24]

The multiple regression equation for a 4^{th} order system can be generalized for n^{th} order systems, taking into account the kernel definitions from Theorem 3.1.1. Multiple regression equations for $k = 1, ..., m(m \geq n)$ can be written as -

$$\int_a^b \alpha_{ab}(s)y(s)ds = \sum_{i=0}^{n} \beta_i \int_a^b K_{DS_1(i),y}(t,s)y(s)ds \tag{3.36}$$

$$\int_a^b \alpha_{ab}(s)(t-s)y(s)ds = \sum_{i=0}^{n} \beta_i \int_a^b K_{DS_2(i),y}(t,s)y(s)ds \tag{3.37}$$

$$\frac{1}{2}\int_a^b \alpha_{ab}(s)(t-s)^2y(s)ds = \sum_{i=0}^{n} \beta_i \int_a^b K_{DS_3(i),y}(t,s)y(s)ds \tag{3.38}$$

$$\vdots$$

$$\frac{1}{(m-1)!}\int_a^b \alpha_{ab}(s)(t-s)^{m-1}y(s)ds = \sum_{i=0}^{n} \beta_i \int_a^b K_{DS_m(i),y}(t,s)y(s)ds \tag{3.39}$$

In a noise-free deterministic setting, the output variable y becomes the measured output coinciding with the nominal output y_T. With $\bar{a} := [a_0; \cdots ; a_{n-1}]$ and $\beta := [a_0; \cdots ; a_{n-1}; a_n] = [\bar{a}; 1]$, let $K_{DS_k(\bar{a})}(t, y_T)$ be row vectors with integral components

$$K_{DS_k(\bar{a})}(t, y_T) := \left[\int_a^b K_{DS_k(0),y}(t,s)y_T(s)ds, \cdots , \int_a^b K_{DS_k(n-1),y}(t,s)y_T(s)ds\right] \tag{3.40}$$

and $K_{DS_{k(a_n)}}(t, y_T)$ be scalars

$$K_{DS_{k(a_n)}}(t, y_T) := \int_a^b K_{DS_{k(n),y}}(t, s) y_T(s) ds \tag{3.41}$$

corresponding to $\beta_n := a_n = 1$.

Rearranging (3.36) - (3.39)

$$\begin{bmatrix} \int_a^b \alpha_{ab}(s) y(s) ds - K_{DS_{1(a_n)}}(t, y_T) \\ \int_a^b \alpha_{ab}(s)(t-s) y(s) ds - K_{DS_{2(a_n)}}(t, y_T) \\ \vdots \\ \frac{1}{(m-1)!} \int_a^b \alpha_{ab}(s)(t-s)^{m-1} y(s) ds - K_{DS_{m(a_n)}}(t, y_T) \end{bmatrix} = \begin{bmatrix} K_{DS_{1(\bar{a})}}(t, y_T) \\ K_{DS_{2(\bar{a})}}(t, y_T) \\ \vdots \\ K_{DS_{m(\bar{a})}}(t, y_T) \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix} \tag{3.42}$$

for $k = 1, \ldots, m (m \geq n)$

Distinct time instants, known as *knots* can be utilized to define the following

$$q^k(t_j, y_T) = \frac{1}{(k-1)!} \int_a^b \alpha_{ab}(t_j, s) (t_j - s)^{k-1} y_T(s) ds - K_{DS_{k(a_n)}}(t_j, y_T); \tag{3.43}$$

$$\begin{aligned} p^k(t_j, y_T) = K_{DS_{k(\bar{a})}}(t_j, y_T) &:= \begin{bmatrix} \int_a^b K_{DS_{k(0),y}}(t_j, s) y_T(s) ds & \cdots & \int_a^b K_{DS_{k(n-1),y}}(t_j, s) y_T(s) ds \end{bmatrix} \\ &= \begin{bmatrix} p_0^k(t_j, y_T) & \cdots & p_{n-1}^k(t_j, y_T) \end{bmatrix} \end{aligned} \tag{3.44}$$

Thus, equation (3.42) can be re-written *knot-wise* for $k = 1, \ldots, m$ as another matrix equation

$$\begin{bmatrix} q^1(t_1,y_T) \\ \vdots \\ q^1(t_N,y_T) \\ \vdots \\ q^m(t_1,y_T) \\ \vdots \\ q^m(t_N,y_T) \end{bmatrix}_{Nm\times 1} = \begin{bmatrix} p_0^1(t_1,y_T) & p_1^1(t_1,y_T) & \cdots & p_{n-1}^1(t_1,y_T) \\ & \ddots & & \\ p_0^1(t_N,y_T) & p_1^1(t_N,y_T) & \cdots & p_{n-1}^1(t_N,y_T) \\ & \vdots & & \\ p_0^m(t_1,y_T) & p_1^m(t_1,y_T) & \cdots & p_{n-1}^m(t_1,y_T) \\ & \ddots & & \\ p_0^m(t_N,y_T) & p_1^m(t_N,y_T) & \cdots & p_{n-1}^m(t_N,y_T) \end{bmatrix}_{Nm\times n} \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}_{n\times 1} \tag{3.45}$$

which can be simplified as

$$Q(y_T) = P(y_T)\,\bar{a} \tag{3.46}$$

where $Q(y_T) \in \mathbb{R}^{Nk}, P(y_T) \in \mathbb{R}^{Nk} \times \mathbb{R}^n, \bar{a} \in \mathbb{R}^n$ and $k = 1,\ldots,m$. Thus, the matrix equation (3.45) can be solved exactly using least squares error minimization with respect to the parameter vector $\bar{a}$ provided adequate identifiability assumptions are met and the output is measured without error.

Identifiability of homogeneous LTI systems from a single realization of a measured output [38]

A homogeneous LTI system such as

$$\dot{x}(t) = Ax(t); \quad y = cx; \quad x \in \mathbb{R}^n; \quad x(0) = b \tag{3.47}$$

is identifiable from a single noise-free realization of its output trajectory y under precise conditions, which admittedly are difficult to verify computationally. This is stated in its equivalent form as

Definition: Model (3.47) *is globally identifiable from b if and only if the functional mapping* $b \mapsto y(\cdot; A, b)$ *is injective on* $\mathbb{R}^n$ *where* $y(\cdot; A, b)$ *denotes the output orbit of* (3.47) .

Theorem 3.4.1 *[43] Model* (3.47) *is globally identifiable from b if and only if the output orbit of* (3.47) *is not confined to a proper subspace of* $\mathbb{R}^n$.

The above criterion has limited use for reasons of practicality: it is difficult to verify com-

putationally, pertains to infinite time horizons $[0, \infty)$ and, most importantly, requires the output trajectory to be known exactly. For the purpose of the present exposition it hence suffices to invoke a *practical version of identifiability* as defined below.

Definition 2: Practical linear identifiability
The homogeneous system (3.47) is practically linearly identifiable on $[a, b]$ with respect to a particular noisy discrete realization of the output measurement process, $y(t), t \in [a, b]$, if and only if there exist distinct knots $t_1, \cdots, t_N \in (a, b]$ which render rank of $P(y) = n$. Any such output realization is then called *persistent*.

In practical applications the N distinct time instants needed can be placed equidistantly over the interval $(a, b]$ or else generated randomly. Since no assumptions are made about system perturbations or measurement noise, the estimation equation (3.46) is solved in terms of a pseudo-inverse P^+ of P:

$$\overline{a} = P^+(y_M)Q(y_M) \tag{3.48}$$

Parameter estimation can be conducted simultaneously with state estimation - Under the assumption of system flatness, the system states are immediately recovered as functions of the time derivatives of the output. Following parametric estimation, the output derivatives can be computed using the recursive kernels in Theorem (3.1.2).

3.5 Calculating the error covariance matrix [38] [24]

It is almost impossible to have a system or a signal that is free from noise, therefore, it is important to consider the implications of the presence of measurement noise. For solving this problem, the noise involved for all purposed is assumed to be additive while Gaussian noise *(AWGN)*. It is easy to realize that the regression equations in section 3.4 would no longer be valid as the reproducing property fails to trace an erroneous output trajectory. This leads to a stochastic regression problem. The stochastic output measurement process, $y_M(t)$ adapted to the natural filtration of the standard Wiener process W on $[a, b]$ is

$$y_M(t, \omega) = y_T(t) + \sigma \dot{W}(t, \omega) \; ; \quad t \in [a, b] \tag{3.49}$$

where $\sigma \dot{W}$ signifies the generalized derivative of the standard Wiener process, identified with a white noise process of constant variance σ^2, where y_T is the true system output [44].

Expectation and covariance functions of white noise are given as

$$E[\dot{W}(t)] = 0 \tag{3.50}$$

$$Cov[\dot{W}(t)\dot{W}(s)] = E[\dot{W}(t)\dot{W}(s)] = \delta(t-s) \tag{3.51}$$

$$Var[\dot{W}(t)] = E(\dot{W}(t))^2 = 1 \quad t, s \in [a, b] \tag{3.52}$$

where δ is the Dirac delta distribution but acting on a square integrable functions as an evaluation functional:

$$\int_a^b g(s)\delta(t-s)ds = g(t) \tag{3.53}$$

Using equation (3.49) to re-write the kernel expression for $k = 1, \ldots, m$ gives

$$\int_a^b K_{DS_k,y}(t,s)y_M(s)ds = \int_a^b K_{DS_k,y}(t,s)y_T(s)\ ds + \int_a^b K_{DS_k,y}(t,s)\sigma\dot{W}(s)\ ds \tag{3.54}$$

The stochastic regression equation is given by

$$\frac{1}{(k-1)!}\int_a^b \alpha_{ab}(t,s)(t-s)^{k-1}y_M(s)ds = \int_a^b K_{DS_k,y}(t,s)y_M(s)\ ds + e(t) \tag{3.55}$$

which has the random regressor vector

$$\left[\int_a^b K_{DS_{k(a_0)},y}(t,s)y_M(s)ds, \cdots, \int_a^b K_{DS_{k(a_n)},y}(t,s)y_M(s)ds\right]^T \tag{3.56}$$

The assumptions of the Gauss-Markov Theorem are violated in the linear regression problem (3.55) because the random regressor is correlated with a regression error, which additionally fails to be homoskedastic. The above regression is thus a typical 'error-in-the-variable' problem with heteroskedastic noise which has been tackled using the instrumental variable (IV) approach adopted by [45]. The cons out-weight the pros for this approach and because of this, the multiple regression equation approach was considered to be the better choice.

One of the most common ways to tackle unknown heteroskedasticity is to utilize a BLUE (Best Linear Unbiased Estimator), specifically GLS (Generalized Least Squares). It leverages inverse covariance weighting in the regression error minimization problem. Let $Q(y_M)$ and $P(y_M)$ be the matrices corresponding to N samples of the measurement process realization y_M at a batch of knots $t_1, t_2, ..., t_N$. The matrix regression equation (3.46) can be re-written

for $k = 1, \ldots, m$ as

$$Q(y_M) = P(y_M)\overline{a} + e \tag{3.57}$$

where

$$e := \begin{bmatrix} e^1(t_1) \\ \vdots \\ e^1(t_N) \\ \vdots \\ e^m(t_1) \\ \vdots \\ e^m(t_N) \end{bmatrix}_{Nm\times 1} \tag{3.58}$$

with

$$e^k(t_j) := \frac{\sigma}{(k-1)!}\int_a^b \alpha_{ab}(t_j, s)(t_j - s)^{k-1}\dot{W}(s)ds - \sigma\int_a^b K_{DS_k,y}(t_j, s)\dot{W}(s)ds; k = 1, \ldots, m \tag{3.59}$$

The error minimization problem given in (3.57) is solved using a Regularized Least Squares (RLS). The standard regression error minimization of the parameter vector $\overline{a}$ is

$$\min_{\bar{a}} \left((\bar{a} - \bar{a}_0)^T W_0^{-1}(\bar{a} - \bar{a}_0) + (Q(y_M) - P(y_M)\bar{a})^T S(Q(y_M) - P(y_M)\bar{a}) \right) \tag{3.60}$$

where W_0 is a given positive - definite matrix that is utilized as a penalty matrix, initialized with a calculated guess, $\overline{a_0}$ is a given parameter vector (which would also be initialized with a calculated guess) and $S \in \mathbb{R}^{N_k \times N_k}$ is the weighing matrix defined as $S := \text{diag}(S_1, ..., S_k)$ for $k = 1, \ldots, m$ and $S_k \in \mathbb{R}^{N\times N}$ are the inverses of the corresponding error covariance matrices, as defined below:

$$[S_k]^{-1} := \begin{bmatrix} \text{Cov}[e^k(t_1), e^k(t_1)] \cdots \text{Cov}[e^k(t_1), e^k(t_N)] \\ \ddots \\ \text{Cov}[e^k(t_N), e^k(t_1)] \cdots \text{Cov}[e^k(t_N), e^k(t_N)] \end{bmatrix}; \quad k = 1, \ldots, m \tag{3.61}$$

Based on (3.50) - (3.52) and the fact that kernel functions are Hilbert-Schmidt (which makes them square integrable), the covariance matrix for a general n^{th} order system with $k =$

$1, \ldots, m (m \geq n)$ [24] can be written as

$$
\begin{aligned}
&\mathrm{Cov}[e^k(t_i), e^k(t_j)] = E[e^k(t_i) e^k(t_j)] \\
&= \sigma^2 E\Big[\Big[\int_a^b \frac{1}{(k-1)!}\alpha_{ab}(\tau)(t_i-\tau)^{k-1}\dot{W}(\tau)d\tau - \int_a^b K_{DS_k,y}(t_i,\tau)\dot{W}(\tau)d\tau\Big] \\
&\qquad \Big[\int_a^b \frac{1}{(k-1)!}\alpha_{ab}(s)(t_j-s)^{k-1}\dot{W}(s)ds - \int_a^b K_{DS_k,y}(t_j,s)\dot{W}(s)ds\Big]\Big] \\
&= \sigma^2 E\Big[\frac{1}{((k-1)!)^2}\int_a^b\int_a^b \alpha_{ab}(\tau)\alpha_{ab}(s)(t_i-\tau)^{k-1}(t_j-s)^{k-1}\dot{W}(\tau)\dot{W}(s)d\tau ds\Big] \\
&\qquad - E\Big[\frac{1}{(k-1)!}\int_a^b\int_a^b \alpha_{ab}(\tau)(t_i-\tau)^{k-1}\dot{W}(\tau)K_{DS_k,y}(t_j,s)\dot{W}(s)d\tau ds\Big] \\
&\qquad - E\Big[\frac{1}{(k-1)!}\int_a^b\int_a^b \alpha_{ab}(s)(t_j-s)^{k-1}\dot{W}(s)K_{DS_k,y}(t_i,\tau)\dot{W}(\tau)ds d\tau\Big] \\
&\qquad + E\Big[\int_a^b\int_a^b K_{DS_k,y}(t_i,\tau)K_{DS_k,y}(t_j,s)\dot{W}(\tau)\dot{W}(s)d\tau ds\Big] \\
&= \frac{\sigma^2}{((k-1)!)^2}\int_a^b\int_a^b \alpha_{ab}(\tau)\alpha_{ab}(s)(t_i-\tau)^{k-1}(t_j-s)^{k-1}E\Big[\dot{W}(\tau)\dot{W}(s)\Big]d\tau ds \\
&\qquad - \frac{\sigma^2}{(k-1)!}\int_a^b\int_a^b \alpha_{ab}(\tau)(t_i-\tau)^{k-1}K_{DS_k,y}(t_j,s)E\Big[\dot{W}(\tau)\dot{W}(s)\Big]d\tau ds \\
&\qquad - \frac{\sigma^2}{(k-1)!}\int_a^b\int_a^b \alpha_{ab}(s)(t_j-s)^{k-1}K_{DS_k,y}(t_i,\tau)E\Big[\dot{W}(s)\dot{W}(\tau)\Big]ds d\tau \\
&\qquad + \sigma^2\int_a^b\int_a^b K_{DS_k,y}(t_i,\tau)K_{DS_k,y}(t_j,s)E\Big[\dot{W}(\tau)\dot{W}(s)\Big]d\tau ds \\
&= \frac{\sigma^2}{((k-1)!)^2}\int_a^b \alpha_{ab}(s)\alpha_{ab}(s)(t_i-s)^{k-1}(t_j-s)^{k-1}ds \\
&\qquad - \frac{\sigma^2}{(k-1)!}\int_a^b \alpha_{ab}(s)(t_i-s)^{k-1}K_{DS_k,y}(t_j,s)ds \\
&\qquad - \frac{\sigma^2}{(k-1)!}\int_a^b \alpha_{ab}(s)(t_j-s)^{k-1}K_{DS_k,y}(t_i,s)ds + \sigma^2\int_a^b K_{DS_k,y}(t_i,s)K_{DS_k,y}(t_j,s)ds
\end{aligned}
$$

3.6 Modified Regularized Least Squares [36] [24]

The covariance matrix derived above depends on the two unknown quantities in the K_{DS} kernels - the variance σ^2 and the parameter vector $\bar{a}$. To cater to this, a feasible, modified version of the least squares algorithm is used where the covariance matrix is estimated

progressively as more data points are collected via the multiple regression equations. This is done in a recursive fashion where consecutive batches of samples are picked up from y_M. The (quadratic) cost function, in terms of $\bar{a}$ is given by

$$J(\bar{a}) = (\bar{a} - \bar{a}_0)^T W_0^{-1} (\bar{a} - \bar{a}_0) + \|Q - P\bar{a}\|_S^2 \tag{3.62}$$

Equation (3.62) ensures that there's a unique solution to this problem, even when the matrix P is not full rank. When P is full rank, including $(\bar{a} - \bar{a}_0)^T W_0^{-1} (\bar{a} - \bar{a}_0)$ can improve the condition number of the matrix resulting in better numerical behavior. The solution to (3.62) is of the form

$$\bar{a} = (W_0^{-1} + P^T S P)^{-1} P^T S Q \tag{3.63}$$

Equation (3.63) becomes computationally expensive and time consuming as the number of measurements increase since the measurements are obtained sequentially. To circumvent this problem, the recursive form of the least squares problem is considered.

Instead of assuming $\bar{a_0}$ to simply be 0, the algorithm calculates a rough OLS estimate for the parameters, using 3.48 which enables stronger noise rejection and accurate parameter estimates. At iteration j, the minimization function can be written as

$$\min_{\bar{a}} \left[\bar{a}^T W_0^{-1} \bar{a} + \|\bar{Q}_j - \bar{P}_j \bar{a}\|_{S_j}^2 \right] \tag{3.64}$$

where the following terms can be defined for $k = 1, \ldots, m$

$$\bar{Q}_j = \begin{bmatrix} Q_0 \\ Q_1 \\ \vdots \\ Q_j \end{bmatrix}; \quad \bar{P}_j = \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_j \end{bmatrix}; \quad \text{with } Q_j = \begin{bmatrix} q_j^1(y_M) \\ \vdots \\ q_j^k(y_M) \end{bmatrix} \text{ and } P_j = \begin{bmatrix} p_j^1(y_M) \\ \vdots \\ p_j^k(y_M) \end{bmatrix} \tag{3.65}$$

and

$$\bar{S}_j = \text{diag}(S_0, S_1, \ldots, S_j); \quad \text{with } S_j = \text{diag}(S_{1_j}, \ldots, S_{k_j}) \tag{3.66}$$

The following section gives the actual recursive steps of the algorithm that was used in this

thesis and for the repository that was built as part of the project. The detailed derivation of the basic RLS solution has been given in [36].

3.6.1 Modified Recursive Regularized Least Squares Algorithm (MRRLS)

- Initialize the estimator:

$$\bar{a}_0 = OLS(\cdot)$$
$$W_0 = \delta I$$

  The parameters are initiated with a rough OLS estimate calculated based on the input (noisy) signal. In case of no prior knowledge about parameters, simply let $W_0 \approx \infty I$. In the case of perfect prior knowledge, $W_0 = 0$. In this thesis, W_0 was initialized as 10^6 I

- Iterate the following two steps.
  (a) Obtain a new batch of knot points (measurements) and calculate the Q_j, P_j and S_j matrices
  (b) Update the estimate $\hat{a}$ and the covariance of the estimation error as per the following equations

$$K_j = W_{j-1}P_j^T(P_jW_{j-1}P_j^T + S_j^{-1})^{-1} \tag{3.67}$$
$$W_j = (I - K_jP_j)W_{j-1} \tag{3.68}$$
$$\hat{a}_j = \hat{a}_{j-1} + K_j(Q_j - P_j\hat{a}_{j-1}) \tag{3.69}$$

  The initial estimate of S_j^{-1} is calculated as the *empirical* variance of $(y_M - y_E)$ where y_E is the estimated output corresponding to the parameter values obtained in iteration $j = 0$. This is updated at each consecutive iteration by the same empirical method until the difference in parameter values converges below a set threshold. This thesis often refers to this algorithm as the 'kernel' algorithm/ method for better clarity for the reader.

3.7 Reconstructing output and derivative trajectories [41]

One of the crucial tasks after the parameters are successfully estimated is to reconstruct the system output and its derivatives. There are multiple methods to do that, the most

common method is to use Kalman filters as mentioned in Chapter 2, algorithm 1. The other is a novel method which involves reconstruction by projection onto the finite dimensional subspace of the RKHS spanned by the fundamental solutions of the characteristic equation of the system, denoted by $\xi_1, \cdots, \xi_n$. Since every solution of the characteristic equation with the estimated parameter vector $\bar{a}$ will satisfy the reproducing property, the projection onto the space of fundamental solutions will be the noise free trajectory of the system.

The fundamental solutions are calculated by integrating the characteristic equations for n initial conditions

$$Y(0)_k := \left[y(0), y^{(1)}(0), \cdots, y^{(n-1)}(0)\right] = e_k; \quad k = 1, \cdots, n \tag{3.70}$$

where e_k are the canonical basis vectors in $\mathbb{R}^n$ i.e,

$$\begin{aligned} e_1 &= [1, 0, \ldots, 0] \\ e_2 &= [0, 1, \ldots, 0] \\ &\vdots \\ e_n &= [0, 0, \ldots, 1] \end{aligned} \tag{3.71}$$

For computational efficiency, it becomes crucial to ortho-normalize the set ξ_k for $k = 1, \cdots, n$ into ζ_k for $k = 1, \cdots, n$ using the Gram-Schmidt ortho-normalization procedure in L^2 over $(a, b]$. The ortho-normalizing procedure is a linear transformation of the set of the fundamental solutions with

$$\begin{aligned} &\text{span}\left\{\xi_k, k = 1, \cdots, n\right\} = \text{span}\left\{\zeta_k, k = 1, \cdots, n\right\} \\ &\langle \zeta_i \mid \zeta_j \rangle_2 = 0 \text{ for } i \neq j; \quad \langle \zeta_i \mid \zeta_i \rangle_2 = 1 \end{aligned} \tag{3.72}$$

where $\langle \cdot \mid \cdot \rangle_2$ denotes the inner product in L^2. Since the noise-free output to be estimated is a linear combination of fundamental solutions

$$y_T = \sum_{i=1}^{n} c_i \zeta_i \text{ with } c_i = \langle y_T \mid \zeta_i \rangle_2, \quad i = 1, \cdots, n \tag{3.73}$$

Considering a similar form for the estimator $\hat{y}_M$ with linear estimators $\hat{c}_i$ for the coefficients c_i in the form

$$\hat{c}_i = \langle y_M \mid \zeta_i \rangle_2 = \int_a^b y_M(s)\zeta_i(s)ds, \quad i = 1, \cdots, n \tag{3.74}$$

Therefore, given a measurement process realization y_M on $[a, b]$, the reconstructed output trajectory is obtained as

$$y_E(t) = \sum_{i=1}^{n} \langle y_M \mid \zeta_i \rangle_2 \, \zeta_i(t); \quad t \in [a, b] \tag{3.75}$$

3.7.1 Reconstruction of output derivatives

Leveraging Theorem 3.1.2, the estimated output $y_E(t)$ can be used to reconstruct the derivatives $y_E^{(i)}, \ i = 1, \cdots, n$ using

$$y_E^{(i)}(t) = \int_a^b K_{DS}^i(t, \tau) y_E(\tau) \, d\tau \quad i = 1, \ldots, n-1 \tag{3.76}$$

where $K_{DS}^i, i = 1, \ldots, n-1$ are the kernel representation for the derivatives. This thesis often refers to this method as the 'projection' method for better clarity for the reader.

3.8 Discussions

Before moving onto the next chapter which compares the ability to accurately obtain outputs and their derivative trajectories, it is imperative that certain results from prior research be re-stated here. These findings define a set of parameter selection best practices, including model order, data size, regression order, and knot point selection. Extensive results are published in [24] but the following paragraphs summarize all these findings which would assist in selecting the best set of parameters for the examples shown in this thesis.

Model selection: The kernel - multiple regression method presented in this chapter does not assume model order as one of the pre-requisites for the process. Metrics such as Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) were used to evaluate different candidate models of various orders to find the one that performed the best. It was found that when the order of the system (which is unknown to the algorithm) and the order of the kernels used were the same, the BIC score was the least - this also established BIC as a robust method to estimate the order of an unknown system, if required.

Data sample size: Systems with a varying range of external noise were studied to identify the relation between accuracy and sample size. It was found out that increasing noise variance requires a higher number of data sample points for an accurate reconstruction. However, there is a trade-off between the increases accuracy and time complexity - therefore, the exact number of data points required for estimation needs to be decided on a case by case basis.

Integration/regression order: This experiment helped in studying the effects of over-parametrization and the exact integration orders used in the multiple regression equations. While the results did not point towards anything conclusive, keeping the number of equations equal to the order of the system and using integration equations $k = 1, \ldots, n$ where n is the order of the equation gave accurate values (low deviation values from the ground truth).

Knot point selection: This experiment engendered an interesting outcome in terms of the method to select knot points. After testing various systems with varying noise levels, the best way to select knot points for accurate and low BIC scores was to select 70% of the points in the middle and 30% of the points across the horizon. However, for the extent of this thesis, all examples provided in the next few chapters will take 100% of the knot points at random across the horizon to maintain equity between all systems.

Chapter 4

State Estimation with Known Calculated Parameters

This chapter focuses on examining how the two major algorithms perform on actual systems now that the theory behind them have been established. The first step in building a framework for the estimating (and filtering) of SISO LTI systems is to evaluate and contrast state estimation methods. Specifically, different types of systems with varying degrees of noise are investigated and employed for state estimation over a finite interval $[a, b]$. Additive White Gaussian Noise (AWGN) is overlaid on the signal to simulate this.

One crucial point in this set of experiments is that it is assumed that the parameters of the system dynamics are known - the parameters used in this chapter are all calculated based on the rectified forward-backward kernel algorithm that has been described in chapter 3, section 3.6. In essence, this chapter compares how effective the state reconstruction abilities of the projection method (as described in section 3.7 in chapter 3) versus the Kalman + RTS algorithm (as described in section 2.4 in chapter 2) are. This will then be followed by simultaneous system parameter and state estimation where there is no apriori knowledge of the dynamics at all, in the next chapter.

To quantify the effectiveness of these algorithms, two major error metrics have been calculated throughout the thesis across all experiments -

1. Root Mean Squared Error (RMSE) [46]:

$$RMSE_{y(t)} := \sqrt{\frac{1}{m}\sum_{j=0}^{j=m}\left(y_T\left(t_j\right) - \hat{y}_E\left(t_j\right)\right)^2}$$

$$RMSE_{y^{(i)}(t)} := \sqrt{\frac{1}{m}\sum_{j=0}^{j=m}\left(y_T^{(i)}\left(t_j\right) - \hat{y}_E^{(i)}\left(t_j\right)\right)^2}$$

2. Maximum Absolute Deviation (MAD) [46]:

$$MAD_{y(t)} := \frac{1}{m}\sum_{j=0}^{j=m}\left|y_T\left(t_j\right) - \hat{y}_E\left(t_j\right)\right|$$

$$MAD_{y^{(i)}(t)} := \frac{1}{m}\sum_{j=0}^{j=m}\left|y_T^{(i)}\left(t_j\right) - \hat{y}_E^{(i)}\left(t_j\right)\right|$$

where $y_T(\cdot), y_T^{(i)}(\cdot)$ are the true values and $y_E(\cdot), y_E^{(i)}(\cdot)$ are the estimated values of the signal and its derivatives, respectively.

4.1 Estimation of LTI system states - examples

4.1.1 Third order system

Consider the following third order SISO LTI system:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & -4 & -2 \end{bmatrix} x \quad ; y = x_1 \quad ; \quad x(0) = [1, 1, 1] \tag{4.1}$$

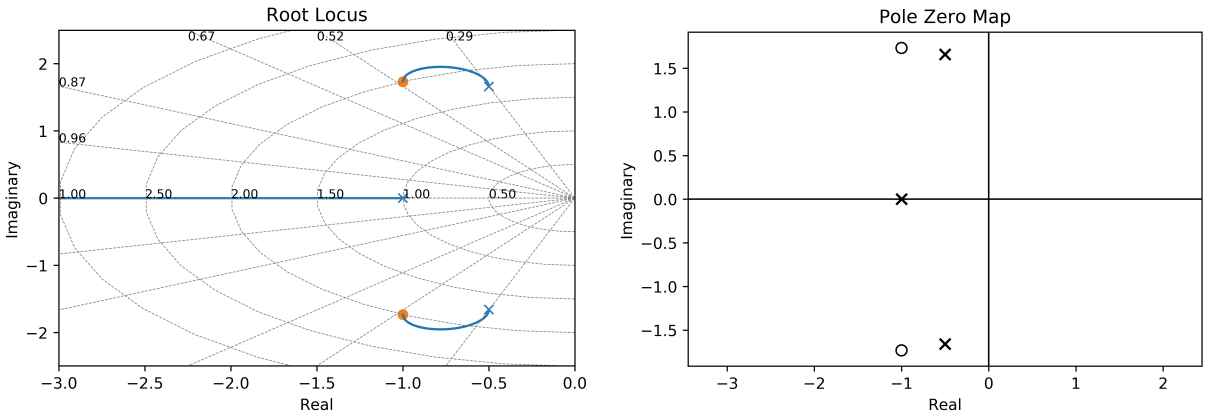
with its corresponding characteristic equation

$$y^{(3)}(t) + 2y^{(2)}(t) + 4y^{(1)}(t) + 3y(t) = 0 \tag{4.2}$$

As evident from figure 4.1, the system above is a stable third order system with poles at

$-0.5 \pm 1.658i$, -1. The true trajectories (without any noise) for all the states - $y, y^{(1)}$ and $y^{(2)}$ have been shown in figure 4.2.

In-order to compare and contrast the state estimation capabilities of *projection* method against Kalman (+RTS) filter using increasingly noisy signals, system dynamics, specifically system parameters are needed. The parameters given in table 4.1 were calculated using the kernel algorithm 3.6.



(a) Root locus plot

(b) Pole zero map

Fig. 4.1 System specifics for system 4.1

Std. Dev.	**SNR (db)**	a_0	a_1	a_2
True value		3	4	2
0	0	3	4	2
0.5	-1.69	3.00	4.00	2.00
1	-4.65	3.06	3.97	1.97
3	-12.55	3.03	4.05	1.95
5	-16.81	3.07	3.90	2.15
7	-19.74	2.93	3.45	1.93
10	-22.82	2.60	3.74	1.38
25	-30.32	3.06	3.50	1.38

Table 4.1 Estimated parameter values for 4.1 using kernels
Here, standard deviation and signal to noise ratio describe the noise levels in the signal and a_0, a_1, a_2 represents the three parameters describing the system dynamics

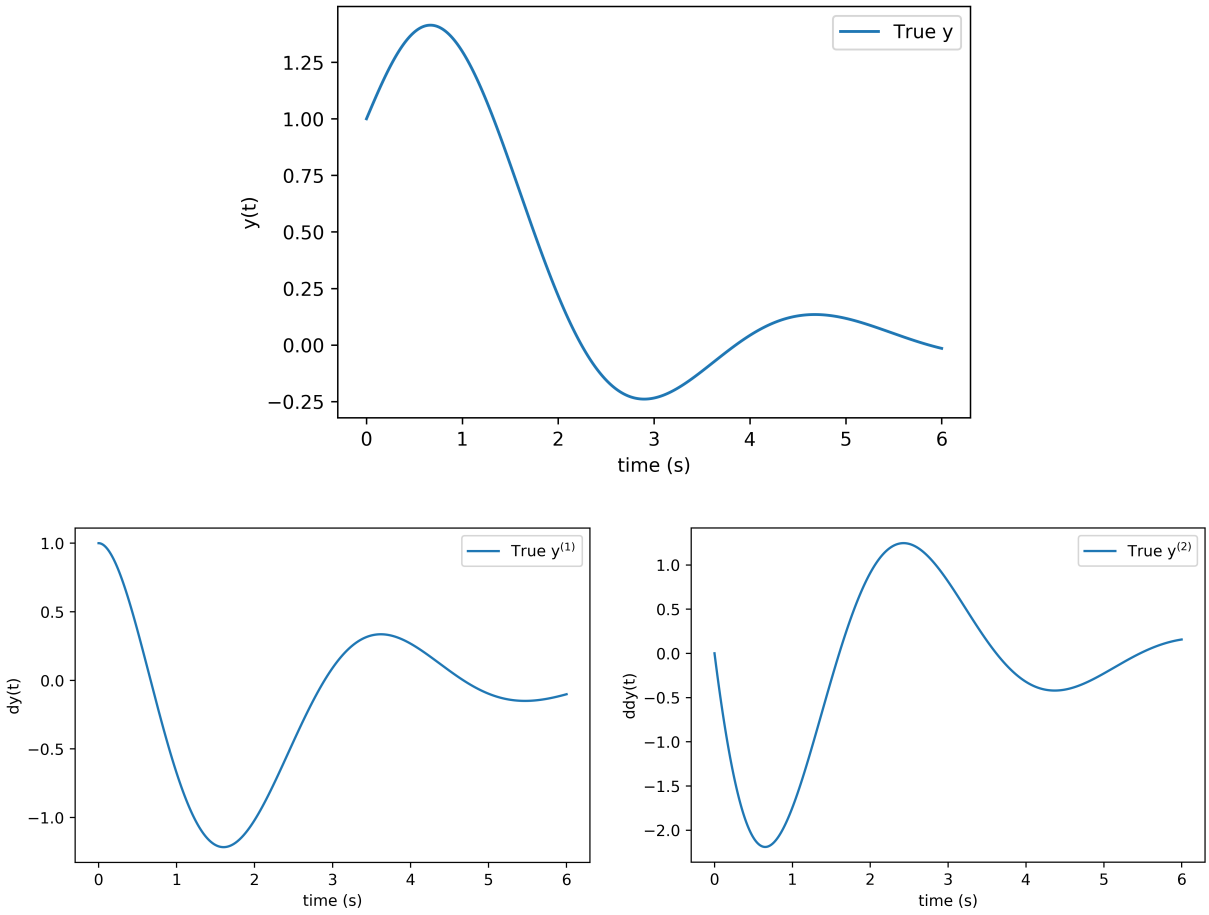


Fig. 4.2 True system state trajectories for 4.1

The parameters in table 4.1 were calculated between $[0, 6]$ seconds based on $60,000$ evenly spaced data-points in that interval. Each iteration involved the selection of 50 random points or *knots* and the iterations continued till a set tolerance - which was 0.01 in this case, was achieved between consecutive estimations. For the sake of brevity, the noiseless case from table 4.1 has been skipped to focus more on the cases where the signals have (increasing levels of) noise.

AWGN of $\sigma = 0.5$ (SNR of -1.69 dB)

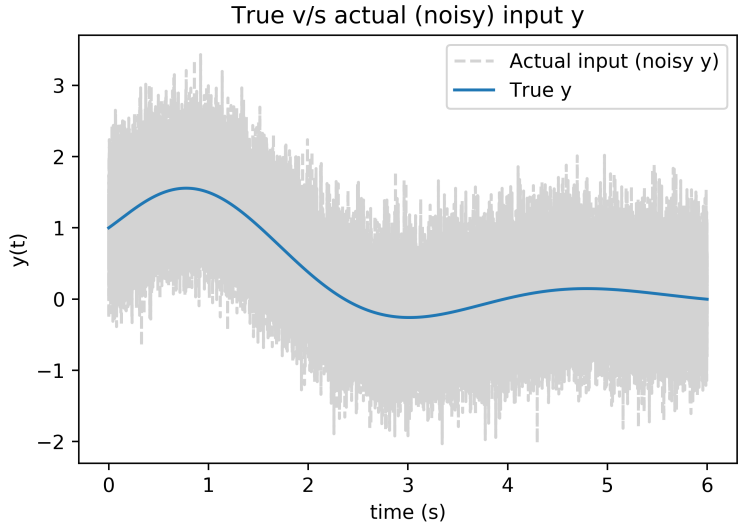


Fig. 4.3 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 10000, P = 0.0001, Q_var = 0.00001

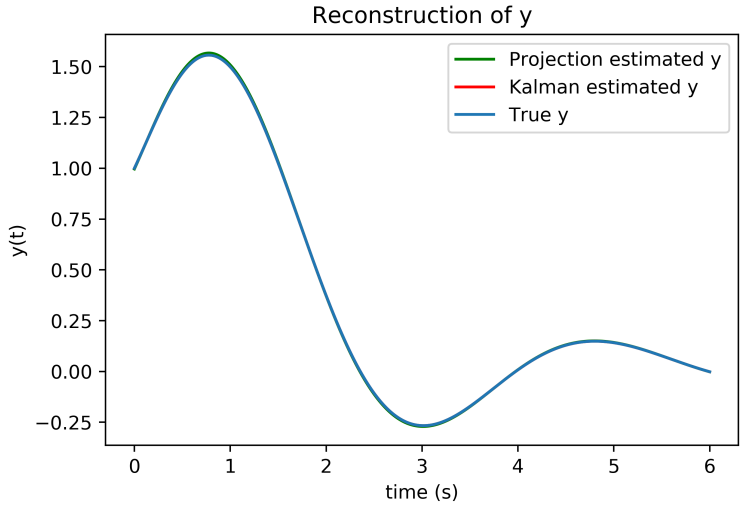


Fig. 4.4 Reconstruction of state y

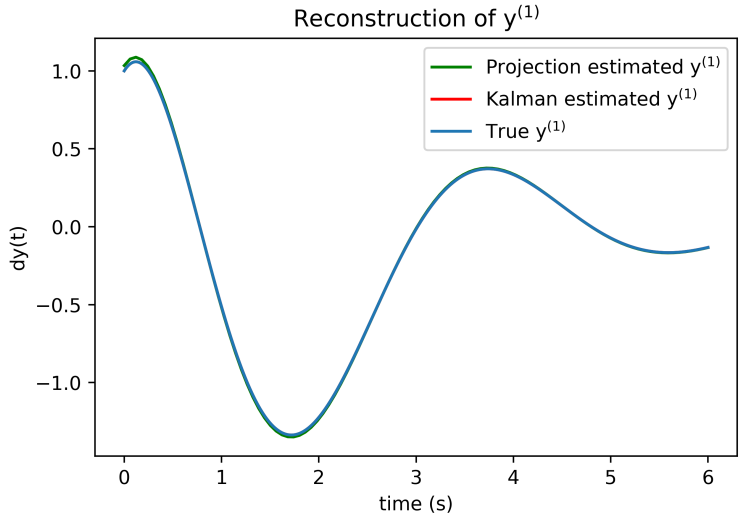


Fig. 4.5 Reconstruction of state $y^{(1)}$

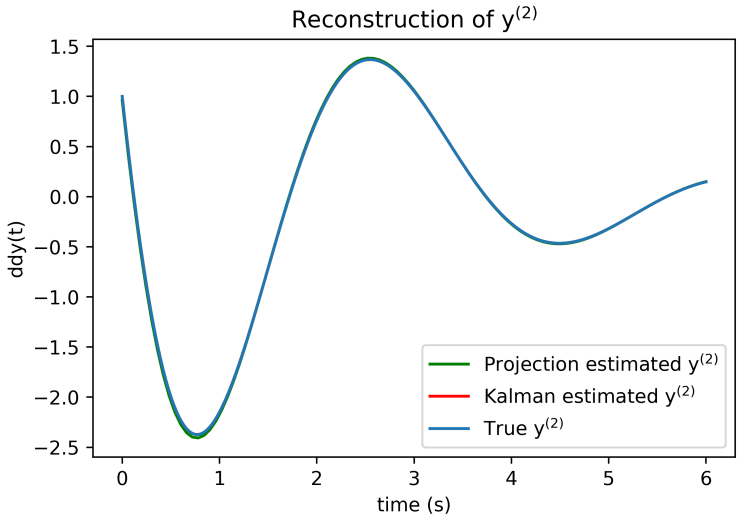


Fig. 4.6 Reconstruction of state $y^{(2)}$

AWGN of $\sigma = 1$ (SNR of -4.65 dB)

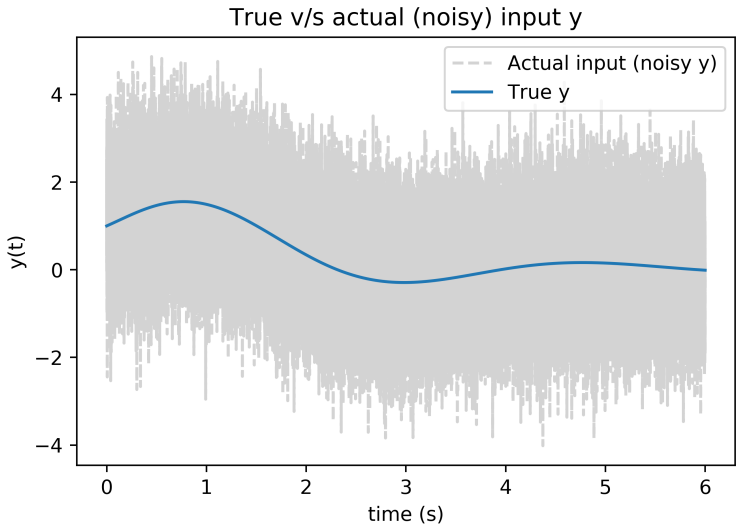


Fig. 4.7 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 10000, P = 0.0001, Q_var = 0.00005

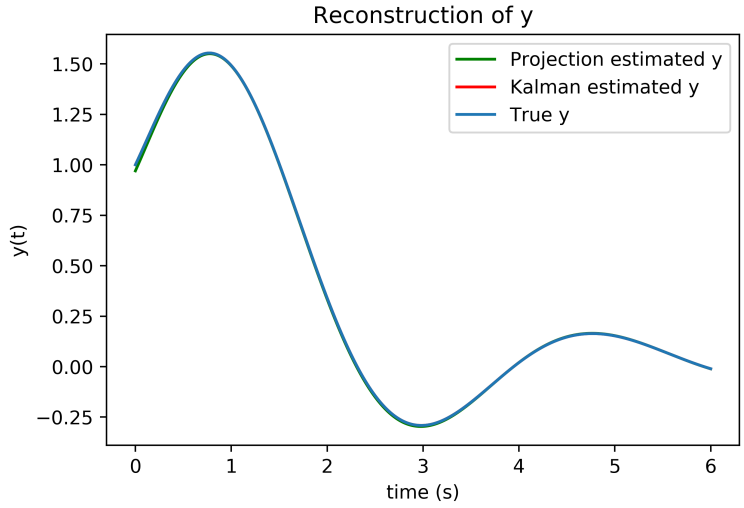


Fig. 4.8 Reconstruction of state y

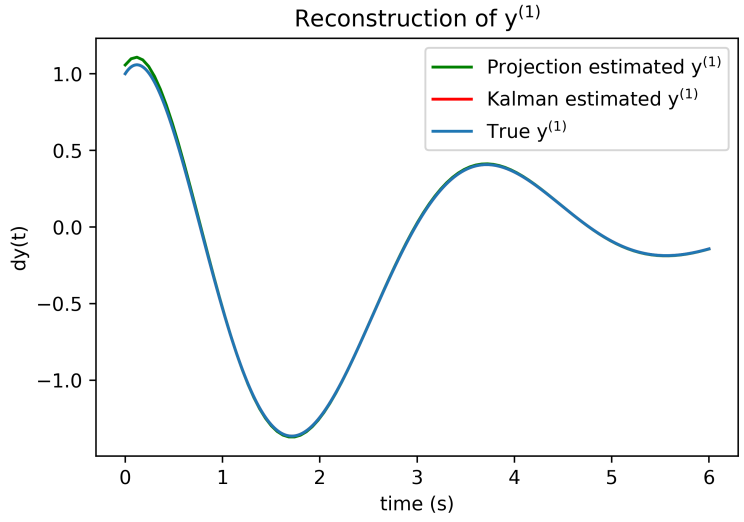


Fig. 4.9 Reconstruction of state $y^{(1)}$

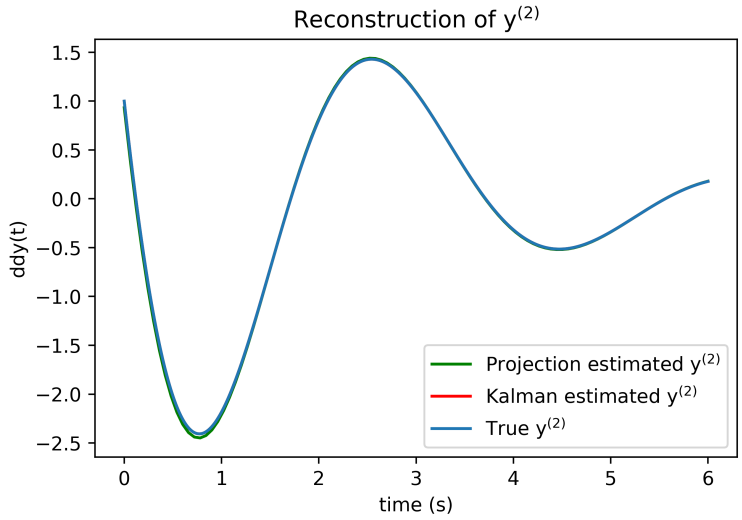


Fig. 4.10 Reconstruction of state $y^{(2)}$

AWGN of $\sigma = 3$ (SNR of -12.55 dB)

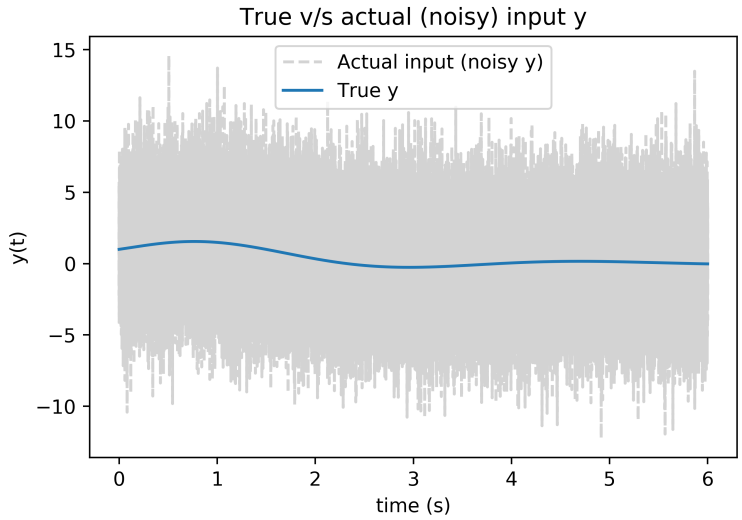


Fig. 4.11 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 10000, P = 0.0001, Q_var = 0.00001

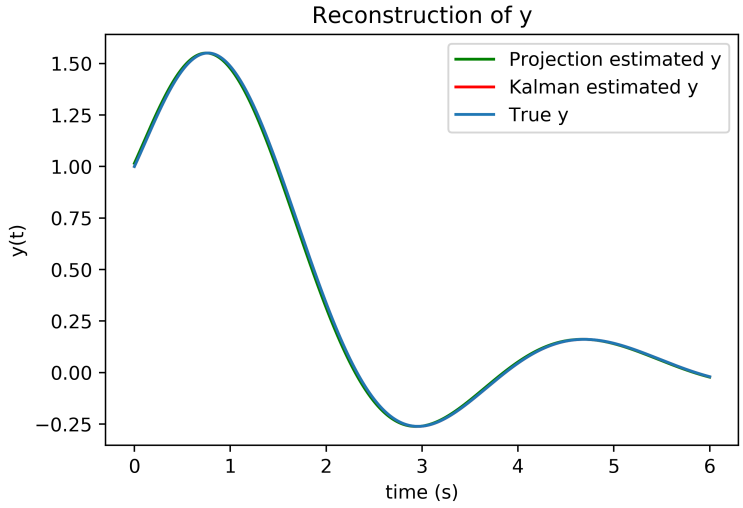


Fig. 4.12 Reconstruction of state y

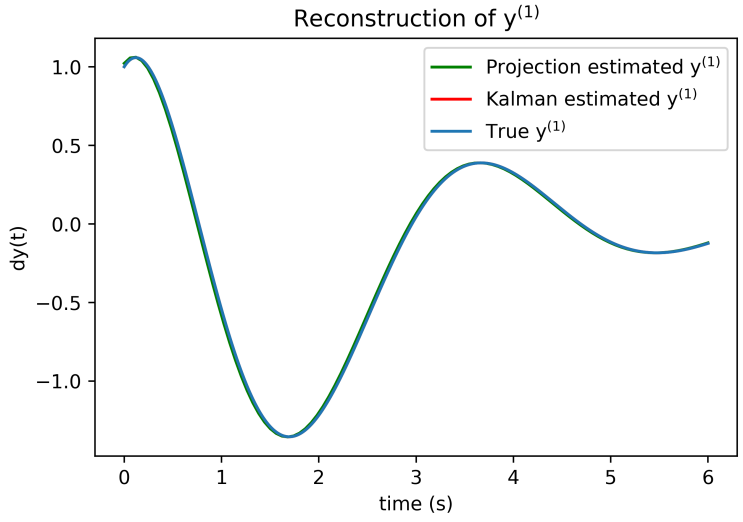


Fig. 4.13 Reconstruction of state $y^{(1)}$

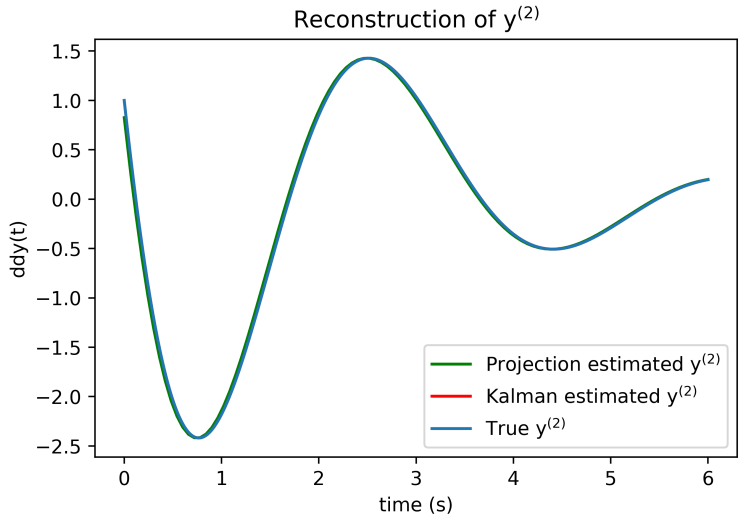


Fig. 4.14 Reconstruction of state $y^{(2)}$

AWGN of $\sigma = 5$ (SNR of -16.81 dB)

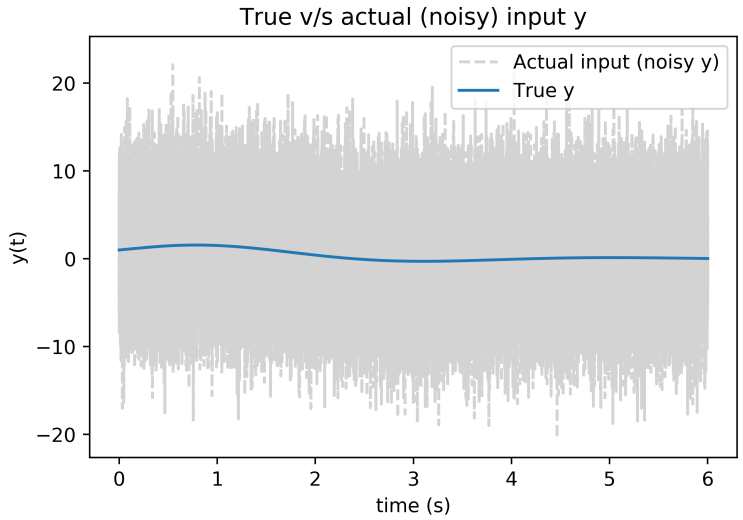


Fig. 4.15 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 10000, P = 0.0001, Q_var = 0.00001

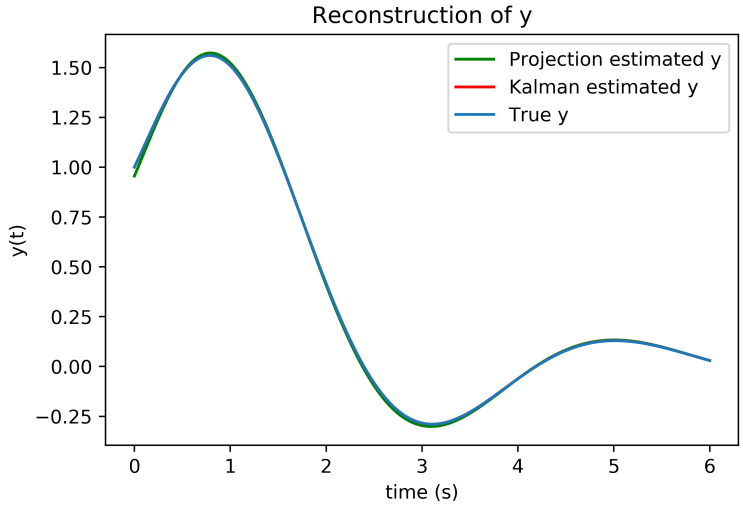


Fig. 4.16 Reconstruction of state y

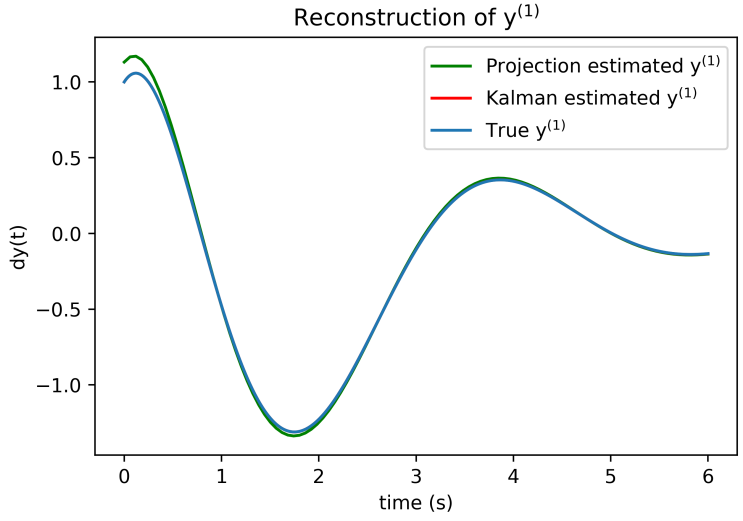


Fig. 4.17 Reconstruction of state $y^{(1)}$

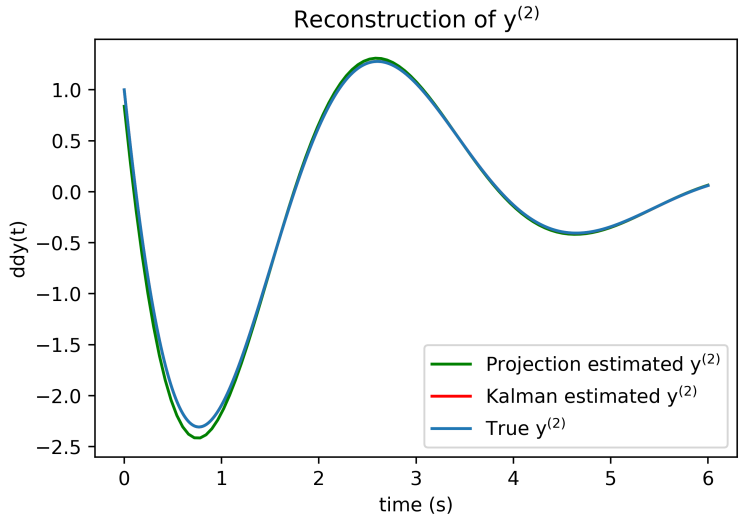


Fig. 4.18 Reconstruction of state $y^{(2)}$

AWGN of $\sigma = 7$ (SNR of -19.74 dB)

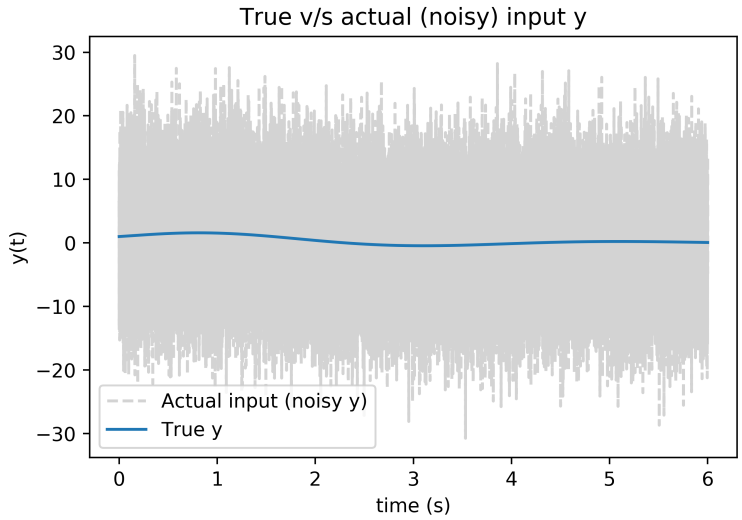


Fig. 4.19 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 10000, P = 0.0001, Q_var = 0.00001

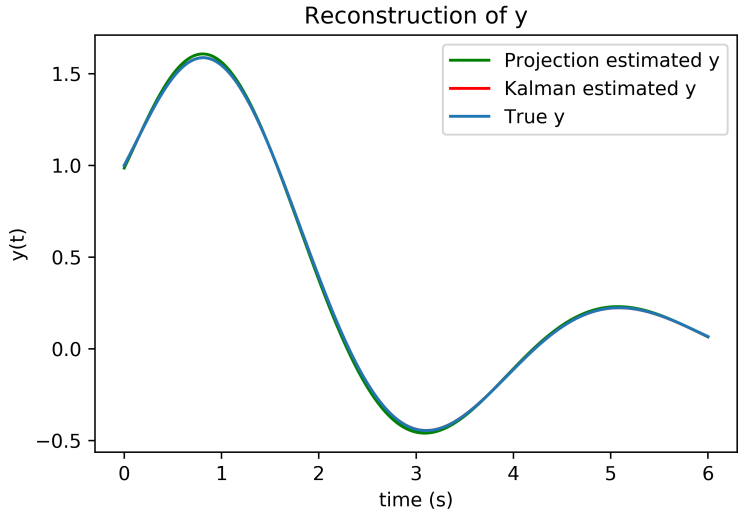


Fig. 4.20 Reconstruction of state y

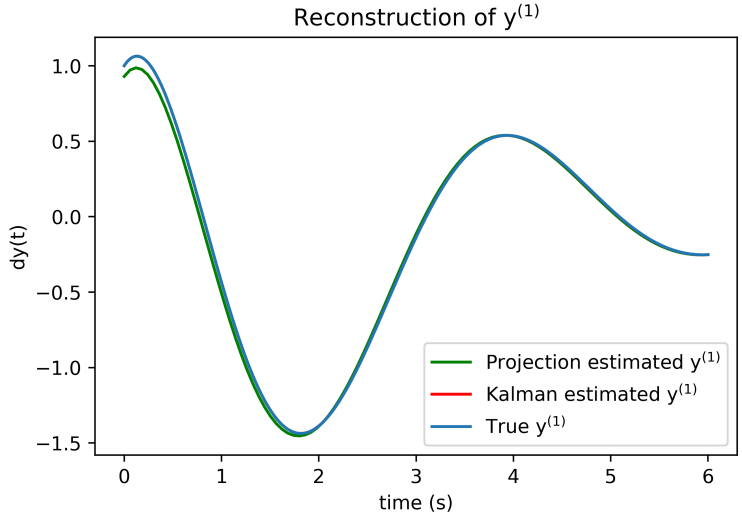


Fig. 4.21 Reconstruction of state $y^{(1)}$

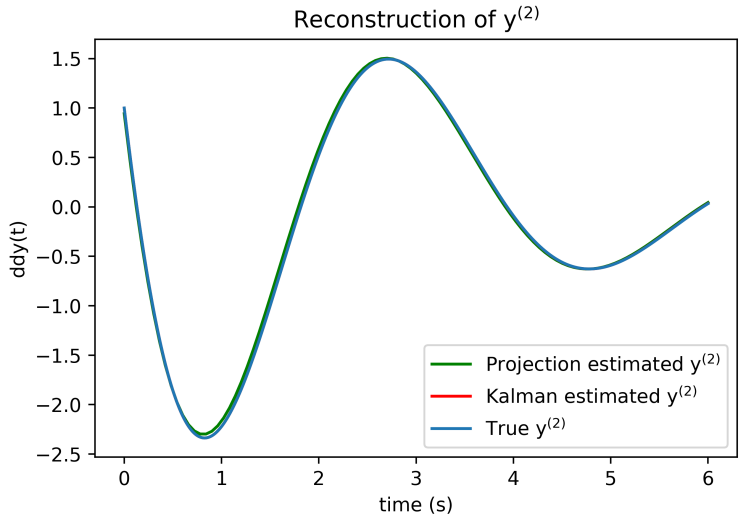


Fig. 4.22 Reconstruction of state $y^{(2)}$

AWGN of $\sigma = 25$ (SNR of -30.32 dB)

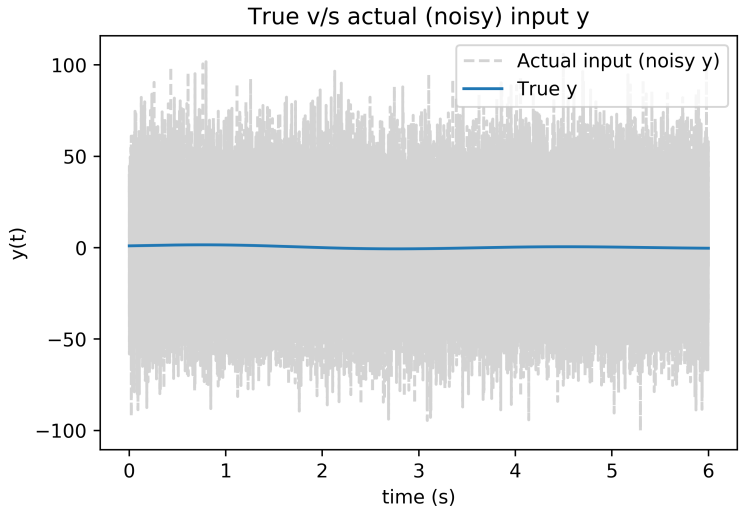


Fig. 4.23 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 10000, P = 0.0001, Q_var = 0.00001

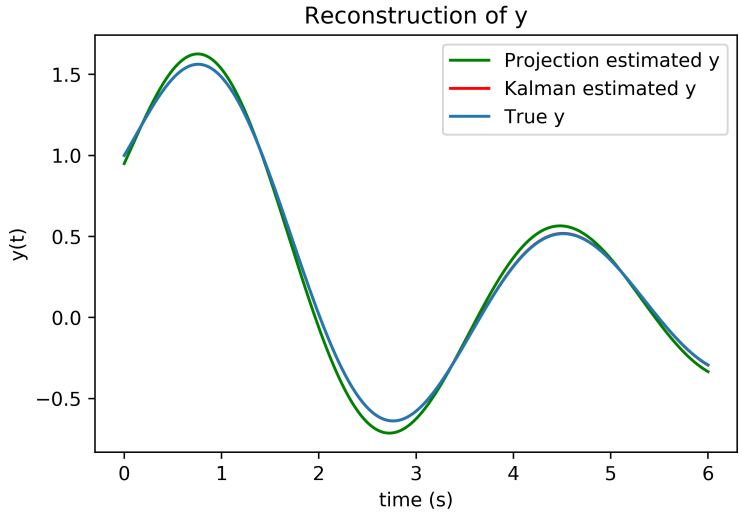


Fig. 4.24 Reconstruction of state y

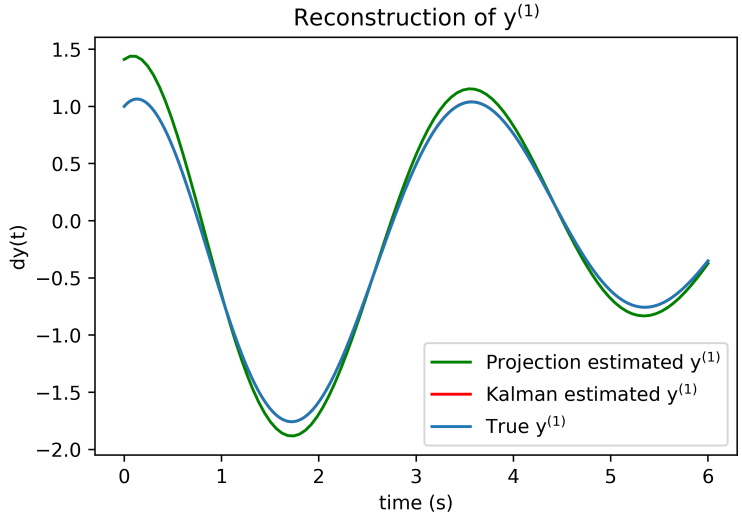


Fig. 4.25 Reconstruction of state $y^{(1)}$

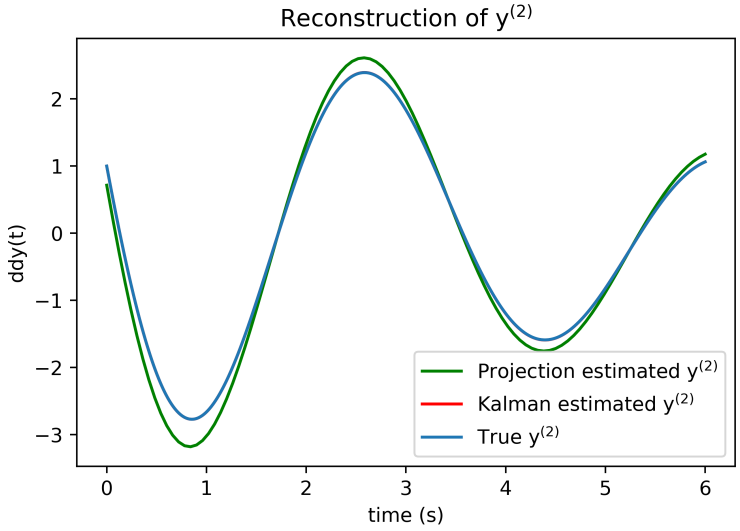


Fig. 4.26 Reconstruction of state $y^{(2)}$

Estimation Method	**Std. Dev.**	**SNR (db)**	**State**	**MAD**	**RMSE**
Projection	0.5	-1.69	y	0.0085	0.0036
			$y^{(1)}$	0.0128	0.0042
			$y^{(2)}$	0.0226	0.0089
Kalman	0.5	-1.69	y	0.0001	0.0000
			$y^{(1)}$	0.0002	0.0000
			$y^{(2)}$	0.0009	0.0002
Projection	1	-4.65	y	0.0114	0.0052
			$y^{(1)}$	0.0308	0.0114
			$y^{(2)}$	0.0419	0.0176
Kalman	1	-4.65	y	0.0004	0.0002
			$y^{(1)}$	0.0004	0.0002
			$y^{(2)}$	0.0009	0.0004
Projection	3	-12.62	y	0.0252	0.0118
			$y^{(1)}$	0.0488	0.0211
			$y^{(2)}$	0.1388	0.0388
Kalman	3	-12.62	y	0.0006	0.0003
			$y^{(1)}$	0.0006	0.0003
			$y^{(2)}$	0.0009	0.0005

Projection	5	-16.81	y	0.0175	0.0087
			$y^{(1)}$	0.0608	0.0157
			$y^{(2)}$	0.1553	0.0427
Kalman	5	-16.81	y	0.0005	0.0003
			$y^{(1)}$	0.0003	0.0002
			$y^{(2)}$	0.0009	0.0002
Projection	7	-19.74	y	0.0211	0.0119
			$y^{(1)}$	0.1052	0.0260
			$y^{(2)}$	0.1762	0.0592
Kalman	7	-19.74	y	0.0009	0.0005
			$y^{(1)}$	0.0011	0.0006
			$y^{(2)}$	0.0016	0.0009
Projection	10	-22.82	y	0.0519	0.0238
			$y^{(1)}$	0.0945	0.0382
			$y^{(2)}$	0.2610	0.0931
Kalman	10	-22.82	y	0.0013	0.0006
			$y^{(1)}$	0.0012	0.0006
			$y^{(2)}$	0.0020	0.0010
Projection	25	-30.32	y	0.0958	0.0505
			$y^{(1)}$	0.3125	0.1043
			$y^{(2)}$	0.4604	0.2156
Kalman	25	-30.32	y	0.0027	0.0013
			$y^{(1)}$	0.0034	0.0019
			$y^{(2)}$	0.0054	0.0031

Table 4.2: State estimation error metrics for 4.1
It is evident that the Kalman (+RTS) filter is a very strong state estimation method, given a precise and robust parameter estimate. The projection method performs quite well but is overshadowed by Kalman filter's exceptional filtering capability.

4.1.2 Fourth order system

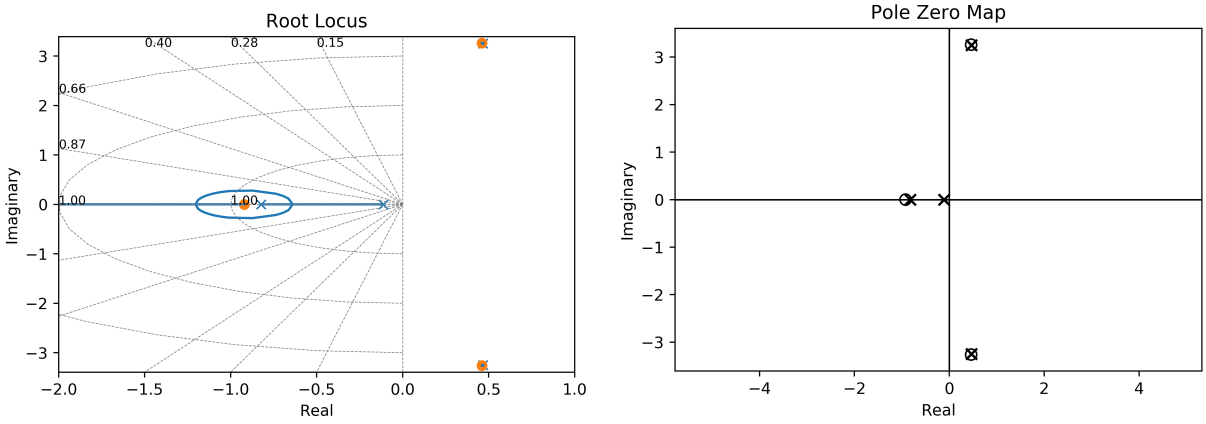
Consider the following fourth order SISO LTI system:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -10 & -10 & 0 \end{bmatrix} x \quad ; y = x_1 \quad ; \quad x(0) = [1, 1, 1, 1] \tag{4.3}$$

with its corresponding characteristic equation

$$y^{(4)}(t) + 0y^{(3)}(t) + 10y^{(2)}(t) + 10y^{(1)}(t) + 1y(t) = 0 \tag{4.4}$$

As evident from figure 4.27, the system above is an unstable fourth order system with poles at $0.47 \pm 3.2i,\ -0.11, -0.82$. The true trajectories (without any noise) for all the states - $y, y^{(1)}, y^{(2)}$ and $y^{(3)}$ has been shown in figure 4.28.



(a) Root locus plot

(b) Pole zero map

Fig. 4.27 System specifics for system 4.3

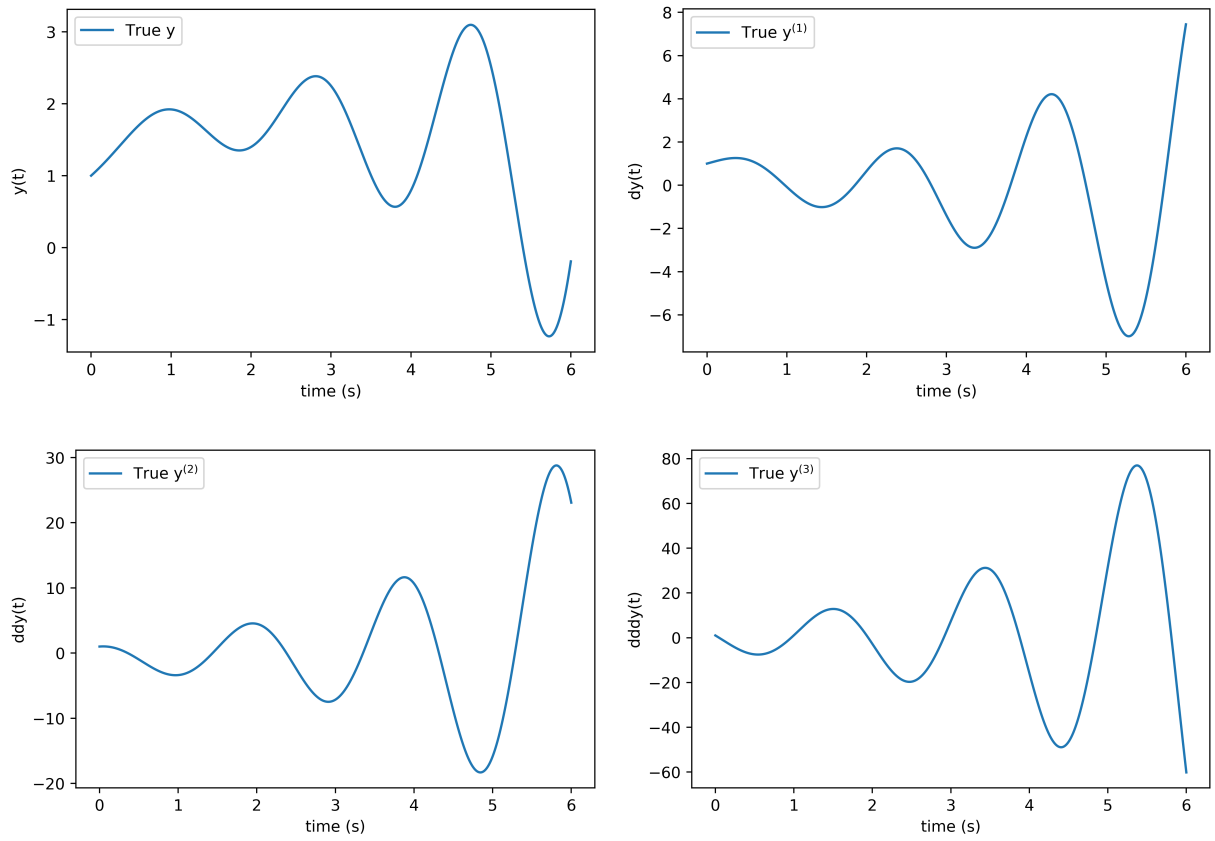


Fig. 4.28 True system state trajectories for 4.3

Std. Dev.	**SNR (db)**	a_0	a_1	a_2	a_3
True value		1	10	10	0
0	0	1.00	10.00	10.00	0.00
2	-1.23	1.05	9.88	9.89	-0.07
3	-4.78	0.94	9.22	9.91	-0.16
5	-9.18	1.00	8.48	9.91	-0.04
10	-15.23	1.01	8.92	10.02	0.16
20	-21.22	1.27	11.02	9.29	0.33
30	-24.73	2.84	11.60	11.46	1.70

Table 4.3 Estimated parameter values for 4.3 using kernels
Standard deviation and signal to noise ratio describe the noise levels in the signal and a_0, a_1, a_2, a_3 represents the four parameters describing the system dynamics

The parameters in table 4.3 were calculated between $[0, 6]$ seconds based on $60,000$ evenly spaced data-points in that interval (same as the previous example). Each iteration involved the selection of 50 *knots* and the iterations continued till a set tolerance of 0.01 was achieved between consecutive estimations.

AWGN of $\sigma = 2$ (SNR of -1.23 dB)

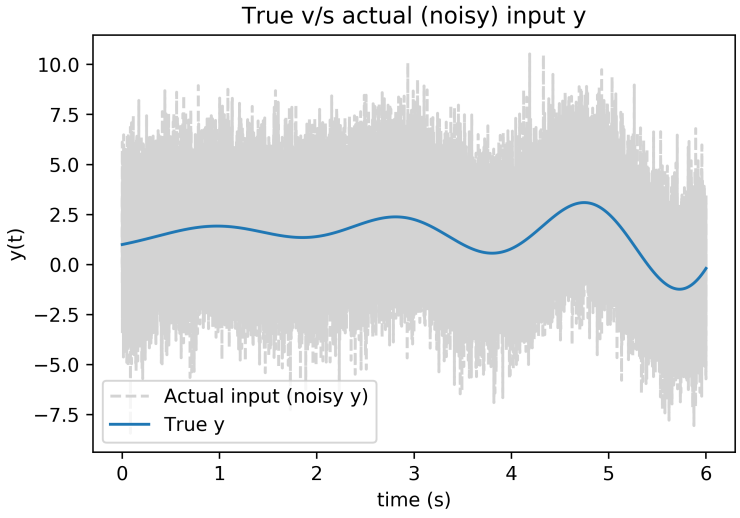


Fig. 4.29 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 20, P = 0.00001, Q_var = 0.001

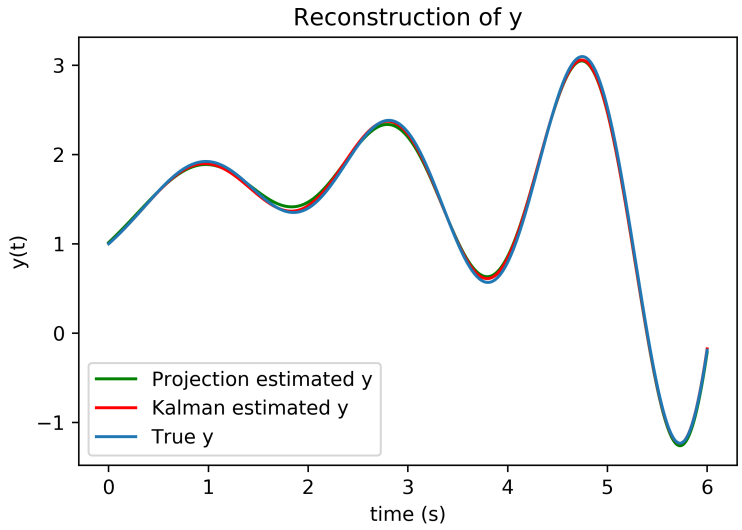


Fig. 4.30 Reconstruction of state y

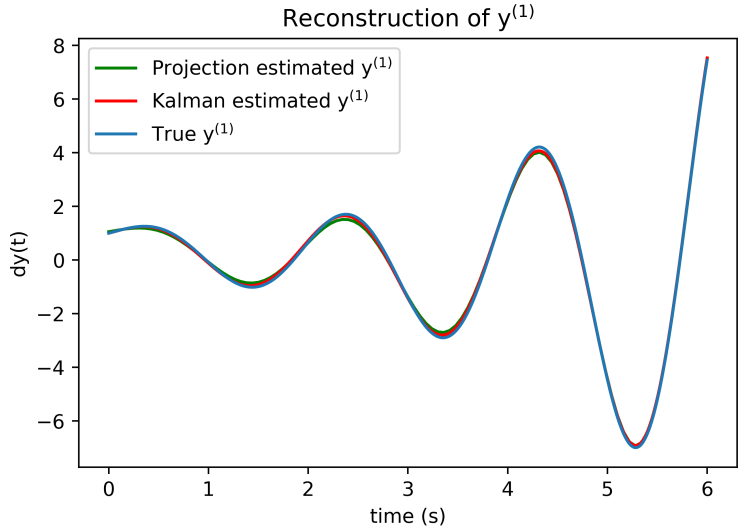


Fig. 4.31 Reconstruction of state $y^{(1)}$

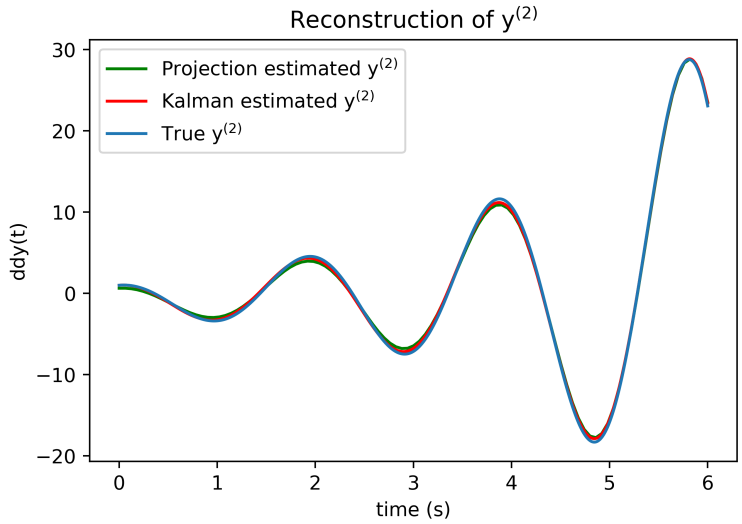


Fig. 4.32 Reconstruction of state $y^{(2)}$

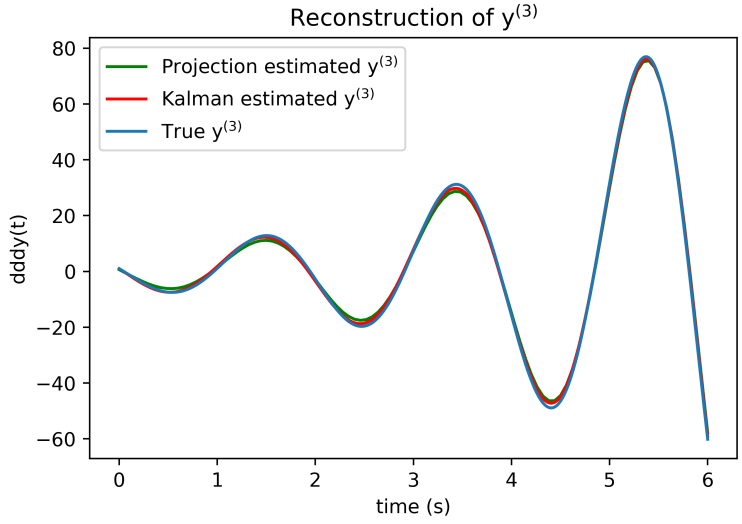


Fig. 4.33 Reconstruction of state $y^{(3)}$

AWGN of $\sigma = 3$ (SNR of -4.78 dB)

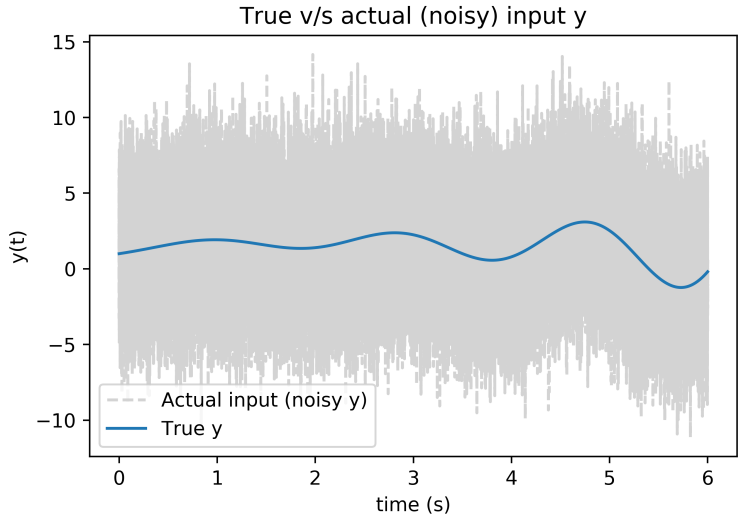


Fig. 4.34 Actual (noisy) input to the algorithm versus true signal

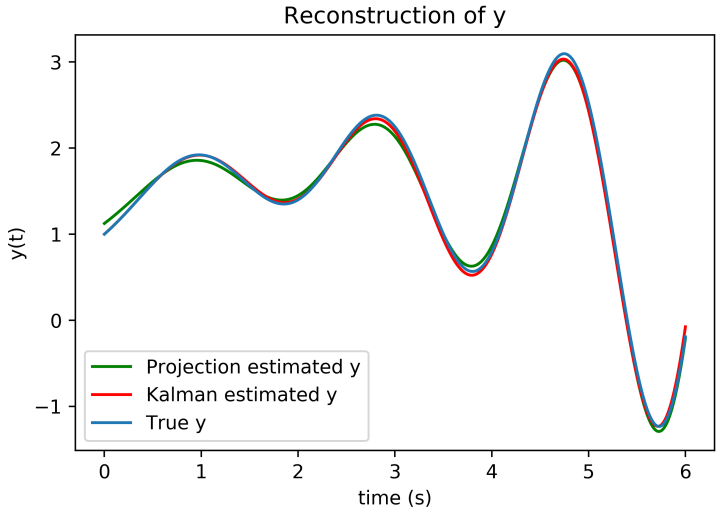


Fig. 4.35 Reconstruction of state y

Parameters for Kalman (+RTS) filter: $R = 5$, $P = 0.00001$, $Q\_var = 0.001$

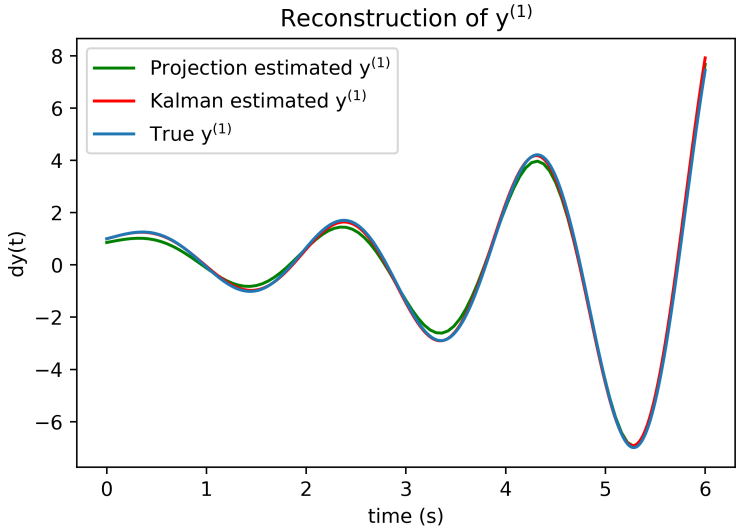


Fig. 4.36 Reconstruction of state $y^{(1)}$

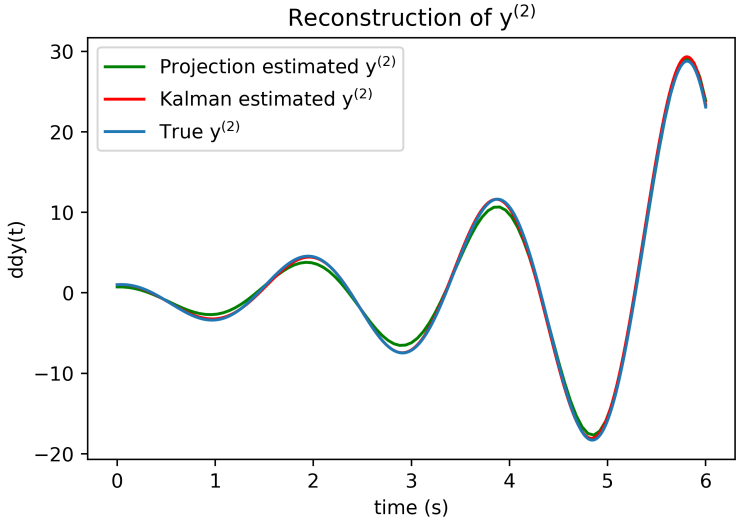


Fig. 4.37 Reconstruction of state $y^{(2)}$

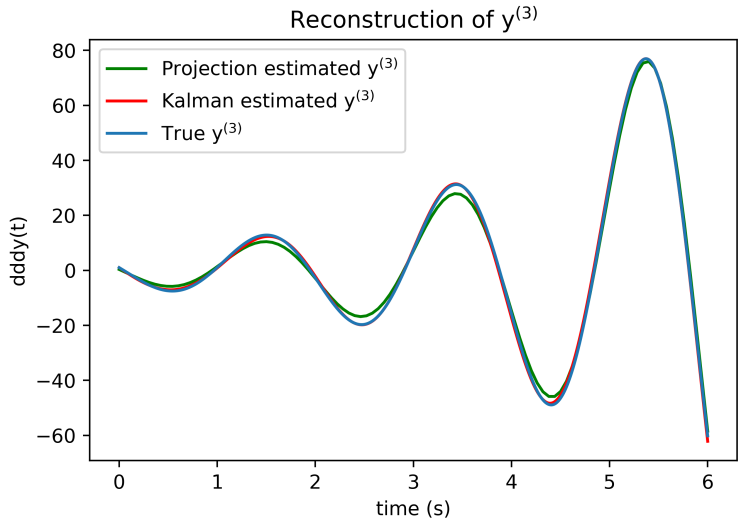


Fig. 4.38 Reconstruction of state $y^{(3)}$

AWGN of $\sigma = 5$ (SNR of -9.18 dB)

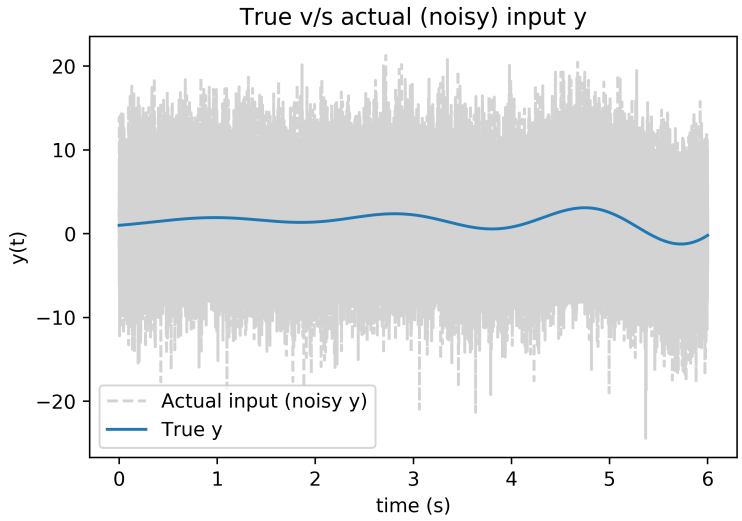


Fig. 4.39 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 5, P = 0.00001, Q_var = 0.001

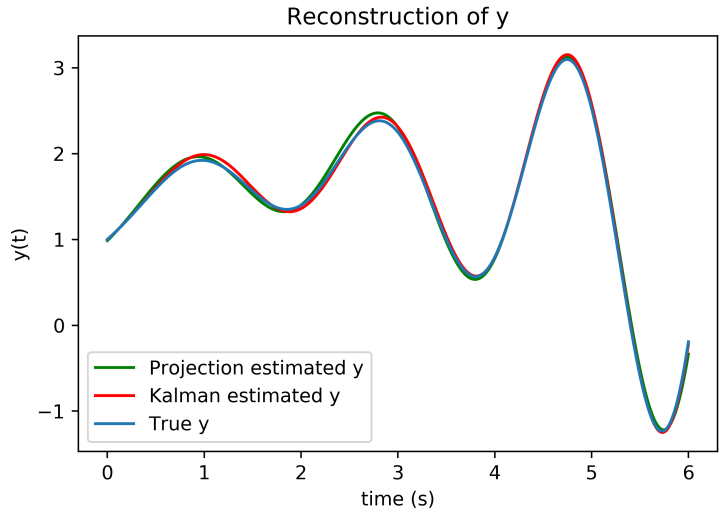


Fig. 4.40 Reconstruction of state y

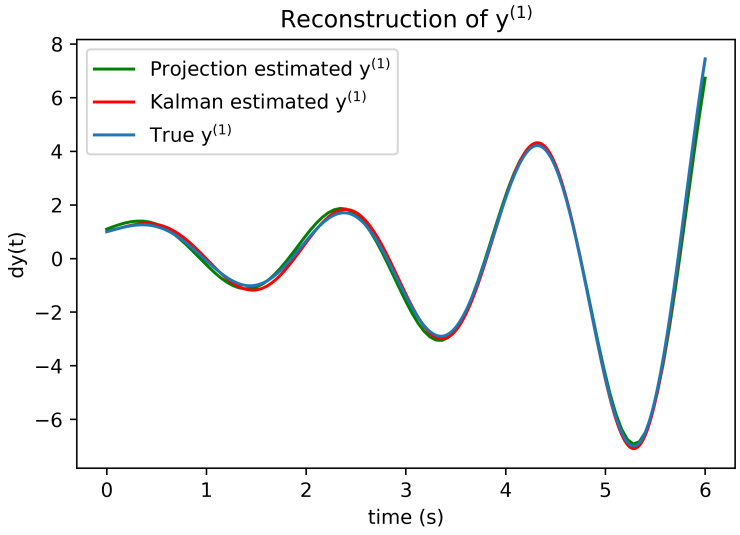


Fig. 4.41 Reconstruction of state $y^{(1)}$

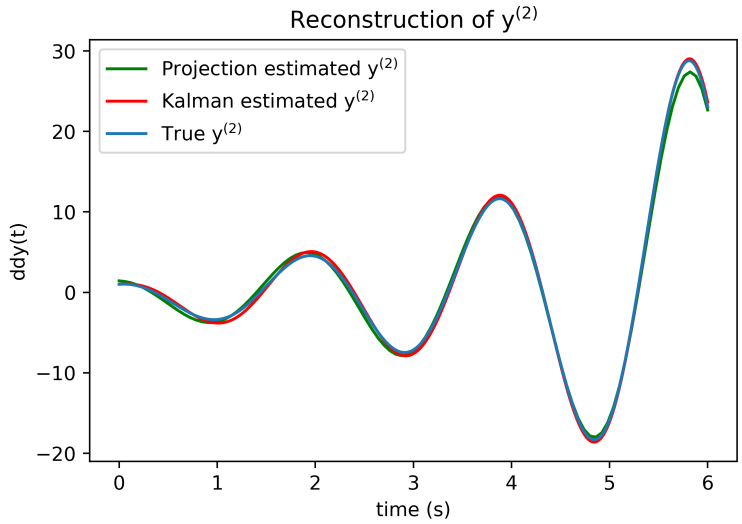


Fig. 4.42 Reconstruction of state $y^{(2)}$

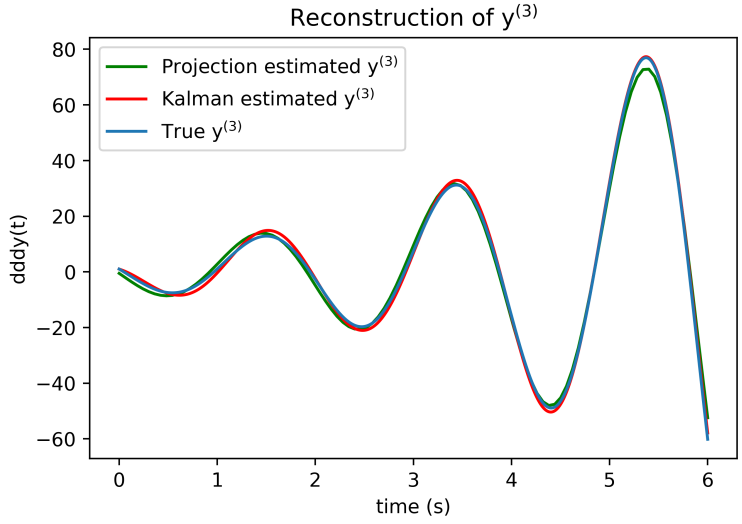


Fig. 4.43 Reconstruction of state $y^{(3)}$

AWGN of $\sigma = 10$ (SNR of -15.23 dB)

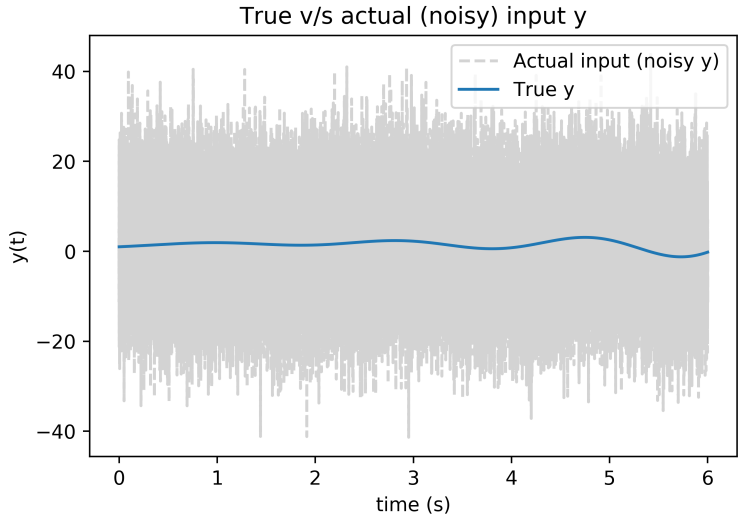


Fig. 4.44 Actual (noisy) input to the algorithm versus true signal

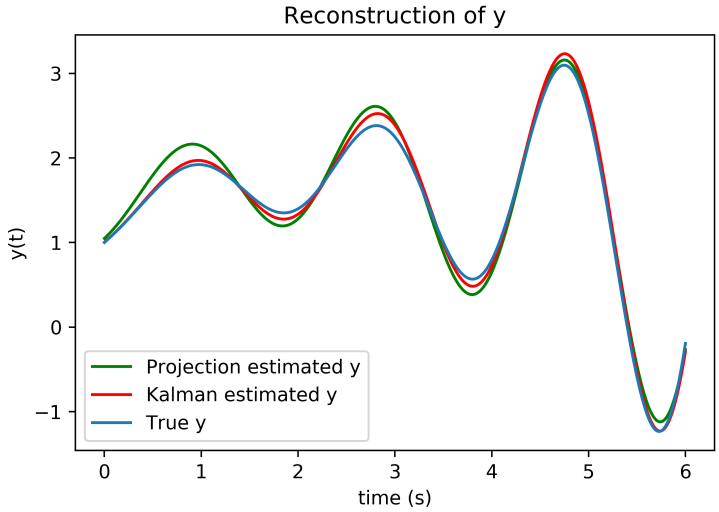


Fig. 4.45 Reconstruction of state y

Parameters for Kalman (+RTS) filter: R = 20, P = 0.00001, Q_var = 0.001

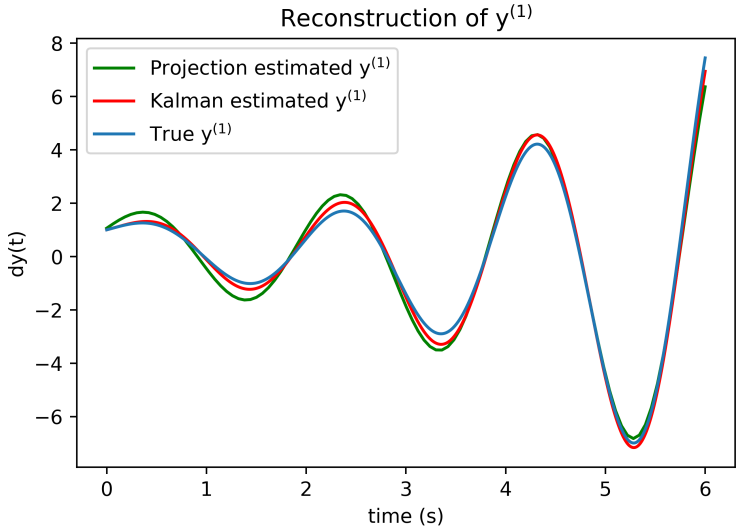


Fig. 4.46 Reconstruction of state $y^{(1)}$

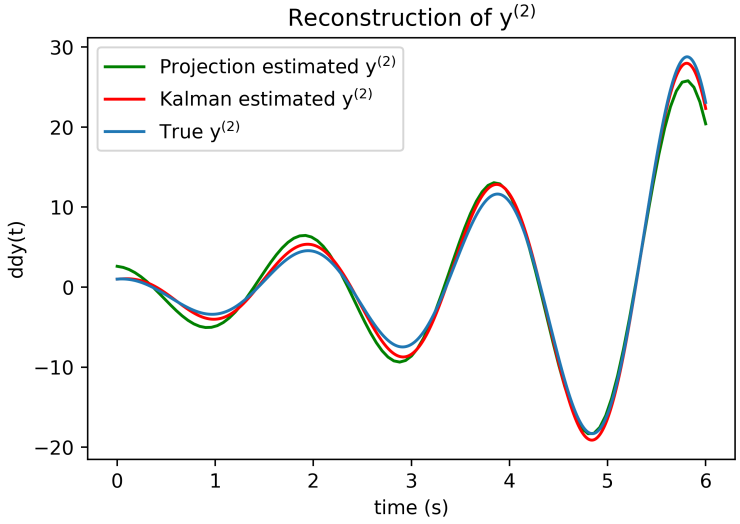


Fig. 4.47 Reconstruction of state $y^{(2)}$

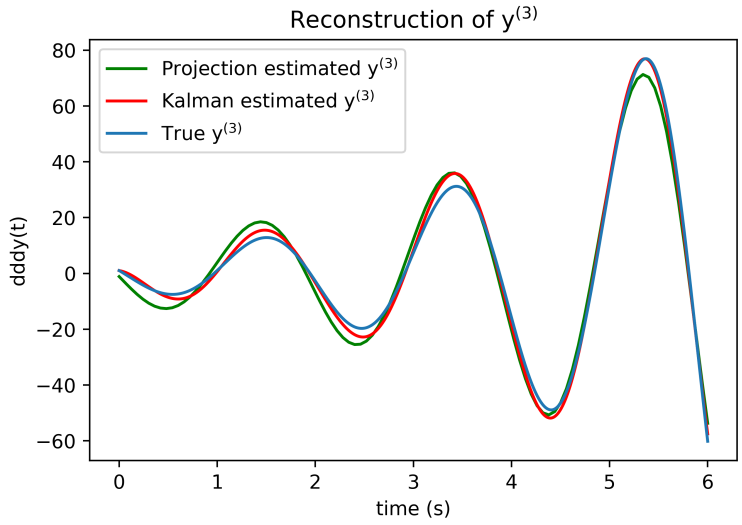


Fig. 4.48 Reconstruction of state $y^{(3)}$

AWGN of $\sigma = 30$ (SNR of -24.73 dB)

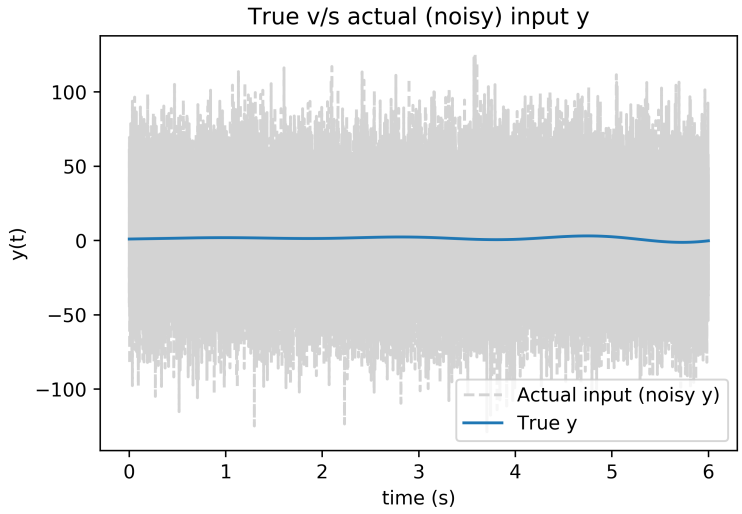


Fig. 4.49 Actual (noisy) input to the algorithm versus true signal

Parameters for Kalman (+RTS) filter: R = 0.01, P = 0.001, Q_var = 0.0001

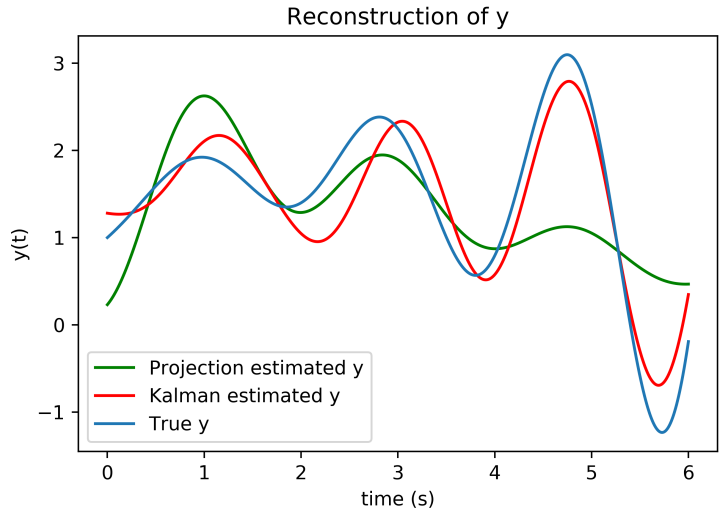


Fig. 4.50 Reconstruction of state y

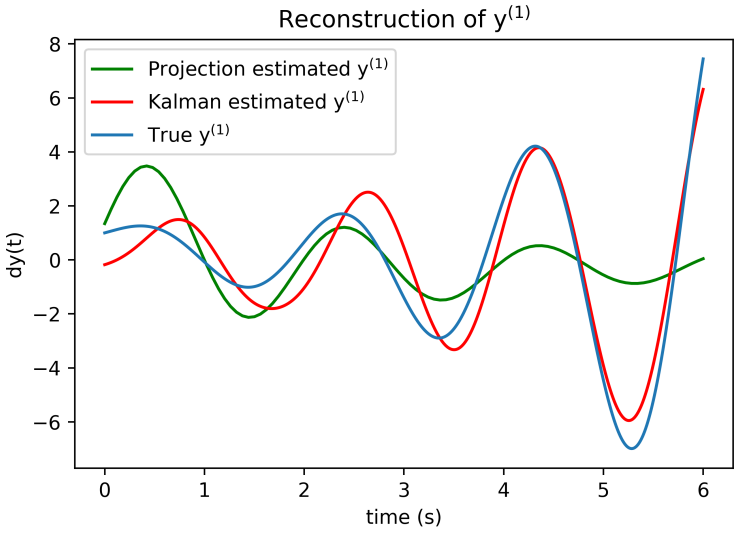


Fig. 4.51 Reconstruction of state $y^{(1)}$

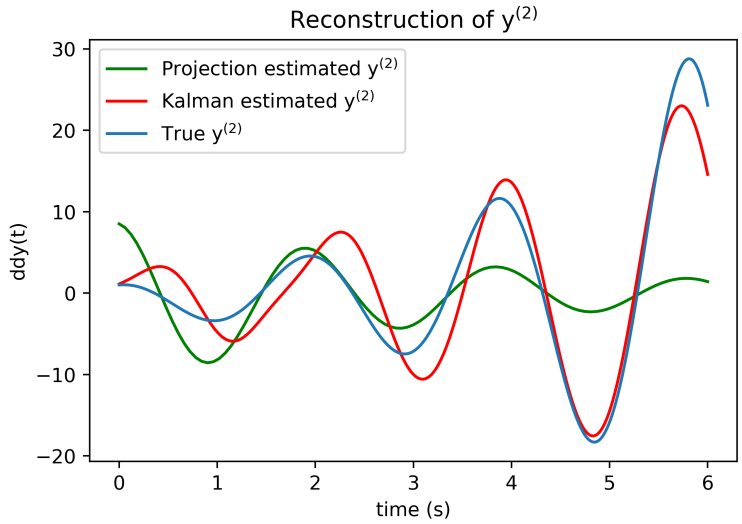


Fig. 4.52 Reconstruction of state $y^{(2)}$

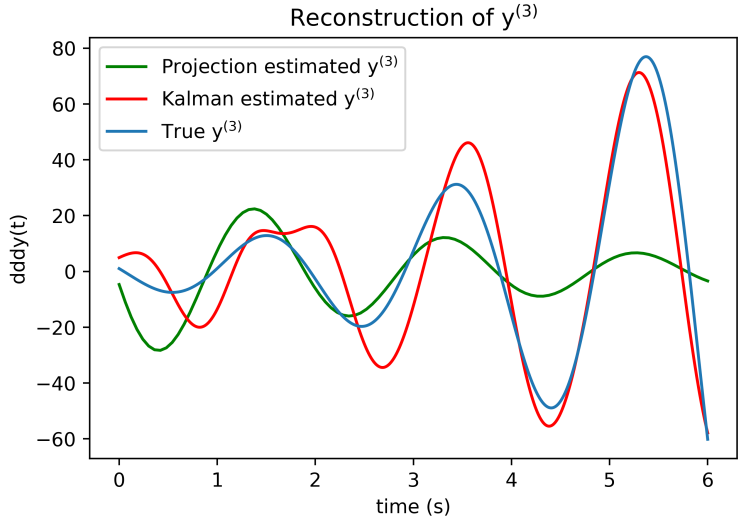


Fig. 4.53 Reconstruction of state $y^{(3)}$

Estimation Method	**Std. Dev.**	**SNR (db)**	**State**	**MAD**	**RMSE**
Projection	2	-1.23	y	0.0682	0.0389
			$y^{(1)}$	0.2175	0.1184
			$y^{(2)}$	0.7625	0.4163
			$y^{(3)}$	2.6432	1.5559
Kalman	2	-1.23	y	0.0488	0.0249
			$y^{(1)}$	0.1560	0.0747
			$y^{(2)}$	0.4939	0.2477
			$y^{(3)}$	1.7026	0.9218
Projection	3	-4.78	y	0.1246	0.0630
			$y^{(1)}$	0.3071	0.1750
			$y^{(2)}$	0.9871	0.5398
			$y^{(3)}$	3.3821	1.9014
Kalman	3	-4.78	y	0.1163	0.0443
			$y^{(1)}$	0.4758	0.1218
			$y^{(2)}$	0.5972	0.2777
			$y^{(3)}$	1.9797	0.7043
Projection	5	-9.18	y	0.1438	0.0510
			$y^{(1)}$	0.7265	0.1827
			$y^{(2)}$	1.6710	0.5710
			$y^{(3)}$	7.7420	1.9328
Kalman	5	-9.18	y	0.0730	0.0388
			$y^{(1)}$	0.1903	0.0981
			$y^{(2)}$	0.5920	0.3201
			$y^{(3)}$	2.2800	1.1539
Projection	10	-15.23	y	0.2635	0.1371
			$y^{(1)}$	1.0827	0.3883
			$y^{(2)}$	3.1557	1.3487
			$y^{(3)}$	7.8287	4.2939
Kalman	10	-15.23	y	0.1464	0.0750
			$y^{(1)}$	0.5020	0.2164
			$y^{(2)}$	1.2612	0.6482
			$y^{(3)}$	4.7992	2.1612

Projection	20	-21.22	y	0.3818	0.1886
			$y^{(1)}$	1.0141	0.5878
			$y^{(2)}$	3.2641	1.9886
			$y^{(3)}$	10.838	6.6894
Kalman	20	-21.22	y	0.3475	0.1662
			$y^{(1)}$	1.4730	0.4489
			$y^{(2)}$	5.2239	1.8660
			$y^{(3)}$	15.882	6.1724
Projection	30	-24.73	y	1.9713	0.8298
			$y^{(1)}$	7.399	2.4007
			$y^{(2)}$	26.977	9.0293
			$y^{(3)}$	70.700	25.319
Kalman	30	-24.73	y	0.7911	0.3562
			$y^{(1)}$	2.0552	1.0112
			$y^{(2)}$	8.512	3.4804
			$y^{(3)}$	23.223	12.068

Table 4.4: State estimation error metrics for 4.3
This system was complex for all estimation algorithms - even the robust kernel (parameter) predictor gave way after $\approx$ -22dBs. Again, it is clear that given a decent set of system parameters, the Kalman (+RTS) filter almost always performs better than the projection method.

Regardless of the system's order or stability, Kalman filter combined with an RTS smoother clearly outperforms the projection method in terms of state estimation (given a good estimate of the system parameters). The examples above go in the direction of developing a baseline estimation framework for any state and parameter estimation algorithm. These examples also demonstrate the efficiency and resilience of the forward-backward kernel approach as a parameter estimator; more on this will be analyzed in the next chapter.

Chapter 5

Unscented Kalman Filter Method for Joint Parameter and State Estimation

For Kalman filters to accurately forecast states at any point in time, as stated in Chapter 2 and demonstrated in Chapter 4, they must have complete knowledge of the underlying system dynamics in order to conduct the estimation optimally. This chapter examines the Unscented Kalman Filter (UKF), which is one of the most popular and widely used methods for nonlinear system estimation. Later in the chapter, the chapter examines a method that uses UKF to forecast the parameters and state of the system concurrently (without any prior knowledge) and compares the findings to the Kernel method to demonstrate the latter's resilience and efficiency.

5.1 Unscented Kalman filter

The Kalman filter is an optimal, minimum mean square error estimator for linear systems. When system dynamics are intrinsically non-linear, the Extended Kalman filter (EKF) has been used. EKF performs a truncated first-order Taylor linearization on the system dynamics equations about the current state and then linear filtering equations are applied. Despite the fact that EKF has been widely employed for a variety of purposes, it has divergence issues since the linearization process rarely captures the correct dynamics of the underlying system. To solve this issue, this derivative-less method circumvents the issue by deterministically sampling the joint density of the states in such a way that the mean and covariances are preserved. The full non linear system dynamics are then applied to these sample points in

order to propagate the density through the prediction part of the filter - a process known as Unscented Transformation [47].

The Unscented Kalman Filter (UKF) [48] is part of a larger class of filters known as Sigma-Point Kalman Filters or Linear Regression Kalman Filters, which linearize a nonlinear function using a statistical linearization technique. The output of UKF is then fed into an RTS smoother (as explained in Chapter 2), which produces a smoothed trajectory of the output and its derivatives [49, 50].

5.1.1 UKF Algorithm [24, 48, 51]

Unscented Transform

1. **Selection of sigma points**

   Propagate the state vector, $\mathrm{x_k}$ (dimension of state space n) through the nonlinear process model $f()$ which has a mean and covariance of $\bar{x}_k$ and P_k respectively.

   Let M_k be a matrix of $2n+1$ sigma vectors $\mathrm{m_{i,k}}$ (with corresponding weights $w^m_{i,k}$ (mean) and $w^c_{i,k}$ (covariance)).

$$w^m = \begin{bmatrix} w^m_0 & w^m_1 & \dots & w^m_{2n} \end{bmatrix} \tag{5.1}$$

$$w^c = \begin{bmatrix} w^c_0 & w^c_1 & \dots & w^c_{2n} \end{bmatrix} \tag{5.2}$$

$$M = \begin{bmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \dots & m_{1,n-1} \\ & & \vdots & \\ m_{2n,0} & m_{2n,1} & \dots & m_{2n,n-1} \end{bmatrix} \tag{5.3}$$

2. **Sigma point computation**

   Below is a walk-through of how sigma points are computed:

The mean of the input is the first sigma point (m_0).

$$m_0 = \bar{x}_k \tag{5.4}$$

For convenience, define $\lambda = \alpha^2(n + \kappa) - n$ where λ is a scaling factor, α determines the spread of the sigma points and κ is a secondary scaling parameter. The remaining sigma points can be computed as:

$$m_i = \begin{cases} \bar{x}_k + [\sqrt{(n+\lambda)P_k}]_j, & \text{for } j = 1, ..., n \\ \bar{x}_k - [\sqrt{(n+\lambda)P_k}]_{j-n}, & \text{for } j = n+1, ..., 2n \end{cases} \tag{5.5}$$

The j subscript selects the j^{th} row or column of the matrix. The covariance matrix is scaled by a constant, square rooted and symmetry is ensured by adding and subtracting it from the mean.

3. **Square root of matrix**

   In order to compute a new set of sigma points, the square root matrix of the posterior covariance matrix is required ($P_k = S_k S_k^T$). This definition is favored because S_k is computed using *Cholesky decomposition*. It decomposes a Hermitian, positive definite matrix into a triangular matrix and its conjugate transpose.

4. **Weight computation**

   The formulation uses one set of weights for the means and another set for the covariances. The weight for the mean and covariance of m_0 is

$$w_0^m = \frac{\lambda}{n+\lambda} \tag{5.6}$$

$$w_0^c = \frac{\lambda}{n+\lambda} + 1 - \alpha^2 + \beta \tag{5.7}$$

   where β is used to incorporate prior knowledge of distribution and is set to 2 for Gaussian distribution. The weights for the rest of the sigma points m_i are the same for the mean and covariance:

$$w_i^m = w_i^c = \frac{1}{2(n+\lambda)}, \text{ for } i = 1, ..., 2n \tag{5.8}$$

Now, consider the following non-linear system, described by the difference equation and the observation model with additive noise:

$$x_{k+1} = f(x_k, u_k) + w_k \tag{5.9}$$

$$y_k = h(x_k) + v_k \tag{5.10}$$

The initial state x_0 is a random vector with known mean $\bar{x}_0 = E[x_0]$ and covariance $P_0 = E[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T]$.

In case of non-additive process and measurement noise, the unscented transformation scheme is applied to the augmented state:

$$x_k^{aug} = \begin{bmatrix} x_k^T & w_{k-1}^T & v_k^T \end{bmatrix}^T$$

$$P_k^{aug} = \begin{bmatrix} P_k & 0 & 0 \\ 0 & P_w & 0 \\ 0 & 0 & P_v \end{bmatrix}$$

Which gives $x_0^{aug} = [\bar{x}_0^T \quad 0 \quad 0]^T$

Predict Step

This step computes the prior using the process model $f()$, which is assumed to be nonlinear. Sigma points M_{k-1} and their corresponding weights w^m, w^c are generated and each sigma point is passed through $f(x, \Delta t)$. This projects the sigma points forward in time according to the process model, forming the new prior, which is a set of sigma points.

For k $\in$ [1,2,...,∞), the sigma points would be:

$$M_{k-1} = [\bar{x}_{k-1} \quad \bar{x}_{k-1} \pm \sqrt{(n+\lambda)P_{k-1}}] \tag{5.11}$$

$$M_k = f(M_{k-1}) \tag{5.12}$$

The transformed points are used to compute the mean and covariance of the prior/forecast

value.

$$\bar{x}_k^- = \sum_{i=0}^{2n} w_i^m m_{i,k} \tag{5.13}$$

$$\bar{P}_k^- = \sum_{i=0}^{2n} w_i^c (m_{i,k} - \bar{x}_k^-)(m_{i,k} - \bar{x}_k^-)^T + Q_{k-1} \tag{5.14}$$

Update Step

Kalman filters perform the update in measurement space. Thus, the sigma points of the prior are converted into measurements using observation model:

$$y_{i,k-1} = h(m_{i,k-1}) \tag{5.15}$$

With the resulting transformed observations, the mean and covariances are computed for these points. The **y** subscript denotes that these are the mean and covariance of the measurement sigma points.

$$y_{k-1}^- = \sum_{i=0}^{2n} w_i^m y_{k-1} \tag{5.16}$$

$$\bar{P}_{y_{k-1}}^- = \sum_{i=0}^{2n} w_i^c (y_{i,k-1} - \bar{y}_{k-1}^-)(y_{i,k-1} - \bar{y}_{k-1}^-)^T + R_k \tag{5.17}$$

To compute the Kalman gain, the cross covariance of the state and the measurements are identified:

$$P_{x_k,y_{k-1}} = \sum_{i=0}^{2n} w_i^c (m_{i,k} - \bar{x}_k^-)(y_{i,k-1} - \bar{y}_{k-1}^-)^T \tag{5.18}$$

Next, the residual and Kalman gain can be computed as

$$K_k = P_{x_k,y_{k-1}} (\bar{P}_{y_{k-1}}^-)^{-1} \tag{5.19}$$

The new state state estimate can be given as

$$\bar{x}_k = \bar{x}_k^- + K_k(\bar{y}_k - \bar{y}_{k-1}^-) \tag{5.20}$$

and the posterior covariance is computed as

$$P_k = \bar{P}_k^- - K_k \bar{P}_{y_{k-1}}^- K_k^T \tag{5.21}$$

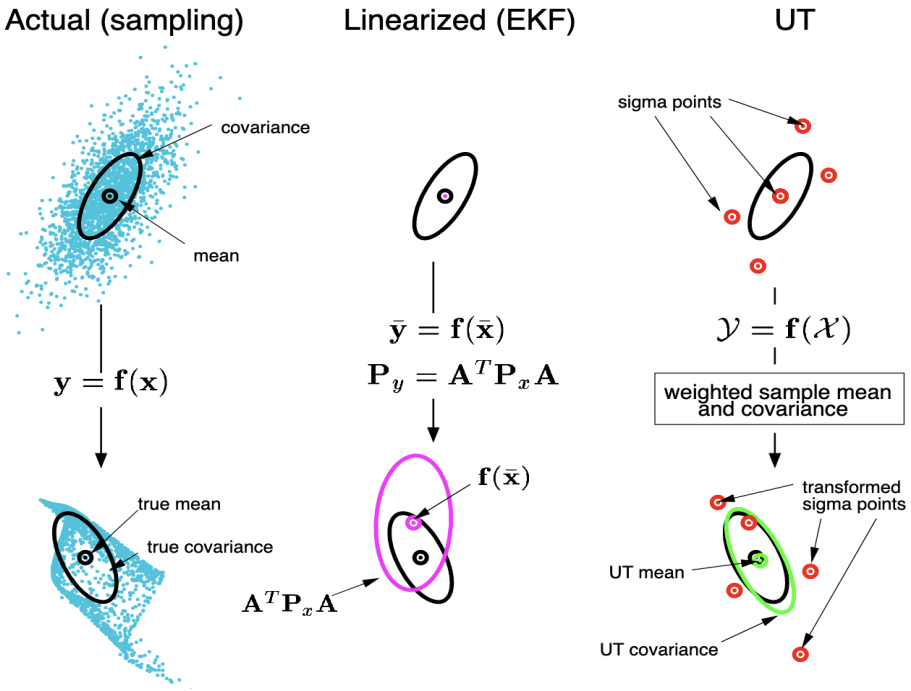


Fig. 5.1 Operation of an Unscented Kalman filter - An Example [15]

5.2 Joint parameter and state estimation

If all parameters of the system dynamics are known *a priori*, then the problem of state estimation can be solved using algorithms such as the particle filter [52] or UKF as described in section 5.1. Good parameter estimation is critical because the model's capacity to effectively derive the process dynamics has a significant impact on the model's quality. In real life, some of the parameters are known and just a few variables fluctuate (and so are unknown *a priori*); nonetheless, this thesis addresses the worst-case situation in which none of the system dynamics parameters are known [53].

There are two general Kalman filter based methods that can simultaneously estimate the unknown states and parameters from noisy measurements - **Dual** and **Joint estimation**. In dual estimation, separate Kalman filters are employed for state and parameter estimation. Figure 5.2 shows how two Kalman filters run in parallel - one adapting the state and the other adapting the parameters, with limited amount of exchange of information between the filters. Decoupling state from parameters results in the loss of any cross-correlations between them, resulting in poor accuracy, even though the computing complexity is lower and the matrix operations may be numerically better conditioned.

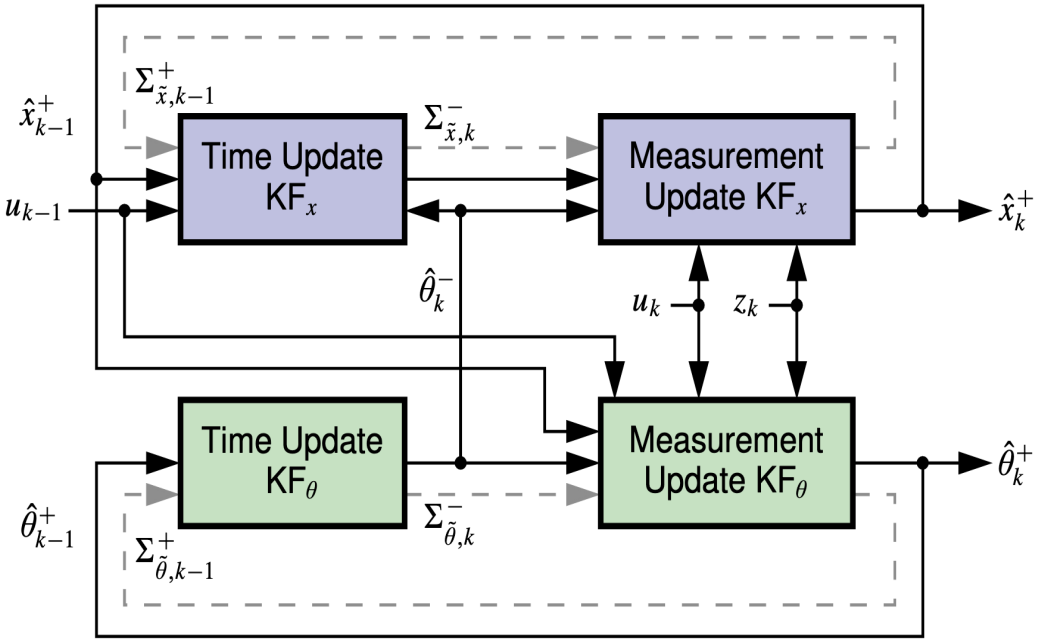


Fig. 5.2 Dual Estimation using KF - Block Diagram [54]

To avoid such issues, this thesis employs joint estimation - more specifically, a **Joint Unscented Rauch Tung Striebel** (JURTS) smoother. In JURTS, the state and the parameter vectors are combined, and JURTS simultaneously estimates the values of this augmented state vector [47, 54]. Using the joint estimation technique in the case of LTI systems (as stated in 2.1) necessitates the employment of a non-linear estimation procedure (hence UKF). Generally, for a n^{th} order LTI system, the aggregated dynamics would result in an extended $2n^{th}$ order non-linear model. For a 4^{th} order system, the dynamics can be represented as

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -a_0 & -a_1 & -a_2 & -a_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x \qquad y = x_1 \tag{5.22}$$

which can also be written as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{a}_0 \\ \dot{a}_1 \\ \dot{a}_2 \\ \dot{a}_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ -a_0x_1 - a_1x_2 - a_2x_3 - a_3x_4 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \qquad y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{5.23}$$

Here in 5.22 and 5.23, a_0, a_1, a_2 and a_3 are the parameters associated with the model and x is the augmented state vector. The interesting thing to note in this setup is that the parameters are being used in the calculation of the states, as they are being predicted. This allows for vacillation in the parameters at each time step (highlighted in the graphs in section 5.3).

The final step in the quest of establishing a framework for the optimal estimation of SISO LTI systems is to compare and contrast the two parameter estimation methods - kernels and JURTS. The following section compares the two joint parameter and state estimate approaches using a system that has been exposed to varied amounts of noise.

5.3 Joint estimation of LTI systems - a fourth order system

Consider the following fourth order SISO LTI system:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1.25 & -3.5 & -4.25 & -3 \end{bmatrix} x \quad ; y = x_1 \quad ; \quad x(0) = [0, 0, 0, 1] \tag{5.24}$$

with its corresponding characteristic equation

$$y^{(4)}(t) + 3y^{(3)}(t) + 4.25y^{(2)}(t) + 3.5y^{(1)}(t) + 1.25y(t) = 0 \tag{5.25}$$

The poles of this system lie at $-1, -1, -0.5 \pm i$ which would make 5.24 a stable fourth-order system. In this experiment, the system is subjected to varying levels of noise and is utilized for testing the joint parameter and state estimation capabilities of the Kernel+projection and JURTS. As is evident, this experiment does not assume that parameters are known beforehand.

Table 5.1 provides an overview of all the variables (and their values) associated with both methods. Please note that the method presented in chapter 3 has been referred to as the 'Kernel+projection' method since the parametric estimation is done by the Kernel (and the regression) equations, as mentioned in 3.6 and the state estimation is done using the projection method (3.7, also experimented with in 4.1).

Estimation Method	**Noise (std. dev.)**	**Variable**	**Value**
Kernel + Projection	All	Time interval	$[0, 12]$
		#Samples	$60,000$
		#Knots	100
		Stopping threshold	0.01
		Initial condition	$[0, 0, 0, 1]$
JURTS	All	Time interval	$[0, 12]$
		#Samples	$60,000$
		Time step	0.0002
		Alpha	0.001
		Beta	2

		Kappa	0
		Initial condition	$[0, 0, 0, 1, 0, 0, 0, 0]$
JURTS	0	P	12
		R	0.0000001
		Q (var)	0.0000005
JURTS	0.2 (-4.46 dB)	P	1
		R	2
		Q (var)	0.00005
JURTS	0.4 (-10.44 dB)	P	1
		R	3
		Q (var)	0.00005

Table 5.1: Variables associated with both the joint parameter and state estimation methods for 5.24 with varying levels of added AWGN noise.

AWGN of $\sigma = 0$ (zero noise)

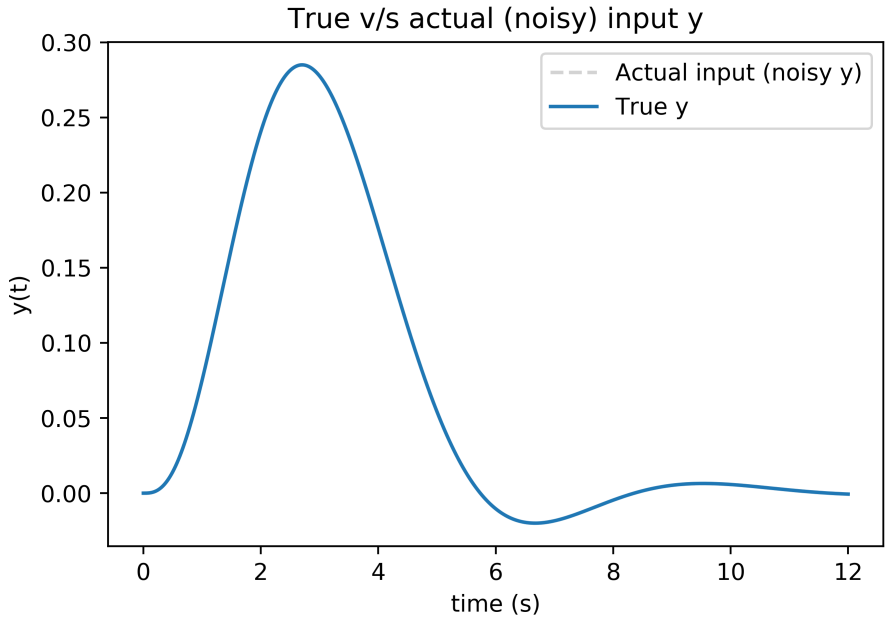
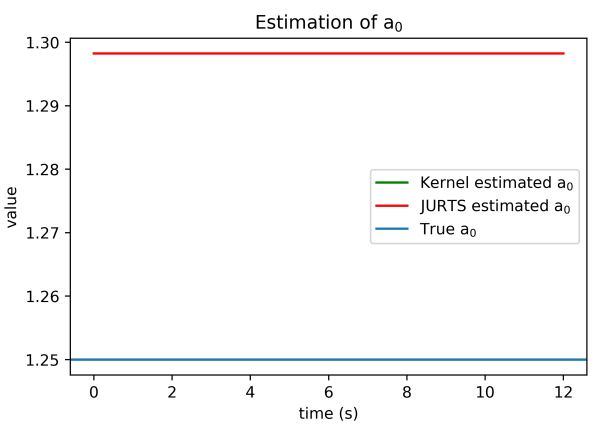
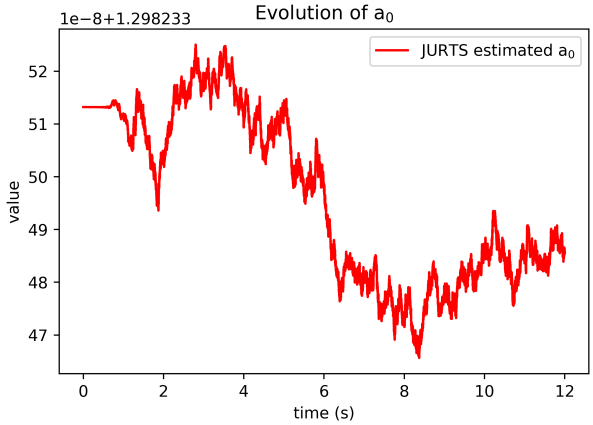


Fig. 5.3 Actual (noisy) input to the algorithms versus true signal

Parameter estimation

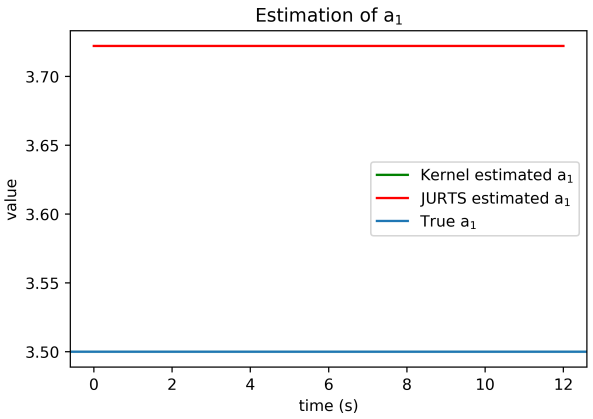


(a) Comparing Kernel and JURTS estimated a_0

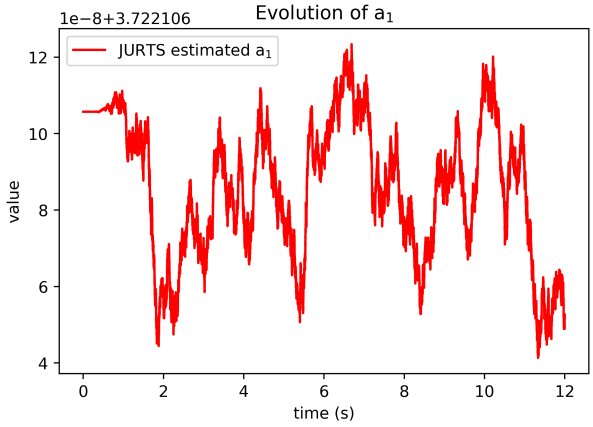


(b) Evolution of a_0 in JURTS

Fig. 5.4 Final parameter estimates - a_0 for 5.24 with 0 noise

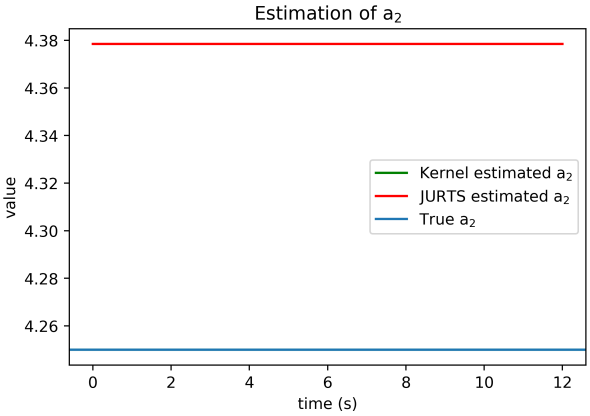


(a) Comparing Kernel and JURTS estimated a_1

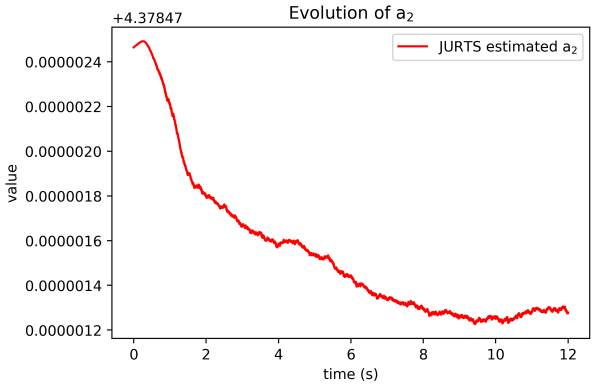


(b) Evolution of a_1 in JURTS

Fig. 5.5 Final parameter estimates - a_1 for 5.24 with 0 noise

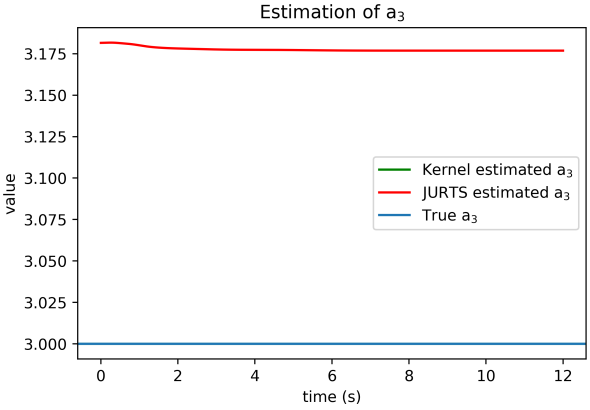


(a) Comparing Kernel and JURTS estimated a_2

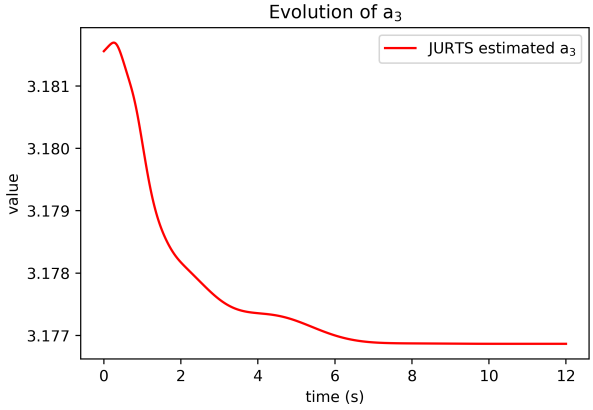


(b) Evolution of a_2 in JURTS

Fig. 5.6 Final parameter estimates - a_2 for 5.24 with 0 noise



(a) Comparing Kernel and JURTS estimated a_3



(b) Evolution of a_3 in JURTS

Fig. 5.7 Final parameter estimates - a_3 for 5.24 with 0 noise

State reconstruction

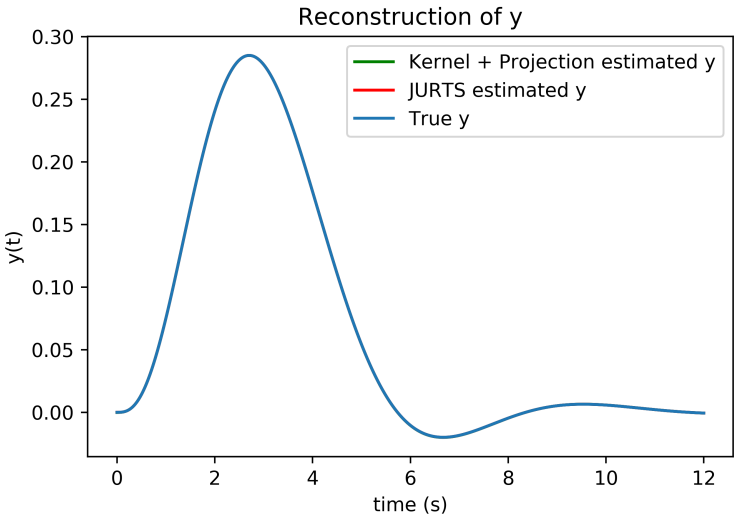


Fig. 5.8 Reconstruction of state y

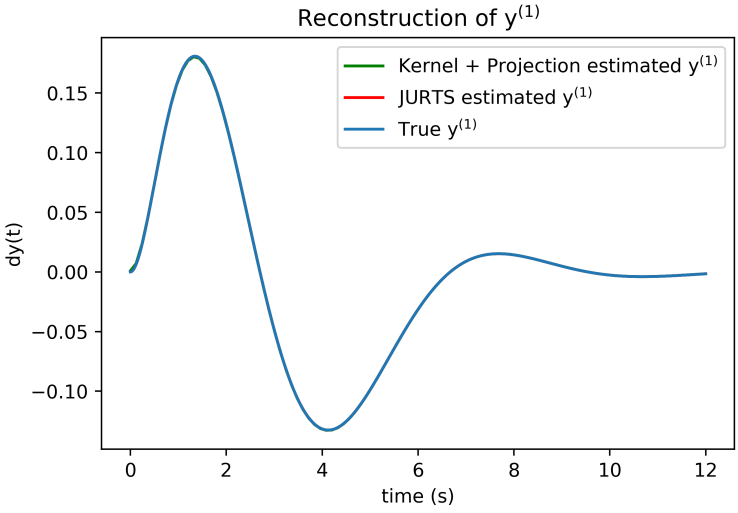


Fig. 5.9 Reconstruction of state $y^{(1)}$

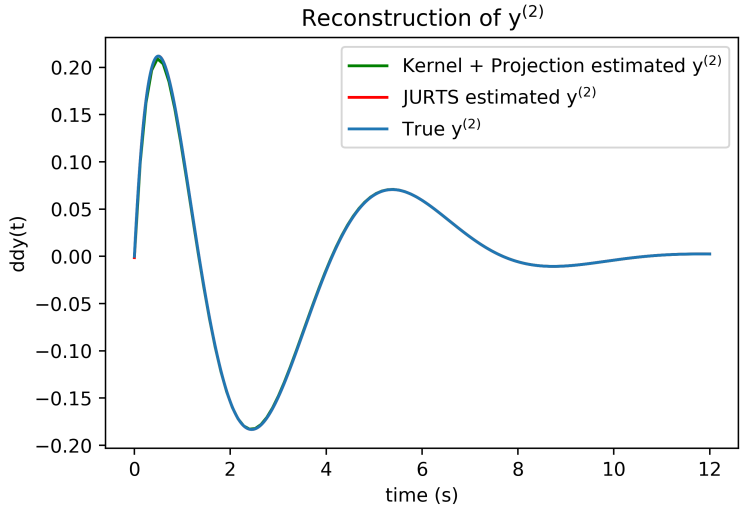


Fig. 5.10 Reconstruction of state $y^{(2)}$

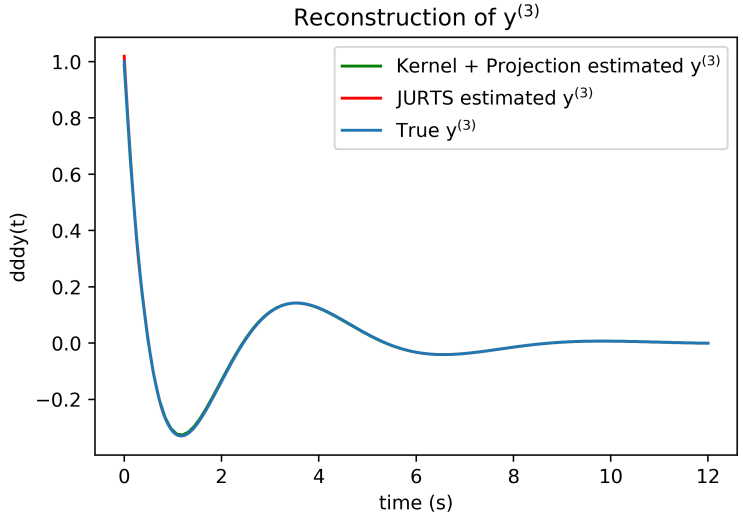


Fig. 5.11 Reconstruction of state $y^{(3)}$

AWGN of $\sigma = 0.2$ (SNR of -4.46 dB)

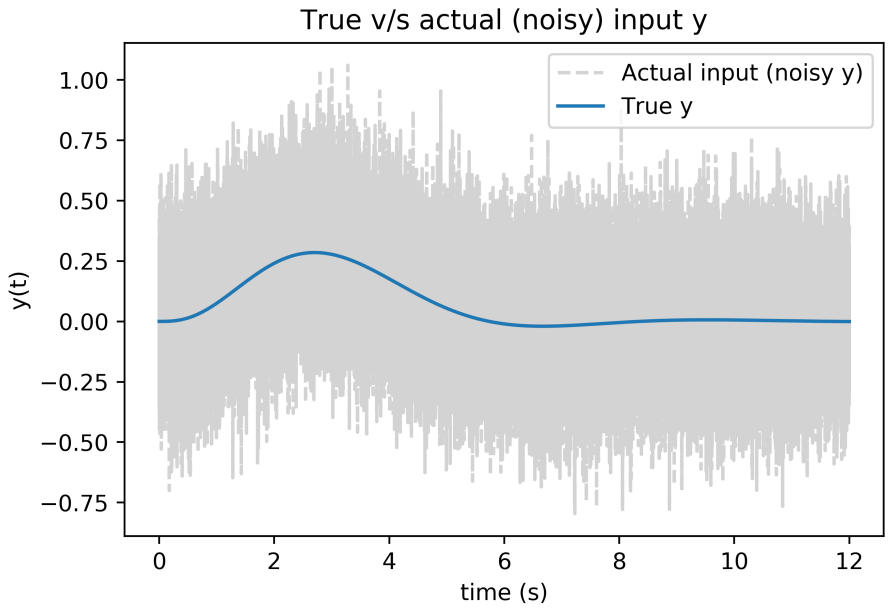
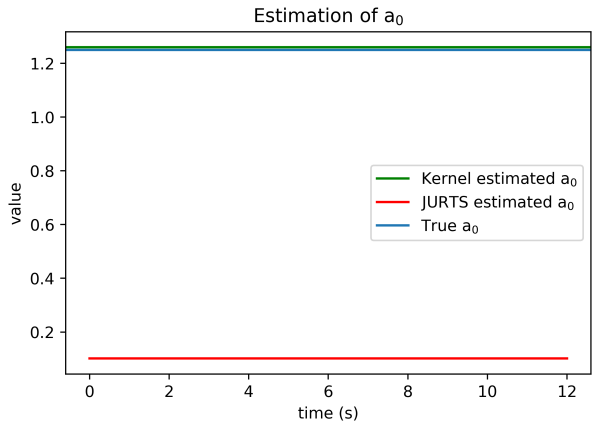
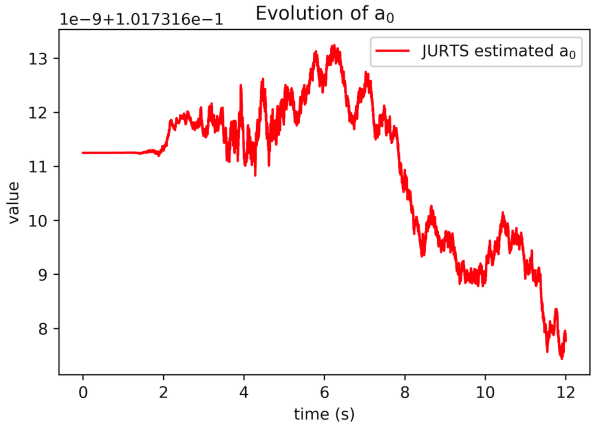


Fig. 5.12 Actual (noisy) input to the algorithms versus true signal

Parameter estimation

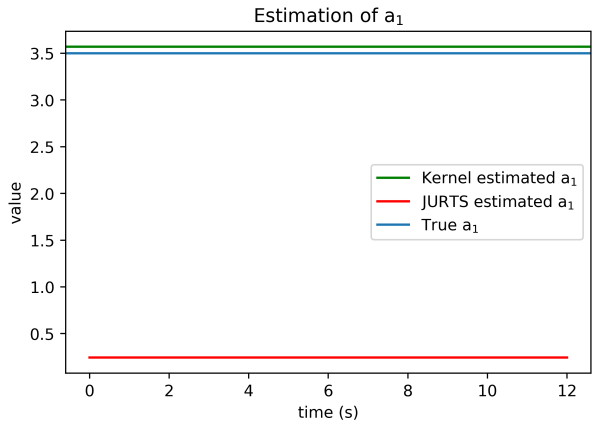


(a) Comparing Kernel and JURTS estimated a_0

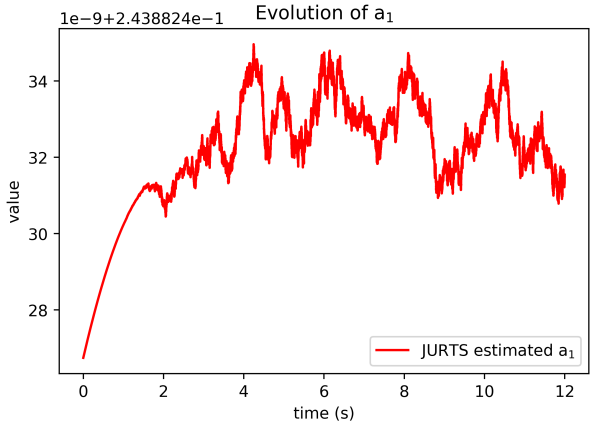


(b) Evolution of a_0 in JURTS

Fig. 5.13 Final parameter estimates - a_0 for 5.24 with $\sigma = 0.2$ noise

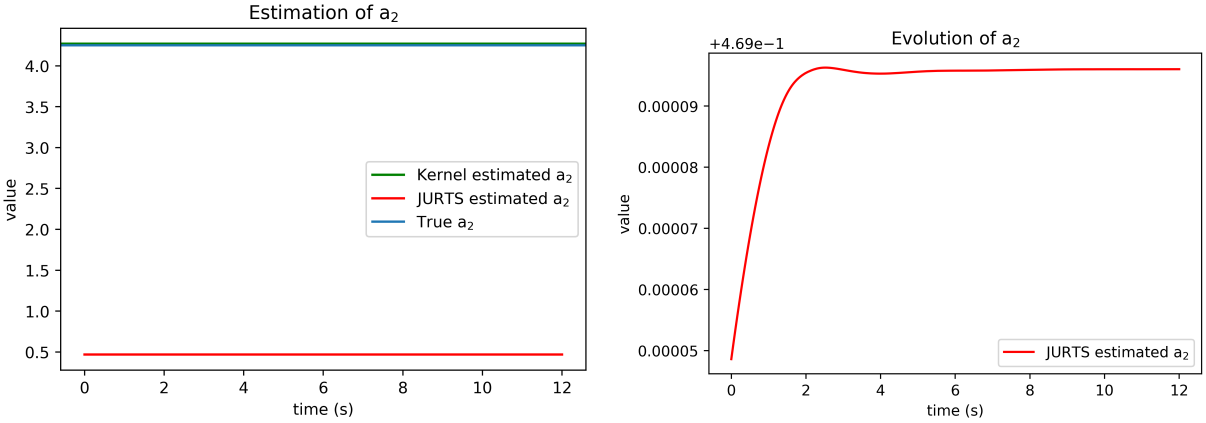


(a) Comparing Kernel and JURTS estimated a_1



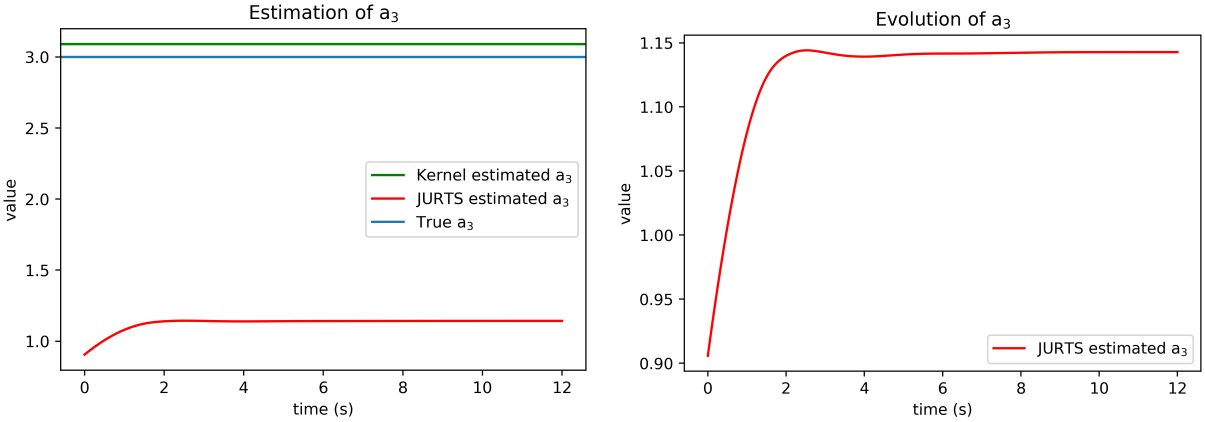
(b) Evolution of a_1 in JURTS

Fig. 5.14 Final parameter estimates - a_1 for 5.24 with $\sigma = 0.2$ noise



(a) Comparing Kernel and JURTS estimated a_2 (b) Evolution of a_2 in JURTS

Fig. 5.15 Final parameter estimates - a_2 for 5.24 with $\sigma = 0.2$ noise



(a) Comparing Kernel and JURTS estimated a_3 (b) Evolution of a_3 in JURTS

Fig. 5.16 Final parameter estimates - a_3 for 5.24 with $\sigma = 0.2$ noise

State reconstruction

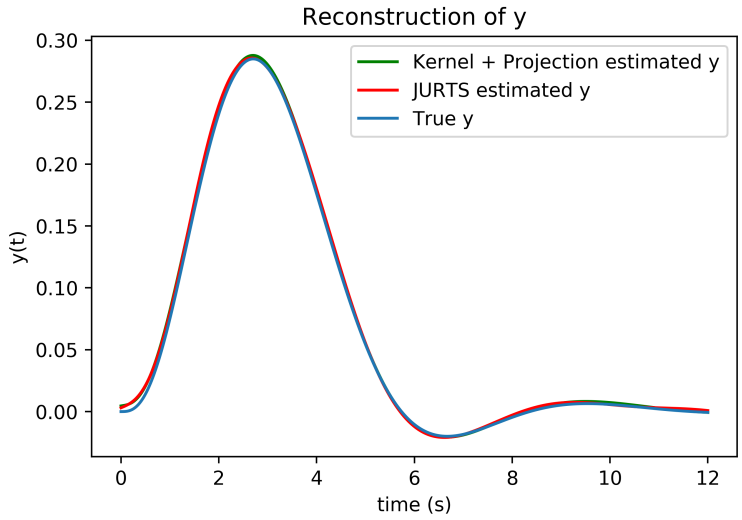


Fig. 5.17 Reconstruction of state y

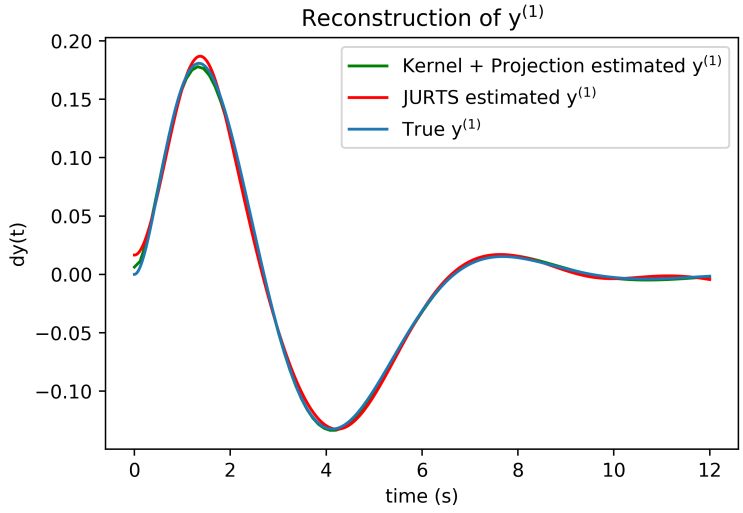


Fig. 5.18 Reconstruction of state $y^{(1)}$

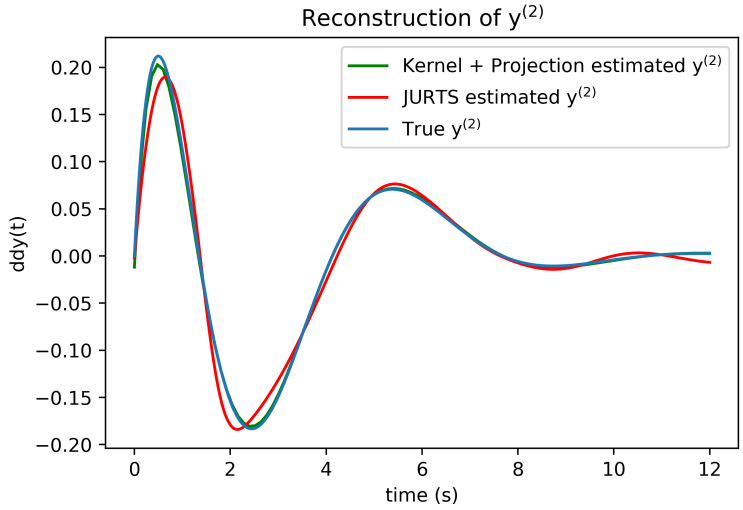


Fig. 5.19 Reconstruction of state $y^{(2)}$

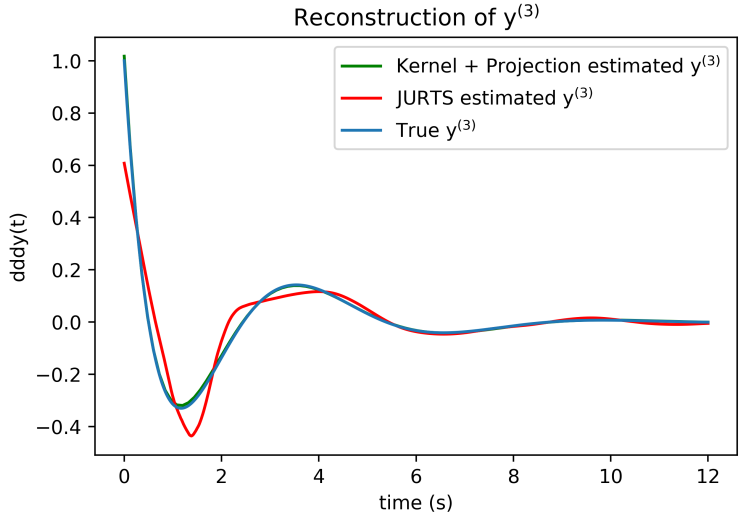


Fig. 5.20 Reconstruction of state $y^{(3)}$

AWGN of $\sigma = 0.4$ (SNR of -10.44 dB)

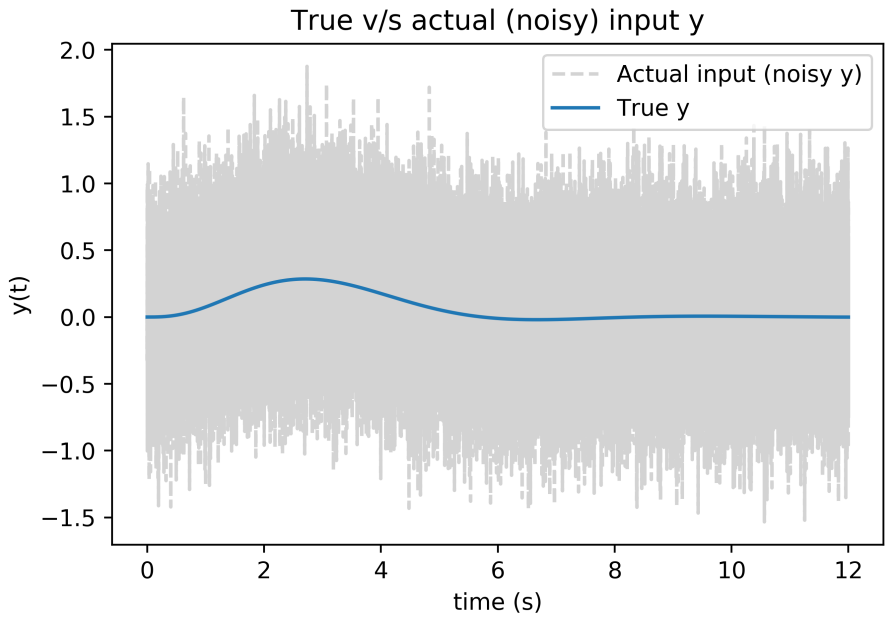
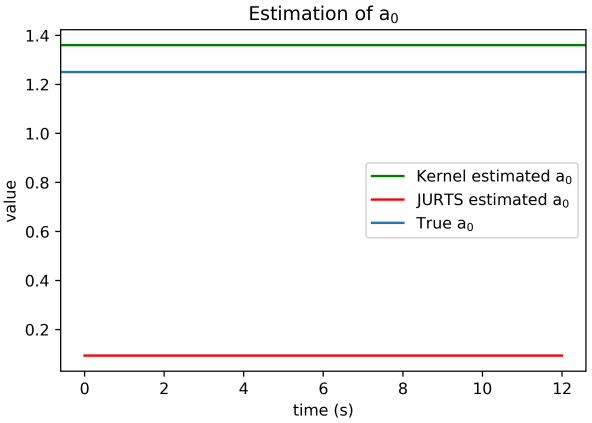
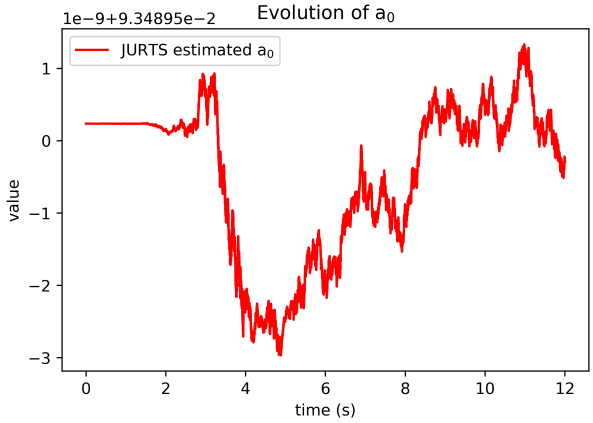


Fig. 5.21 Actual (noisy) input to the algorithms versus true signal

Parameter estimation

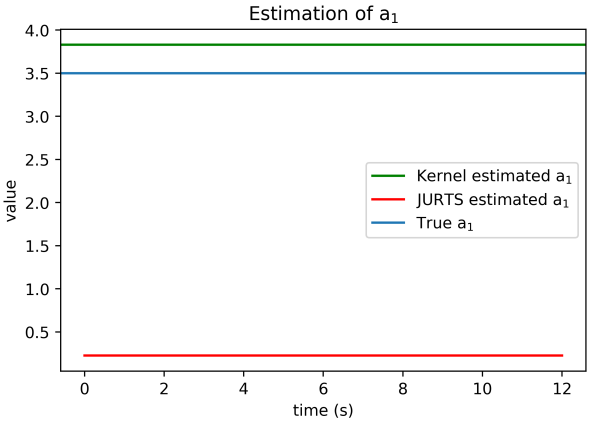


(a) Comparing Kernel and JURTS estimated a_0

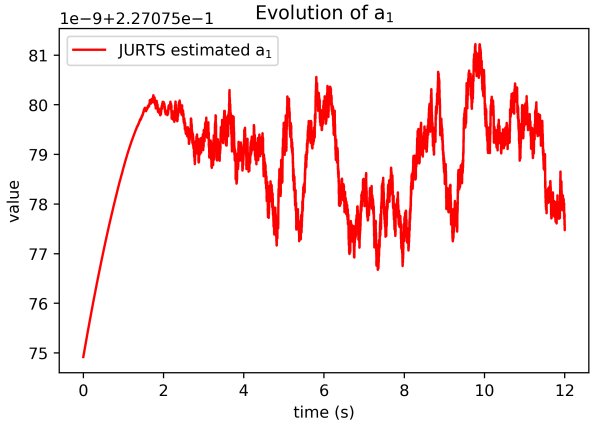


(b) Evolution of a_0 in JURTS

Fig. 5.22 Final parameter estimates - a_0 for 5.24 with $\sigma = 0.4$ noise

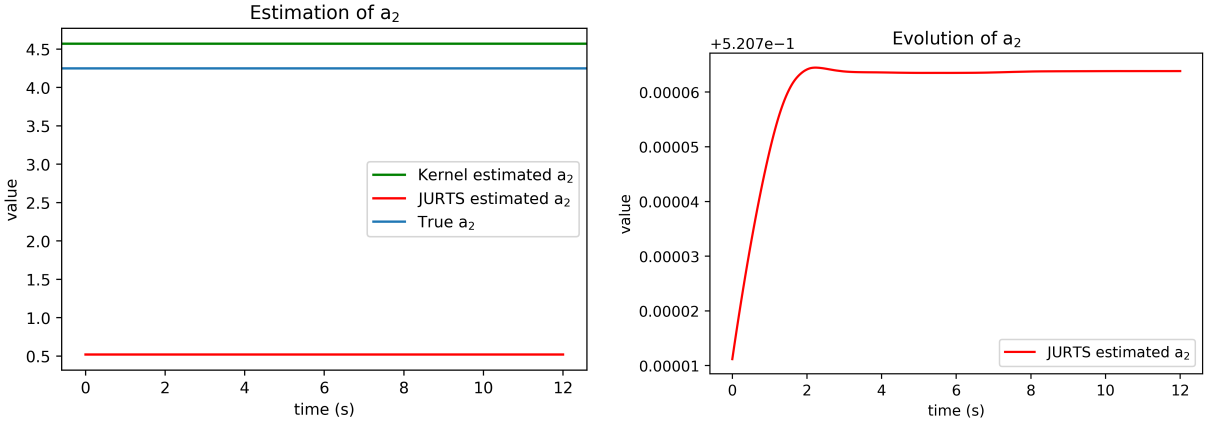


(a) Comparing Kernel and JURTS estimated a_1



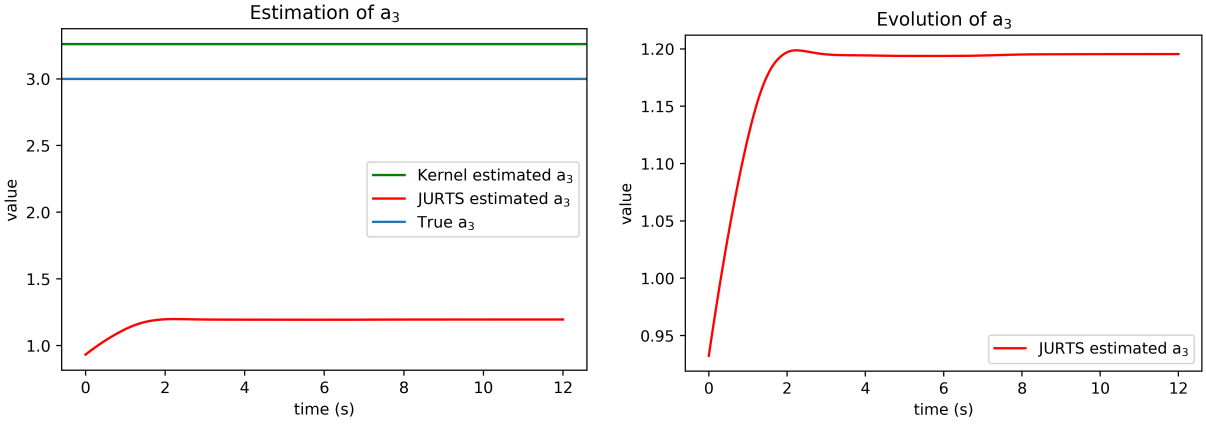
(b) Evolution of a_1 in JURTS

Fig. 5.23 Final parameter estimates - a_1 for 5.24 with $\sigma = 0.4$ noise



(a) Comparing Kernel and JURTS estimated a_2 (b) Evolution of a_2 in JURTS

Fig. 5.24 Final parameter estimates - a_2 for 5.24 with $\sigma = 0.4$ noise



(a) Comparing Kernel and JURTS estimated a_3 (b) Evolution of a_3 in JURTS

Fig. 5.25 Final parameter estimates - a_3 for 5.24 with $\sigma = 0.4$ noise

State reconstruction

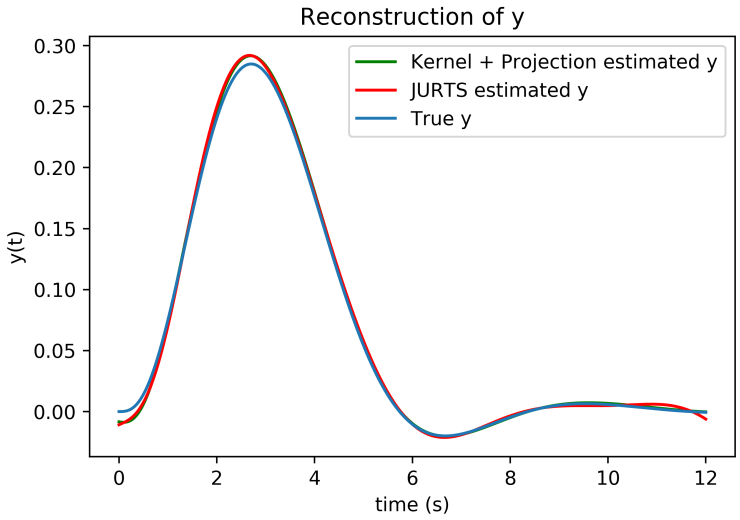


Fig. 5.26 Reconstruction of state y

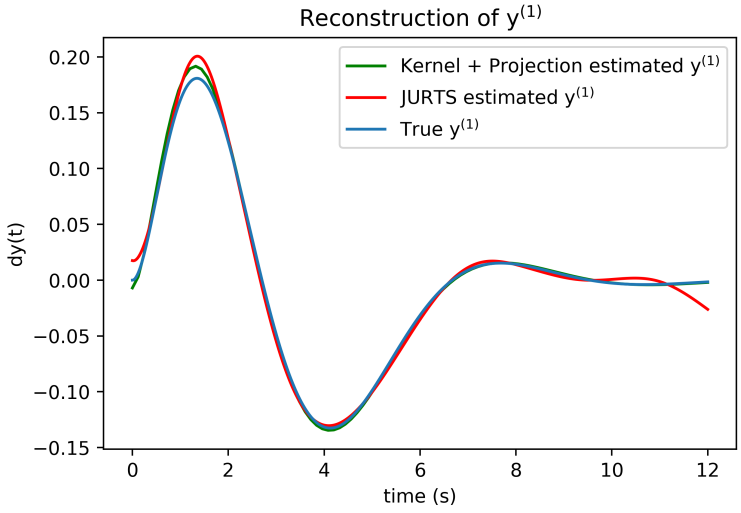


Fig. 5.27 Reconstruction of state $y^{(1)}$

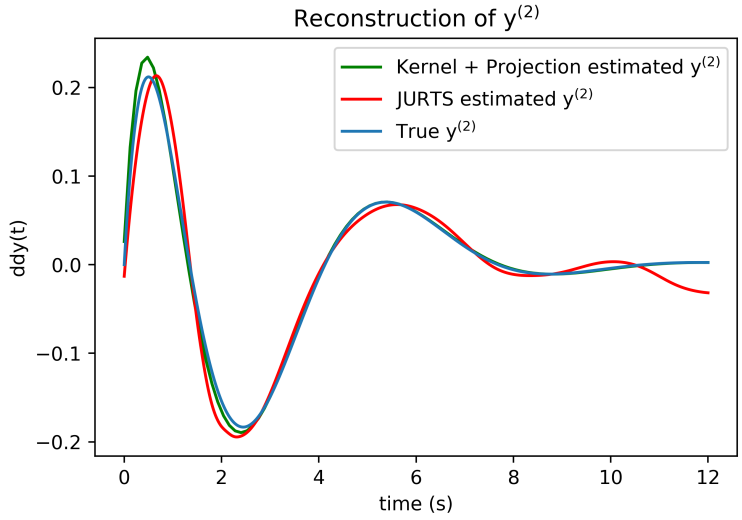


Fig. 5.28 Reconstruction of state $y^{(2)}$

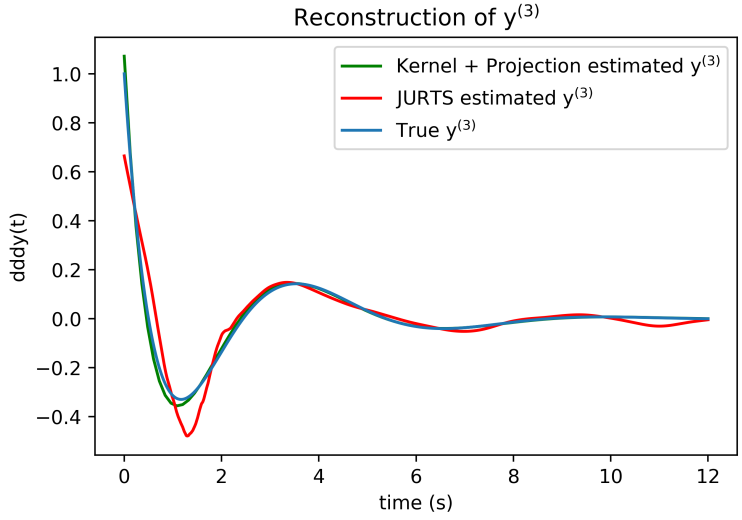


Fig. 5.29 Reconstruction of state $y^{(3)}$

Std. Dev.	**SNR (dB)**	**Estimation Method**	a_0	a_1	a_2	a_3
True value			1.25	3.5	4.25	3
0	0	**Kernel**	1.25	3.5	4.25	3
		JURTS	1.29	3.72	4.37	3.17
0.2	-4.46	**Kernel**	1.29	3.57	4.27	3.09
		JURTS	0.1	0.24	0.47	1.14
0.4	-10.44	**Kernel**	1.36	3.83	4.57	3.26
		JURTS	0.09	0.23	0.52	1.19

Table 5.2: Parameter estimates for 5.24 under different noise levels.

Estimation Method	**Std. Dev.**	**SNR (db)**	**State**	**MAD**	**RMSE**
Kernel+projection	0	0	y	0.0001	0
			$y^{(1)}$	0.0022	0.0003
			$y^{(2)}$	0.0063	0.0013
			$y^{(3)}$	0.0133	0.0019
JURTS	0	0	y	0	0
			$y^{(1)}$	0	0
			$y^{(2)}$	0.0014	0
			$y^{(3)}$	0.0194	0.0017
Kernel+projection	0.2	-4.46	y	0.0067	0.0028
			$y^{(1)}$	0.0071	0.0017
			$y^{(2)}$	0.0158	0.0034
			$y^{(3)}$	0.0205	0.0041
JURTS	0.2	-4.46	y	0.0075	0.0031
			$y^{(1)}$	0.0166	0.0040
			$y^{(2)}$	0.0489	0.0129
			$y^{(3)}$	0.3922	0.0508
Kernel+projection	0.4	-10.44	y	0.0092	0.0035
			$y^{(1)}$	0.0131	0.0037
			$y^{(2)}$	0.0320	0.0070
			$y^{(3)}$	0.0719	0.0141

JURTS	0.4	-10.44	y	0.0110	0.0040
			$y^{(1)}$	0.0246	0.0067
			$y^{(2)}$	0.0463	0.0159
			$y^{(3)}$	0.3354	0.0514

Table 5.3: State estimation error metrics for 5.24 under different noise levels.

It is important to note that the parameters estimated by JURTS in table 5.2 were estimated iteratively. The parameters estimated at time step t were then used to estimate the joint state and parameter vector x at time $t + 1$. This allowed for vacillation in the parameters - as evident in the evolution graphs above.

While the JURTS algorithm presented in this chapter is a decent option when there is no external noise muddling the system, the analysis presented above clearly indicates the superiority of the kernel algorithm as an accurate, efficient, and robust parameter estimation method under any amount of noise. Not just the results in 5.3 but, even the Kalman + RTS filter's state estimation ability demonstrated in 4.1 was powered by the accurate parameter estimates from the kernel algorithm. An additional aspect to take into consideration is the number of times the algorithms were re-run to get to their optimal levels. The kernel algorithm does not need any tuning or modifications to process the variations in the input (noisy) signal. On the other hand, JURTS, inherently being an Unscented Kalman filter, requires a lot of tuning of its vast array of variables to reach its optimal performance (evident from table 5.1).

The primary objective, as stated before, is to establish a basic pipeline of processes that one could leverage to estimate the parameters and state of any SISO LTI system - so far, the thesis has rigorously compared various types of Kalman-based filters against the kernel method. However, as important as theory and experiments are in establishing a clear path, it's also critical to ensure that anyone who wants to use these methods may do so without having to be a coding specialist. As a result, the next chapter discusses a Python-based toolkit that was created specifically for this purpose.

Chapter 6

Python Estimation Toolkits (PETs) [55]

Over the past few years, Dr. Michalska, along with her research group has delved deep into the forward-backward double-sided kernel-based estimation algorithm. The major advantage of this method, which has been explained extensively in this thesis, is that no *a priori* knowledge of the structure of the model, initial conditions, or the statistical characteristics of the system noise is required. This thesis strives to document the algorithm in detail and compares it to traditional Kalman filter-based estimation algorithms to recommend the most optimal sequence of steps one can take to estimate an unknown noisy (measured) signal.

The most important step towards this documentation process was to create a repository on GitHub - a library that essentially hosts all the different techniques mentioned in this thesis. This repository, aptly named Python Estimation Toolkits (PETs) [55] (referred to as 'the repository') was co-developed with Manoj Krishna Ventakesan (research partner) as a tool that could be used by anyone interested in utilizing the powerful algorithm(s) presented in this thesis. The repository, in conjunction with this thesis, would prove to be an effective way to understand the algorithm, the codes, the logic, and the implementation of a novel and classical method of state and parameter estimation.

This chapter begins by laying out all of the prerequisites for running *PETs* on the system. Following it is a detailed explanation of how the repository works, including a detailed account of the different estimation algorithms available. The chapter wraps up with possible use-cases and how the modularity of the repository makes it suitable for extensions and

augmentations.

6.1 Requirements

There are three crucial prerequisites for optimal utilization of the repository. Below is a summary of them all:

1. **Feasible model structure**
   The repository (and this thesis) requires that the system must be able to be represented in a controllable state-space canonical form. All the functions written in the repository 'assume' that the system is in canonical form as shown in section 2.1. This is not uncommon since almost all sub-space estimation methods use this representation

2. **True and noisy signals**
   The repository requires both a noisy measured signal (which needs to be estimated) and the true signal (to calculate the error metrics) to optimally plot all the graphs. The noise present in the signal is assumed to be an Additive White Gaussian Noise (AWGN). The exact functions associated with these tasks will be discussed in section 6.2

3. **Python and additional external libraries**
   This library has been coded completely in python, so a computer with Python installed is necessary. Additionally, the repository utilizes a plethora of external libraries to perform various tasks within each of the different scripts/functions. All the required libraries would be listed in the GitHub repository under `/PETS/requirements.txt` which then could be installed using the console, leveraging the popular pip command

   ```
   pip install requirements.txt
   ```

6.2 Process flow

This section describes the process flow associated with the repository - a summary of which has been illustrated in figure 6.1.

Installation

As already mentioned in section 6.1, after downloading the repository onto the local system, certain additional external python libraries need to be installed before PETs can be used

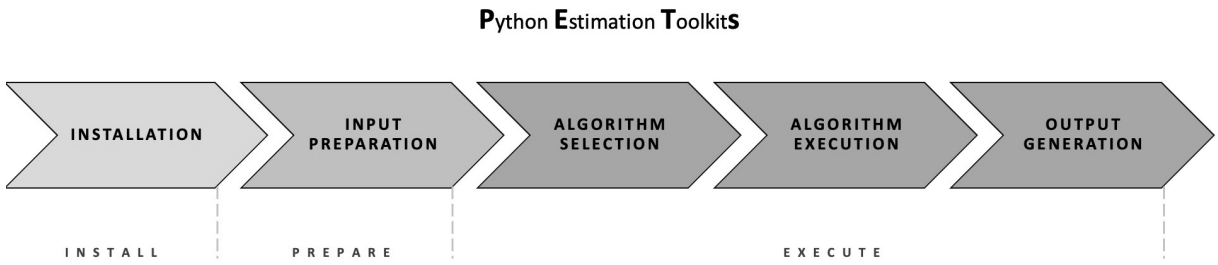


Fig. 6.1 Process flow for the PETs repository - Install, Prepare and Execute

for any estimation problem. Packages such as Pandas, Numpy, Matplotlib, Sklearn, and Scipy provide a set of useful functions packed into one efficient library. A list of all these libraries has been provided in the text file at the root and can be installed using the pip install command.

Access to a console (in-order to run the scripts) and a simple text editor is also necessary.

Input preparation

There are 2 major inputs that the repository requires - the signal file and the configuration file. The signal file, which has been aptly named as *noisy_input.py* contains a simple function in which the user can either, a) import the noisy signal from a file and convert it into a returnable array or b) create their function inside that generates the noisy signal and return that as an array. Having both options gives the user a sense of flexibility and makes it easier to switch from simulated signals into actual real-world noisy data without making any changes to the structure of the repository. In addition to the noisy signal, the function also requires the user to provide the clean (true) signal as well, which will be utilized in carrying out the error calculations and graphing. The file is located at `/PETS/src/pets/noisy_input.py`

The configuration file is a simple JSON dictionary that contains all the user-tunable parameters associated with each of the four different estimation algorithms that the repository provides. This allows for easier access to all the variables without going through the trouble of exploring functions in different scripts. The user can leverage any simple text editor to make changes to these files and save them as is. The configuration file can be found at `/PETS/configs/config_*algorithm*.json`. Figure 6.2 shows an example of what the configuration file would look like - in this case, for a Kalman filter. Every 'key' in this JSON dictionary is a variable that would assist in running the Kalman filter and every 'value' is a tunable value for the associated key. Every algorithm will have one such file associated with

it.

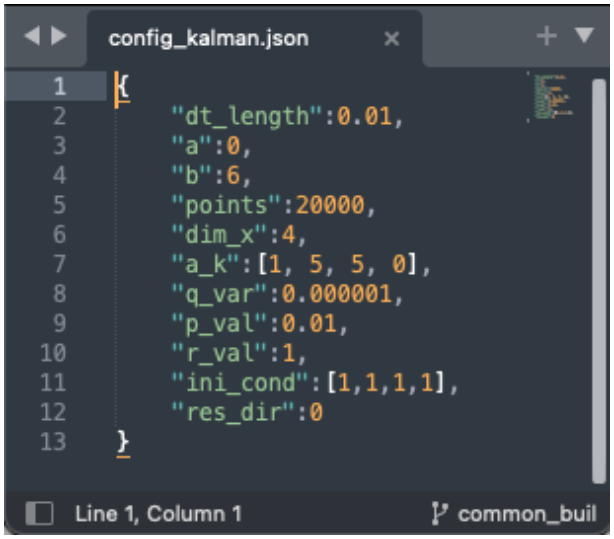


Fig. 6.2 Config file: An example

Algorithm selection

After the inputs have been all setup, the user can now select the type of algorithm that they wish to execute. This can be done by opening up the console (or CLI), navigating to `/PETS/scripts/` and then running

```
python run_estimation.py -m *method*
or
python3 run_estimation.py -m *method*
```

This runs the script that calls upon different functions based on the *-m method* value selected.

Algorithm execution

The user has the option of selecting one of the four available *-m method* values in the `run_estimation.py` call command, each one of which executes a different algorithm-

- *kernel_projection*: Uses the kernel method to estimate both the system parameters which are then used by the projection method to estimate the states
- *kalman_statesonly*: Uses Kalman + RTS filter to estimate the state of the system, given the system parameters

- *kalman_ukf*: Uses Unscented Kalman filter + RTS to estimate the state and the system parameters
- *kernel_kalman*: Uses the parameters calculated from the kernel algorithm to find the state of the system using Kalman + RTS filter

The primary script, `run_estimation.py` can be found under `/PETS/scripts` and all the source codes associated with each of these algorithms are stored under `/PETS/src/pets/`. The folder structure diagram in figure 6.3 will make navigation easier inside the repository -

```
PETS/
├── configs/
│   ├── config_*algorithm*.JSON
│   ├── README.md
├── data/
├── resources/
│   ├── reference thesis/
│   └── other resources/
├── scripts/
│   ├── run_estimation.py
│   └── README.md
├── src/
│   ├── pets/
│   │   ├── noisy_input.py
│   │   ├── gen_results.py
├── requirements.txt
├── README.md
```

Fig. 6.3 Folder structure - PETs repository

Output generation

In order to assess the performance of the estimation algorithms, PETs generates two key metrics for evaluation - root mean squared error (RMSE) and maximum absolute deviation (MAD). Along with the exact error values, it also generates a text file containing all the values at each time step, of both the true and the estimated signal (along with its derivatives) in a simple text file. Employing the true value of the signal that's provided by the user, and comparing it with the estimated signal from the algorithm engenders a true summary of the effectiveness of the process.

In addition to the error metrics and the value log, the repository also outputs a number of

graphs comparing the original true signal with the estimated signal, along with its derivatives. This is all done by `gen_results.py` which is located at `/PETS/src/pets/gen_results.py` and is automatically triggered once the estimation process is complete. This script takes the directory (on the local machine) that's been provided in the configuration file and creates four types of files

1. *rmse_mad.txt* file: This file contains the rmse and mad values comparing the true and the estimated signal from the algorithm
2. **.png* files: Multiples files containing graphs of true versus estimated - for both the actual signal and its derivatives
3. *state_estimates.tsv* file: This file contains the log of all values of the signals at every time instance in a simple tab separated format, which is machine readable - this can then be used to generate more analysis or graphs
4. *parameter_estimates.tsv* file: This file contains the final estimated parameters of the system (only available where the parameters have not been provided in the configuration files)

6.3 Estimation algorithms

While the entire thesis focuses on the mathematical and logical aspects of these algorithms, it is also critical to discuss their actual implementation. This section focuses on providing a brief overview of the technical implementation of two such algorithms. The implementation of the other two kernel based algorithms have been explained in Manoj Krishna Ventakesan's (research partner) thesis document [23].

Kalman with known system parameters

Command to execute: `python run_estimation.py -m kalman_statesonly`

This algorithm uses an optimized Kalman filter package called `filterpy` [31] which provides a various range of functions to implement both the simple linear Kalman filter, as well as the RTS smoother on top of it. Based on the various parameters fetched from the configuration file `config_kalman.json` the script `run_kalman.py` identifies the order and engenders the estimated signal using `kalman_known.py`, along with its derivatives that are then sent to `gen_results.py` where all the value logs, error metrics and graphs are computed and stored

in the given directory.

Looking into the configuration file for this algorithm, as shown in figure 6.2 *dt_length*, *a*, *b* and *points* are the same terms as defined in chapter 4. Variable *dim_x* is the dimension of the input signal, *a_k* [array] is the system parameter vector, *q_var* is the process noise variance, *p_var* is the co-variance multiple (multiplied with a identity matrix of dimension *dim_x*), *r_val* is the measurement noise and *ini_cond* [array] gives the initial conditions that the algorithm must assume for all the states. The last variable *res_dir* [string] is the directory on the local system, where all the results would be stored.

Kalman with unknown system parameters

Command to execute: `python run_estimation.py -m kalman_ukf`

This algorithm uses the same Kalman filter package `filterpy` [31] used in the algorithm above but this time there is no apriori knowledge about the system parameters. Building of the joint state and parameter vector is done in the script so its easier to implement it on the user-end. Parameters are fetched from `config_kalman_unknown.json` by `run_kalman_unknown.py` which in turn triggers `kalman_unknown.py` to calculate the joint vector and estimate the vector using JURTS. `gen_results.py` takes all this information and outputs the results into the specified directory.

The configuration file for this algorithm contains all the keys as the regular Kalman filter (without the *a_k*). Apart from this, variable *alpha* is used to determine the spread of sigma points around the mean, *beta* is a factor that is associated with the incorporation of prior knowledge of the distribution of the mean and *kappa* is a secondary scaling parameter that is usually set to 0. One crucial thing to remember in case of Kalman with unknown system parameters is to be careful about the dimension of the *ini_cond* array - if the value of *dim_x* is n then the length of *ini_cond* array should be $2n$. This is because (as previously stated) the script combines the state and parameter vectors together and the initial condition is used to initiate this combined (joint) vector.

6.4 Uses and future modifications

Beyond the obvious objective this repository was created to fulfill, PETs also would prove to be a worthy starting point for any analysis where estimation is involved. The mix of both novel and conventional methods for solving the same problem, in the same place is

bound to be useful. The package has also been designed in such a way that each part of the 'puzzle' can be broken down and tinkered with to add additional functionalities easily. For instance, some of the other research students under Dr.Michalska have worked on extending the forward-backward kernel to non-linear problems which were then used on MIMO systems as well as a multitude of control-related problems [17, 25]. This repository can be used for those problems as well - thus increasing its effectiveness over an even wider range of problems. Adding support for constraints (explained in chapter 7) and extension to non-linear systems would be the ideal next steps for this repository.

The repository also contains various support documents that give more insights and tips/tricks to navigate through all the code - READMEs in every folder, a separate resources tab that links all the brilliant thesis documents from Dr.Michalska's group (including this one) along with additional documents that were used in the creation of this thesis and the PETs repository.

Chapter 7

Constrained Kalman Filtering

Filtering problems are typically complex and non-linear, and a wide range of real-world problems contain an intrinsic state-space equality constraint. The advantages of incorporating constraints far outweigh the computing expenses of the approach. There are two main approaches to do this: first, by adding the equality constraint to the filter's measurement space at each epoch, and second, by finding the unconstrained estimate from a Kalman filter and projecting it to the equality constrained space. The sections that follow provide an overview of both of these approaches. [56–58].

7.1 Constraints using augmented measurement space [56]

This method involves observing the constraints at every iteration as a *noise-free measurement*. Consider a system defined in a structure that is conducive for utilizing Kalman filter(s):

$$x_k = F_{k,k-1}x_{k-1} + u_{k,k-1}, \quad u_{k,k-1} \sim N\left(0, Q_{k,k-1}\right) \tag{7.1}$$

$$z_k = H_k x_k + v_k, \quad v_k \sim N\left(0, R_k\right) \tag{7.2}$$

where x_k is the state of the system, $F_{k,k-1}$ is the transition dynamics matrix that translates the system from x_{k-1} to x_k (an alternative representation of the A matrix). z_k is the observed measurement which in equation (7.1) is given by the product of the state and the transformation matrix H_k. The noise terms $u_{k,k-1}$ and v_k encompass known and unknown errors in $F_{k,k-1}$ and H_k and are normally distributed with mean 0 and variances $Q_{k,k-1}$ and R_k, respectively.

Assuming well defined constraints without any null solutions - i.e., D_k has full row rank, constraints could be formulated as

$$D_k x_k = \delta_k \tag{7.3}$$

Equations (7.1) and (7.2) can be re-written as

$$x_k^D = F_{k,k-1} x_{k-1}^D + u_{k,k-1}, \quad u_{k,k-1} \sim N\left(0, Q_{k,k-1}\right) \tag{7.4}$$

$$z_k^D = H_k^D x_k^D + v_k^D, \quad v_k \sim N\left(0, R_k^D\right) \tag{7.5}$$

Using the superscript D, the equations representing the construction of the augmentation in the measurement space can be given as

$$z_k^D = \begin{bmatrix} z_k \\ \delta_k \end{bmatrix} \tag{7.6}$$

$$H_k^D = \begin{bmatrix} H_k \\ D_k \end{bmatrix} \tag{7.7}$$

$$R_k^D = \begin{bmatrix} R_k & 0 \\ 0 & 0 \end{bmatrix} \tag{7.8}$$

This augmentation forces $D_k x_k^D = \delta_k$ to be equal at every iteration. Utilizing this, all the Kalman filter prediction and updation equations can be rewritten - the detailed equations are given in [56].

7.2 Constraints by projection [56]

This method of incorporating projections is very intuitive - first, an unconstrained Kalman filter is run and then, the estimates obtained are projected onto the constrained space at each iteration. For a given time step k, the minimization problem formed can be described as -

$$\hat{x}_{k|k}^P = \arg\min_x \left\{ \left(x - \hat{x}_{k|k}\right)' W_k \left(x - \hat{x}_{k|k}\right) : D_k x = \delta_k \right\} \tag{7.9}$$

where, $\hat{x}_{k|k}^P$ is the constrained estimate, $\hat{x}_{k|k}$ is the unconstrained estimate (output from the Kalman filter) and W_k is a chosen positive definite symmetric weighing matrix.

The optimal constrained estimate can be given as

$$\hat{x}^P_{k|k} = \hat{x}_{k|k} - W_k^{-1} D_k' \left(D_k W_k^{-1} D_k' \right)^{-1} \left(D_k \hat{x}_{k|k} - \delta_k \right) \tag{7.10}$$

As it turns out, plugging in W_k as $P^{-1}_{k|k}$ gives the most natural solution that perfectly describes the uncertainty of the state.

Nonlinear equality constraints

To extend the concept of equality constraints into the domain of non-linearity, the linear constraint is replaced by

$$d_k(x_k) = \delta_k \tag{7.11}$$

where $d_k(\cdot)$ is a vector valued function.

Linearizing (7.11) about the current state prediction $\hat{x}_{k|k-1}$ gives,

$$d_k \left(\hat{x}^P_{k|k-1} \right) + D_k \left(x_k - \hat{x}^P_{k|k-1} \right) \approx \delta_k \tag{7.12}$$

where D_k is defined as the Jacobian of d_k evaluated at $\hat{x}^P_{k|k-1}$ which implies that the nonlinear constraint can be approximated as

$$D_k x_k \approx \delta_k + D_k \hat{x}^P_{k|k-1} - d_k \left(\hat{x}^P_{k|k-1} \right) \tag{7.13}$$

While the research work only involved a small amount of experimentation on constraints, preliminary results employing the derivative of the characteristic equation as an equality constraint showed that this is an essential subtopic to pursue.

Conclusion

This thesis initially presents the conventional Kalman filter approach for state estimation from a measured input (noisy) signal, smoothed by an RTS smoother. This was scrupulously compared against a novel joint state and parameter estimation approach that uses forward-backward double-sided kernels which were modified and optimized based on the work done by [24, 38]. The MRRLS algorithm referred to in this thesis as the 'kernel' algorithm, employs a smart initial estimate to improve upon the recursive least squares algorithm presented by [24].

However, the predominant contribution of this thesis is the development of a *roots-to-fruits* estimation pipeline dedicated to providing a versatile, modular, and accessible approach to filtering and estimation of SISO LTI systems. This involved the development of a Python-based toolkit that hosts all the estimation algorithms discussed in this thesis. A full comparison of all of these algorithms' performances was required in addition to the theory behind them; this would allow the reader to make an informed decision on which strategies to use to solve their estimation problem.

Recommendations and future work

- The comparison of the projection and Kalman + RTS approach for state estimation demonstrates that the Kalman-based method performs better in the case of known system parameters and noise dynamics. The projection approach, on the other hand, might a faster way to generate adequate accurate estimations quickly - unlike the Kalman-based method, which requires repetitive tuning of all variables

- Kernels, on the other hand, have shown their prowess over the Kalman-based JURTS filter (in this thesis) and other methods (as demonstrated in [24, 41]) as the superior parameter estimation methodology. The results in 4.1 and 5.3 represent how versatile the algorithm is - kernel estimates fit perfectly with other algorithms, even when

muddled by high noise

- PETs is an immensely powerful library that currently covers all the algorithms mentioned in the thesis. Specifically designed to be operated without the need to be know everything *in-and-out*, the pipeline established throughout this thesis heavily relies on the library being the platform for all the implementation

- Based on the condition and knowledge of the system/noise dynamics, the following can be clearly inferred

  No/low *noise*, **full** *knowledge* : While both projection and Kalman+RTS work well, the results clearly indicate that the Kalman-based method would work best for state estimation. This can be done using the `kalman_statesonly` function in the library, which leverages the user-defined parameters specified in the `config_kalman_statesonly` configuration file

  High/very high *noise*, **full** *knowledge* : Although the Kalman+RTS filter takes a lot of adjustment, the results suggest that it outperforms the projection approach even at high noise levels when the estimated coefficients are strong

  No/low *noise*, **no/partial** *knowledge* : The pioneering kernel+projection approach surpasses all previous methods when it comes to parameter estimation and utilizing them for state estimation. The kernel parameter estimations are strong enough to yield excellent state estimates using the projection approach for low noise levels.

  High/very high *noise*, **no/partial** *knowledge* : Using the best of both worlds, the optimal solution can be obtained by first obtaining parameter estimates using kernels and then generating state estimates using the Kalman + RTS filter. Function `kernel_kalman` in the library can be utilized to implement this method

- While the kernel estimation method's robustness and superiority have been eloquently demonstrated throughout this thesis, Chapter 7 provides a quick overview of constraints and how they can improve the accuracy of estimates. This could be one of the main areas where more research could be done. Because of its modular nature, the PETs library could be extended to add functionality, such as efficient non-linear estimates, extension to specific MIMO systems, and a downstream controller configuration

References

[1] Lotfi A Zadeh. From circuit theory to system theory. *Proceedings of the IRE*, 50(5):856–865, 1962.

[2] Karl Johan Åström and Peter Eykhoff. System identification—a survey. *Automatica*, 7(2):123–162, 1971.

[3] Gaetan Kerschen, Keith Worden, Alexander F Vakakis, and Jean-Claude Golinval. Past, present and future of nonlinear system identification in structural dynamics. *Mechanical systems and signal processing*, 20(3):505–592, 2006.

[4] Ward Heylen, Stefan Lammens, Paul Sas, et al. *Modal analysis theory and testing*. Katholieke Universiteit Leuven Leuven, Belgium, 1997.

[5] David J Ewins. *Modal testing: theory, practice and application*. John Wiley & Sons, 2009.

[6] Rudolf Emil Kalman. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 1(2):152–192, 1963.

[7] Ronald Alexander, Gilson Campani, San Dinh, and Fernando V Lima. Challenges and opportunities on nonlinear state estimation of chemical and biochemical processes. *Processes*, 8(11):1462, 2020.

[8] Carl Friedrich Gauss. *Theory of the motion of the heavenly bodies moving about the sun in conic sections: a translation of Carl Frdr. Gauss" Theoria motus": With an appendix. By Ch. H. Davis*. Little, Brown and Comp., 1857.

[9] Daniel W Stroock. *An introduction to Markov processes*, volume 230. Springer Science & Business Media, 2013.

[10] AM Yaglom. Outline of some topics in linear extrapolation of stationary random processes. In *Proc. Fifth Berkeley Symp. Math. Stat, and Prob.*, pages 259–278, 1970.

[11] K Vastola and H Poor. Robust wiener-kolmogorov theory. *IEEE Transactions on Information Theory*, 30(2):316–327, 1984.

[12] Mohinder S Grewal and Angus P Andrews. *Kalman filtering: Theory and Practice with MATLAB*. John Wiley & Sons, 2014.

[13] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[14] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[15] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[16] Xue-Bo Jin, Ruben Jonhson RobertJeremiah, Ting-Li Su, Yu-Ting Bai, and Jian-Lei Kong. The new trend of state estimation: from model-driven to hybrid-driven methods. *Sensors*, 21(6):2085, 2021.

[17] Adharsh Mahesh Kumaar. Comparison of different online kernel and kalman filter based estimation and filtering methods for siso and mimo non-liner systems. Master's thesis, McGill University, 2022.

[18] Debarshi Patanjali Ghoshal, Kumar Gopalakrishnan, and Hannah Michalska. Using invariance to extract signal from noise. In *2017 American Control Conference (ACC)*, pages 2588–2593. IEEE, 2017.

[19] Debarshi Patanjali Ghoshal, Kumar Gopalakrishnan, and Hannah Michalska. Algebraic parameter estimation using kernel representation of linear systems. *IFAC-PapersOnLine*, 50(1):12898–12904, 2017.

[20] Debarshi Patanjali Ghoshal, Kumar Gopalakrishnan, and Hannah Michalska. Kernel-based adaptive multiple model target tracking. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1338–1343. IEEE, 2017.

[21] Debarshi Patanjali Ghoshal and Hannah Michalska. Finite interval estimation of lti systems using differential invariance, instrumental variables, and covariance weighting. In *2020 American Control Conference (ACC)*, pages 731–736. IEEE, 2020.

[22] Debarshi Patanjali Ghoshal and Hannah Michalska. Finite-interval kernel-based identification and state estimation for lti systems with noisy output data* this work was supported by the national science & engineering research council of canada. In *2019 American Control Conference (ACC)*, pages 4982–4989. IEEE, 2019.

[23] Manoj Krishna Venkatesan. Finite time joint parameter and state estimation using python. Master's thesis, McGill University, 2022.

[24] Shantanil Bagchi. Finite-time identification and estimation of non-linear system dynamics using kernel-based sliding window. Master's thesis, McGill University, 2021.

[25] Luis Medrano del Rosal. Model-free predictive controller using double-sided kernels for system identification. Master's thesis, McGill University, 2022.

[26] David G Luenberger. Observing the state of a linear system. *IEEE transactions on military electronics*, 8(2):74–80, 1964.

[27] Peter S Maybeck. The kalman filter: An introduction to concepts. In *Autonomous robot vehicles*, pages 194–204. Springer, 1990.

[28] Peter S Maybeck. *Stochastic models, estimation, and control*. Academic press, 1982.

[29] Mohinder S. Grewal and Angus P. Andrews. *Optimal Smoothers*, pages 183–223. Wiley–Blackwell, 2008.

[30] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[31] Roger R. Labbe Jr. Filterpy. `https://github.com/rlabbe/filterpy`, 2015.

[32] K Biswas and A Mahalanabis. Optimal fixed lag smoothing for time delayed system with colored noise. *IEEE Transactions on Automatic Control*, 17(3):387–388, 1972.

[33] John B Moore. Discrete-time fixed-lag smoothing algorithms. *Automatica*, 9(2):163–173, 1973.

[34] Roland Priemer and André G Vacroux. On smoothing in linear discrete systems with time delays. *International Journal of Control*, 13(2):299–303, 1971.

[35] Herbert E Rauch, F Tung, and Charlotte T Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450, 1965.

[36] Thomas Kailath, Ali H Sayed, and Babak Hassibi. *Linear estimation*, volume BOOK. Prentice Hall, 2000.

[37] B Anderson. Properties of optimal linear smoothing. *IEEE Transactions on Automatic Control*, 14(1):114–115, 1969.

[38] Debarshi Patanjali Ghoshal and Hannah Michalska. Forward-backward kernel-based state and parameter estimation for linear systems of arbitrary order. *arXiv preprint arXiv:2012.09033*, 2020.

[39] Debarshi Patanjali Ghoshal. *Finite-interval estimation using double-sided kernels and differential invariants*. Thesis, McGill University, Montreal, December 2020.

[40] Santosh Devapati. Kernel-based generalized least squares for joint state and parameter estimation in lti systems. Master's thesis, McGill University, 2020.

[41] Anju John. Estimation for siso lti systems using differential invariance. Master's thesis, McGill University, 2019.

[42] Augustin-Louis Cauchy. *Résumé des leçons données à l'école royale polytechnique sur le calcul infinitésimal*. Imprimerie royale, 1823.

[43] S Stanhope, Jonathan E Rubin, and David Swigon. Identifiability of linear and linear-in-parameters dynamical systems from a single trajectory. *SIAM Journal on Applied Dynamical Systems*, 13(4):1792–1815, 2014.

[44] Stefan Schäffler. *Generalized Stochastic Processes*. Springer, 2018.

[45] Debarshi Patanjali Ghoshal and Hannah Michalska. Finite interval estimation of lti systems using differential invariance, instrumental variables, and covariance weighting. In *2020 American Control Conference (ACC)*, pages 731–736, 2020.

[46] Weijie Wang and Yanmin Lu. Analysis of the mean absolute error (mae) and the root mean square error (rmse) in assessing rounding model. In *IOP conference series: materials science and engineering*, page 012049. IOP Publishing, 2018.

[47] JH Gove and DY Hollinger. Application of a dual unscented kalman filter for simultaneous state and parameter estimation in problems of surface-atmosphere exchange. *Journal of Geophysical Research: Atmospheres*, 111(D8), 2006.

[48] E Wan and R Van Der Merwe. The unscented kalman filter, ch. 7: Kalman filtering and neural networks. edited by s. haykin, 2001.

[49] Simo Särkkä. Unscented rauch–tung–striebel smoother. *IEEE transactions on automatic control*, 53(3):845–849, 2008.

[50] Simo Särkkä. Continuous-time and continuous–discrete-time unscented rauch–tung–striebel smoothers. *Signal Processing*, 90(1):225–235, 2010.

[51] Rudolph Van Der Merwe and Eric A Wan. The square-root unscented kalman filter for state and parameter-estimation. In *2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221)*, volume 6, pages 3461–3464. IEEE, 2001.

[52] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F-radar and signal processing*, pages 107–113. IET, 1993.

[53] Saneej Balakrishnapillai Chitralekha. *Computational tools for soft sensing and state estimation*. PhD thesis, University of Alberta, 2011.

[54] Dr. Gregory Plett. Simultaneous state and parameter estimation using kalman filters.

[55] Nithilasaravanan Kuppan and Manoj Venkatesan. Python estimation toolkits. `https://github.com/NithilaSaravanan/PETS/`, 2022.

[56] Nachi Gupta. Kalman filtering in the presence of state space equality constraints. In *2007 Chinese Control Conference*, pages 107–113. IEEE, 2007.

[57] Dan Simon and Tien Li Chia. Kalman filtering with state equality constraints. *IEEE transactions on Aerospace and Electronic Systems*, 38(1):128–136, 2002.

[58] Minjea Tahk and Jason L Speyer. Target tracking problems subject to kinematic constraints. *IEEE transactions on automatic control*, 35(3):324–326, 1990.