Kalman Filter Based Sensor Placement For Burgers Equation

by

Stanislav N Zonov

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, Canada, 2019

Abstract

The algorithm used for determining sensor placement in this thesis will be based on the Kalman filter. This filter is very famous and its application are numerous - some examples include aircraft navigation, finance and weather forecasting. It is used to extract an estimate of the true state of a system based on noisy measurements. For linear systems, where the noise satisfies certain assumptions, the Kalman filter minimizes the expected squared error between the filter's estimate and the true state of the system. By varying the sensor location, and therefore the observation matrix, one can further minimize the expected squared error to determine an optimal sensor placement. For linear systems, a regular as well as steady state Kalman filter are used for the sensor placement algorithm. This thesis examines this concept further, for nonlinear systems by using the Extended Kalman filter.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors Professor Kirsten Morris and Professor Marek Stastna for the continuous support of my studies and related research, for patience, motivation, and immense knowledge. I am very grateful for the opportunity to complete my Masters.

I would like to thank all the colleagues on the sixth floor of MC for sharing knowledge and most importantly good humour with me!

Last but not the least, I would like to thank my family and friends for supporting me.

Dedication

I dedicate this thesis to the MC building in which I completed my Masters and my undergrad in and have spent countless hours studying inside. I first met this building in math circles in highschool and thus have known MC for ten years. It has been a long time and I am ready to say goodbye. Goodbye to the massive concrete walls, the dim computer labs, the C&D and the fourth floor classrooms. I will cherish your memories forever.

Table of Contents

List of Figures **viii**

1 Introduction **1**

2 Model **3**

2.1 Burgers . . . 3

2.2 Numerics . . . 4

2.2.1 Modal Methods . . . 5

2.2.2 Finite Difference . . . 8

3 Kalman Filter **13**

3.1 Introduction . . . 13

3.2 Discrete-time Kalman filter . . . 13

3.3 Continuous-time Kalman Filter . . . 20

3.4 Kalman Filter Implementation . . . 27

3.4.1 Update Phase . . . 27

3.4.2 Predict Phase . . . 29

3.4.3 Implementation . . . 32

3.5 Extended Kalman Filter . . . 33

4 Sensor placement introduction **35**

4.1 Cost Function 35

4.1.1 Steady State Kalman Filter 36

4.1.2 Transient Kalman Filter 39

4.2 Verification 42

5 Nonlinear sensor placement **47**

5.1 Modal Discretization 49

5.2 Spatial Discretization 53

6 Conclusion **63**

References **65**

APPENDICES **69**

A QR for Kalman Update phase **70**

B Kalman implementation **73**

B.1 Discrete-Discrete Extended Kalman Filter 73

B.2 Continuous-Discrete Extended Kalman Filter 78

List of Figures

2.1 In left figure, the IMEX is used to solve Burgers equation. In the right panel, the Euler-Maruyama scheme is used. For both figures: $N = 60$, $\nu = 6$, $L = 100$, $\Delta t = 0.002$, $a_1(0) = a_2(0) = a_3(0) = 5$ and $a_4(0) = \ldots = a_{60}(0) = 0$. 8

2.2 In the left panel, the periodic Burgers equation is solved for using scheme (2.38) and plotted for various output times. In the right panel, the eigenvalues of $[X]$ from (2.46) are plotted for various timesteps. All eigenvalues are less than one in absolute value and are therefore in the stability region. The numerical method is thus stable. For both figures: $N = 200$, $L = 100$, $\Delta x = 0.5$, $\Delta t = 0.01$, $\nu = 2$, $u(x, t = 0) = \sin((\pi/L)x) + \sin((2\pi/L)x) + 0.5$. 12

3.1 The graph shows four curves useful in understanding and assessing the performance of the dt-KF. Comparing the ideal position (without any process noise) to actual position shows the effect of Q. Comparing measured position to actual position shows the effect of R. Finally, comparing estimated position to actual position demonstrates the Kalman filter's performance. For examples where initial estimate does not match the state or other interesting examples please refer to [27]. 20

4.1 Trace of the steady state Kalman covariance is computed for all x_0. The location that is the optimal sensor placement is $x = 1.55$ as it minimizes the cost function J_{ss} in (4.4) 38

4.2 The trace of the Kalman filter covariance is plotted for two different sensor locations. Around $t = 6$ the trace at $x = 1.6$, which is initially larger then that at $x = 0.8$, stops being larger than that at $x = 0.85$. 41

4.3 The log of the trace of the Kalman covariance for various sensor locations for the same candidate set considered in example (4.1). As time increases, the trace decreases. 41

4.4 The log of the square of RMSE (4.21) is plotted for various sensor locations. The expression (4.20) approximates (4.19) well. 44

4.5 The incorrect case initializes estimate of a_2 as zero. Since the sensor is located at the node of second mode, it cannot pick up its amplitude. Therefore, the incorrect estimate remains unchanged and only converges to the real state once the real state has diffused to zero. The wrong guess estimate's a_2 value is on on top of the Time axis. 45

4.6 The finite sample approximation of Kalman trace is no longer close to the real trace when the initial guess is incorrect. 46

5.1 From left to right: initial condition u_1 and states at later times and u_2 and states at later times as defined in (5.8-5.9). 49

5.2 The trace of the Kalman trace is computed over time 0 to 1 for various sensor locations on the x axis. The log of the trace is plotted for initial condition u_1. 50

5.3 The trace of the Kalman trace is computed over time 0 to 1 for various sensor locations on the x axis. The log of the trace is plotted for initial condition u_2. 51

5.4 The log of $J^N_{[t_s,t_f]}$ is plotted for both initial conditions. The cost function is minimized at different locations: for u_1 it is at $x = 2.5$ with a cost of 3.3309×10^{-6} while for u_2 it is at $x = 2.9$ with a cost of 5.6026×10^{-6}. . . 52

5.5 Left panel: The initial condition u_0 from (5.16) of the periodic domain and average trajectory is plotted. Right panel: A space time plot of the evolution of the initial condition (i.e. the trajectory of the system). The colour and associated colour bar show the magnitude of the state at a given time - the brighter the colour the higher the magnitude. By the end of simulation time (T $=$ 200) the waveform has mostly diffused out. 54

5.6 The average relative error between noisy measurement and true state value at each location is plotted and normalized with respect to the largest relative error at a given time. The noise is zero mean with constant variance 0.1^2. The higher the ratio, the smaller the u_j^n at that location. The initial condition is (5.16). 55

5.7 On the left is the average state trajectory with shaded region showing area where the state is relatively high compared to other locations at the same time. On the right, the normalized relative error between noisy sensor measurement and true state from Figure 5.6. The locations where the state has a high value is where the relative error is the lowest - this is not a surprise and is expected. The initial condition is (5.16). 55

5.8 Kalman trace is computed over time 0 to 200 for various sensor locations on the x axis. The color represents the magnitude of the trace at a specific time and location. 56

5.9 The true state is plotted along with estimate with sensor at $x_0 = 31$ and estimate with sensor at $x_0 = 57$, all at $t = 71.67$. Note: small gap for x in range $[20, 50]$ between estimate with sensor at $x = 57$ and true state signifying that estimator with sensor location at $x_0 = 31$ is performing better. The initial condition is (5.16). 57

5.10 The diagonal entries of the Kalman covariance are plotted for two different sensor locations at $t = 71.67$. The i^{th} entry corresponds to the $x = i$ grid point. The true state is expected to be much better approximated by the sensor at $x_0 = 31$ in the domain $[20, 50]$. The initial condition is (5.16). . . 58

5.11 The squared RMSE for all 150 simulation runs. The plot is very similar in shape to the trace plot in Figure 5.8. However, the scale is not the same. . 59

5.12 All possible $J^1_{[0,200]}50$ scores given the sensor candidates. The location that satisfies (5.6) is at $x_0 = 5$ with a score of 0.013538. 60

5.13 Relative error log plot between average Kalman trace using $N = 150$ as reference solution and varying N. The relative error stays low throughout. 61

5.14 For state space size M and 20000 measurements the average computation time in seconds is shown for Matlab implementation for the EKF. For the investigation the sample size is $N = 150$ with 50 candidate locations meaning that for $M = 50$ the computation time on a single core is approximately 1.5 days while for $M = 250$ is 9.5 days. If sensor count is set to 2 (choosing an optimal pair of sensors) then $\binom{50}{2}$ total candidate pairs need to be investigated instead of 50 - a significant computational cost. 62

Chapter 1

Introduction

When one is sick, thermometers are often used as part of medical treatment. The thermometer is an example of a sensor and the associated measurement is the body temperature. Measurements are used to obtain more information about the system and in the example considered, they are used to diagnose the severity of fever. The accuracy provided by the temperature measurements are sufficient for most health applications but as all measurements, they fall victim to noise and uncertainty. For older, mercury based thermometers, an example of noisy or biased measurements are those that come after forgetting to shake the thermometer beforehand. The problem of sensor placement is not as abstract at it sounds and in fact is a very common and important one. For thermometers, the sensor placement problem is resolved by a set of medically recommended locations. Some locations are better at providing more useful information than others.

For studying sensor placement, a system of interest needs to be selected first. Intricate biological systems such as the human body will not be chosen and instead a more simple model will be that is represented by a well known, simple mathematical equation. The purpose of simplification is to allow for a rigorous investigation to then provide groundwork for future investigations for more specific, complex systems.

The algorithm used for determining sensor placement in this thesis will be based on the Kalman filter. This filter is very famous and its application are numerous - some examples include aircraft navigation, finance and weather forecasting. It is used to extract an estimate of the true state of a system based on noisy measurements. For linear systems, where the noise satisfies certain assumptions, the Kalman filter minimizes the expected squared error between the filter's estimate and the true state of the system. The error based minimization property with the combination of varying sensor position allows for an intuitive brute force type algorithm that by varying the sensor location further minimizes the expected squared error to produce an optimal sensor placement. This thesis examines this concept further, for nonlinear systems.

In chapter 2 the system of interest, namely Burgers equation, is introduced along with ways to numerically approximate it. Next, in chapter 3, the Kalman filter is introduced in detail along with its implementation. Chapter 4 introduces the basics of the sensor placement algorithms and nonlinear results are discussed in 5. Finally in chapter 6 conclusions are made and possible avenues of future work reflected upon.

Chapter 2

Model

The system considered for sensor placement will be Burgers equation. It is the simplest equation combining both quadratic nonlinearity (i.e. nonlinear advection) and diffusion. The equation form is similar to those that govern fluid flow and will be used as bridgehead in investigating sensor placement for nonlinear fluid equations. A great introduction to the equation and its properties can be found in [40] while numerics are covered in [2],[30] and [31]. The basic physical intuition behind the equation will be introduced first followed by a section on numerical methods for this equation.

2.1 Burgers

Burgers equation is

$$u_t + uu_x = \nu u_{xx}. \tag{2.1}$$

In models related to waves there is some distribution of material or some state of medium of interest [40]. Consider $\rho(x, t)$ as the density per unit length and $q(x, t)$, the flux of material per unit time. Assuming conservation of material, one can conclude that over some boundary defined by x_2 and x_1, the rate of change of material within the region $(x_1 > x > x_2)$ is balanced by the flow across x_1 and x_2:

$$\frac{d}{dt}\int_{x_2}^{x_1} \rho(x, t)dx = q(x_2, t) - q(x_1, t). \tag{2.2}$$

When the limit $x_1 \to x_2$ is taken then one obtains the conservation equation

$$\frac{\partial \rho}{\partial t} + \frac{\partial q}{\partial x} = 0. \tag{2.3}$$

In many cases there can be a relationship between flux and density such that $q = Q(\rho)$. An example of this is when more dense regions in traffic flow have lower flux. Substituting the relation into the conservation equation (2.3) yields

$$\rho_t + u(\rho)\rho_x = 0 \tag{2.4}$$

where $u(\rho) = Q'(\rho)$ is the propagation velocity of ρ. In the case of $u(\rho)$ constant then (2.4) reduces to the advection equation. In some cases, it would be better to assume further that q is not just a function of ρ but of its gradient ρ_x and that Q is quadratic in ρ:

$$q = Q(\rho) - \nu\rho_x. \tag{2.5}$$

In the case of traffic flow, drivers reduce their speed to account for an increasing density of cars ahead of them. This in turn lowers the net flux in a region of increasing density. Substituting (2.5) into the conservation equation (2.3) simplifies as follows

$$\rho_t + (Q(\rho) - \nu\rho_x)_x = 0 \tag{2.6}$$

$$\rho_t + Q'(\rho) = \nu\rho_{xx} \tag{2.7}$$

$$\rho_t + u(\rho)\rho_x = \nu\rho_{xx}. \tag{2.8}$$

Multiplying (2.8) by $u'(\rho)$:

$$u'(\rho)(\rho_t + u(\rho)\rho_x) = u'(\rho)(\nu\rho_{xx}) \tag{2.9}$$

$$(u'(\rho)\rho_t) + u(\rho)\left(u'(\rho)\rho_x\right) = u'(\rho)(\nu\rho_{xx}) \tag{2.10}$$

$$u_t + uu_x = \nu u'(\rho)\rho_{xx} \tag{2.11}$$

$$u_t + uu_x = \nu u_{xx} - \nu u''(\rho)\rho_x^2. \tag{2.12}$$

Since Q is quadratic in ρ, then (2.12) reduces to

$$u_t + uu_x = \nu u_{xx} \tag{2.13}$$

which is Burgers equation. In this equation uu_x term leads to steepening and breaking, while the νu_{xx} term represents diffusion.

2.2 Numerics

Two types of discretizations will be considered. The first will convert Burgers into a system of ordinary differential equations (ODEs) by assuming u can be expressed as a truncated series of sines (or modes). The ODEs will be solved using an Euler type scheme. The second discretization will be an upwind scheme, the implementation of which will involve finite differences.

2.2.1 Modal Methods

In the modal scheme, Burgers equation will be approximated by a system of ODEs. This will be done by setting u to be a truncated sine series with N total modes:

$$u(x,t) = \sum_{j=1}^{N} a_j(t) \sin\left(\left(\frac{j\pi}{L}\right) x\right) \tag{2.14}$$

where $a_j(t)$ is the time dependent coefficient of the j^{th} mode. The approach is not novel and for instance has been used to discretized the shallow water equation in [3]. The boundary conditions are chosen to be Dirichlet

$$u(0,t) = u(L,t) = 0 \tag{2.15}$$

where L is domain length. The boundary conditions make the sine basis approximation natural. Substituting (2.14) into Burgers equation, multiplying by some $\sin\left(\left(\frac{k\pi}{L}\right)x\right)$ and integrating over the domain yields

$$\begin{aligned} \sum_{j=1}^{N} a_j'(t) \int_0^L \sin\left(\left(\frac{j\pi}{L}\right) x\right) \sin\left(\left(\frac{k\pi}{L}\right) x\right) dx + \\ \int_0^L \sin\left(\left(\frac{k\pi}{L}\right) x\right) \left(\left[\sum_{j=1}^{N} a_j(t) \sin\left(\left(\frac{j\pi}{L}\right) x\right)\right] \left[\sum_{j=1}^{N} \left(\frac{j\pi}{L}\right) a_j(t) \cos\left(\left(\frac{j\pi}{L}\right) x\right)\right]\right) dx = \\ -\nu \sum_{j=1}^{N} \left(\frac{j\pi}{L}\right)^2 a_j(t) \int_0^1 \sin\left(\left(\frac{k\pi}{L}\right) x\right) \sin\left(\left(\frac{j\pi}{L}\right) x\right) dx. \end{aligned} \tag{2.16}$$

The orthogonal nature of sine

$$\int_0^L \sin\left(\left(\frac{j\pi}{L}\right) x\right) \sin\left(\left(\frac{k\pi}{L}\right) x\right) dx = \begin{cases} \frac{L}{2},\ k = j \\ 0,\ \text{otherwise} \end{cases}$$

implies that equation (2.16) is reduced to

$$\begin{aligned}\left(\frac{L}{2}\right) a_k'(t) = \\ -\int_0^L \sin\left(\left(\frac{k\pi}{L}\right)x\right)\left(\left[\sum_{j=1}^{N} a_j(t)\sin\left(\left(\frac{j\pi}{L}\right)x\right)\right]\left[\sum_{j=1}^{N}(j\pi)\,a_j(t)\cos\left(\left(\frac{j\pi}{L}\right)x\right)\right]\right)dx \\ -\nu\left(\frac{k\pi}{L}\right)^2\left(\frac{L}{2}\right)a_k(t).\end{aligned} \tag{2.17}$$

Equation (2.17) is an ODE for the k^{th} coefficient of the k^{th} sine mode. By varying $k = 1...N$, N total such ODEs are derived which convert Burgers' PDE to a system of nonlinear ODEs. The simplfication of the nonlinear sum term in the system of ODEs involves evaluating the following integral

$$\int_0^L \sin\left(\left(\frac{k\pi}{L}\right)x\right)\sin\left(\left(\frac{j\pi}{L}\right)x\right)\cos\left(\left(\frac{m\pi}{L}\right)x\right)dx. \tag{2.18}$$

The integral can be evaluated with the use of the identities

$$\begin{cases}\sin(u)\sin(v) = \frac{1}{2}\left(\cos(u-v)-\cos(u+v)\right)\\ \cos(u)\cos(v) = \frac{1}{2}\left(\cos(u-v)+\cos(u+v)\right)\end{cases} \tag{2.19}$$

which simplify (2.18) to

$$\begin{aligned}\frac{1}{4}\int_0^L \cos\left(\left(\frac{(k-j-m)\pi}{L}\right)x\right)dx + \frac{1}{4}\int_0^L \cos\left(\left(\frac{(k-j+m)\pi}{L}\right)x\right)dx \\ -\frac{1}{4}\int_0^L \cos\left(\left(\frac{(k+j-m)\pi}{L}\right)x\right)dx - \frac{1}{4}\int_0^L \cos\left(\left(\frac{(k+j+m)\pi}{L}\right)x\right)dx.\end{aligned} \tag{2.20}$$

The intergral bounds further simplify (2.20) to

$$\begin{cases}\frac{L}{4},\text{ if } k-j-m=0\\ \frac{L}{4},\text{ if } k-j+m=0\\ -\frac{L}{4},\text{ if } k+j-m=0\\ -\frac{L}{4},\text{ if } k+j+m=0\\ 0,\text{ otherwise}\end{cases}. \tag{2.21}$$

Hence (2.17)'s nonlinear sum term can be written in terms of a matrix $A_k \in \mathbb{R}^{N\text{x}N}$ such that

$$[a_1(t), \ldots, a_N(t)]\, A_k\, [a_1(t), \ldots, a_N(t)]^T . \tag{2.22}$$

Referring to (2.21), A_k is defined as

$$A_k(j,m) = \left(\frac{m\pi}{L}\right) \begin{cases} \frac{L}{4}, \text{ if } k-j-m=0 \\ \frac{L}{4}, \text{ if } k-j+m=0 \\ -\frac{L}{4}, \text{ if } k+j-m=0 \\ -\frac{L}{4}, \text{ if } k+j+m=0 \\ 0, \text{ otherwise} \end{cases} . \tag{2.23}$$

Therefore, the k^{th} mode's coefficient $a_k(t)$ is governed by the nonlinear ODE

$$a'_k(t) = -\,[a_1(t), \ldots, a_N(t)]\, A_k\, [a_1(t), \ldots, a_N(t)]^T - \nu \left(\frac{k\pi}{L}\right)^2 a_k(t). \tag{2.24}$$

One can combine (2.24) for all $k = 1...N$ such that

$$\frac{d}{dt}\left(\begin{bmatrix} a_1(t) \\ \vdots \\ a_N(t) \end{bmatrix}\right) = \frac{d}{dt}\left(\vec{a}\right) = f(\vec{a}). \tag{2.25}$$

One fairly general approach to solving (2.25) is to use an implicit-explicit method (IMEX) [29]. This can be done by splitting f in (2.25) into two components

$$\frac{d}{dt}\left(\vec{a}\right) = g(\vec{a}) + h(\vec{a}) \tag{2.26}$$

where g is non-stiff and h is the stiff component. Stiffness refers to the problem of computing a numerical solution that is smooth and slowly varying but that requires a very small timestep[29]. The stiff component is the diffusive term

$$-\,\nu \left(\frac{k\pi}{L}\right)^2 a_k(t) \tag{2.27}$$

as the largest $k = N$ will restrict the time step for all other k. Implicit Euler's can be used to deal with the stiff diffusive term while explicit Euler's can be used for the nonlinear term. The discretization is

$$a_k^{n+1} = \left(a_k^n - (\Delta t)\, [a_1^n, \ldots, a_N^n]\, A_k\, [a_1^n, \ldots, a_N^n(t)]^T\right) - (\Delta t)\, \nu \left(\frac{k\pi}{L}\right)^2 a_k^{n+1} \tag{2.28}$$

where $a_k^n = a_k(t_n) = a_k\left(n(\Delta t)\right)$. For this thesis, process noise will be needed to be added, changing (2.25) to

$$\frac{d}{dt}(\vec{a}) = f(\vec{a}) + \vec{n}(t) \tag{2.29}$$

where $\vec{n}(t) \in \mathbb{R}^N$ is continuous white noise. The equation (2.29) is a stochastic ordinary differential equation (SDE) and will be solved using the Euler-Maruyama scheme:

$$\vec{a}_k^{n+1} = \vec{a}_k^n + \Delta t f(\vec{a}_k^n) + \sqrt{\Delta t}\vec{q} \tag{2.30}$$

where $\vec{q}$ is sampled Gaussian noise with zero mean and covariance Q. More details on the noise and relation between $\vec{n}(t)$ and $\vec{q}$ will be introduced in chapter 3. The Euler-Maruyama scheme is a basic time-stepping scheme with relatively poor accuracy, and is used in many applications for solving SDEs. More sophisticated methods are described in [33].

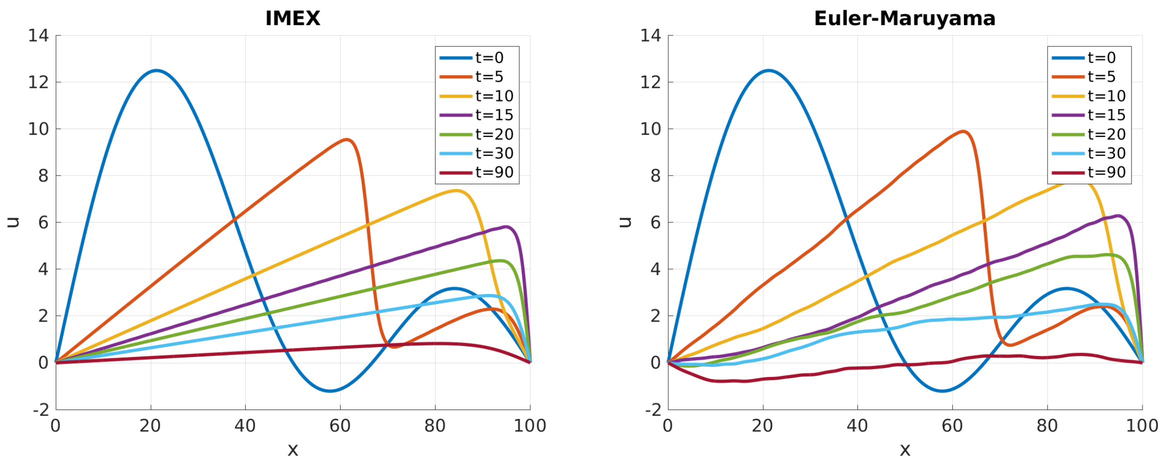

Figure 2.1: In left figure, the IMEX is used to solve Burgers equation. In the right panel, the Euler-Maruyama scheme is used. For both figures: $N = 60$, $\nu = 6$, $L = 100$, $\Delta t = 0.002$, $a_1(0) = a_2(0) = a_3(0) = 5$ and $a_4(0) = \ldots = a_{60}(0) = 0$.

2.2.2 Finite Difference

Consider the linearized Burgers equation where u is replaced with constant c:

$$u_t + cu_x = \nu u_{xx}. \tag{2.31}$$

A potential discretization [2] is the Forward-Time Central Space (FTCS) method that transforms (2.31) into

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}=\nu\frac{u_{j+1}^n-2u_j^n+u_{j-1}^n}{(\Delta x)^2}, \tag{2.32}$$

where u_j^n approximates $u(x_j, t_n)$. This is a first-order, explicit scheme with truncation error $O[\Delta t, (\Delta x)^2]$. The scheme has been used in sensor placement investigations in [21] with the timestep replaced with RK4. Using Von Neumann analysis, the stability restrictions leads to

$$\mu^2 \leq 2r \tag{2.33}$$

$$r \leq \frac{1}{2}, \tag{2.34}$$

where $r = \nu\Delta t/(\Delta x)^2$ and $\mu = c\Delta t/\Delta x$. The mesh Reynolds number [2] is defined as

$$\mathrm{Re}_{\Delta x} = c\frac{\Delta x}{\nu} \tag{2.35}$$

which in combination with restrictions (2.33 - 2.34) gives

$$2\mu \leq \mathrm{Re}_{\Delta x} \leq \frac{2}{\mu}. \tag{2.36}$$

An important issue involving Burgers equation and finite-difference schemes like the centered scheme above is that they exhibit unphysical oscillations. The FTCS will lead to oscillations [2] if

$$2 \leq \mathrm{Re}_{\Delta x} \leq \frac{2}{\mu} \tag{2.37}$$

with $\mathrm{Re}_{\Delta x}$ greater than $2/\mu$ the scheme will lead the solution to blow up [2]. The oscillation can be removed if the second-order central difference discretization for the advection term is replaced with a first-order one that acknowledges the direction that information propagates:

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_j^n-u_{j-1}^n}{2\Delta x}=\nu\frac{u_{j+1}^n-2u_j^n+u_{j-1}^n}{(\Delta x)^2}. \tag{2.38}$$

This change is known as upwinding. Provided that $c > 0$ and assuming that $\nu = 0$ then (2.31) represents information propagating in the positive or rightwards direction. The points chosen to discretize the advection term are in the upstream or equivalently in the opposite direction of the wave propagation. This scheme eliminates oscillation but introduces extra dissipation, especially if $\mathrm{Re}_{\Delta x} > 2$. The scheme (2.38) is extended to the nonlinear case as follows

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + u_j^n \frac{u_j^n - u_{j-1}^n}{\Delta x} = \nu \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}. \tag{2.39}$$

In the inviscid case ($\nu = 0$), the scheme is adequate for smooth solutions but "will not, in general, converge to a discontinuous weak solution of Burgers' equation as the grid is refined" [31]. For the purpose of this thesis shocks are never an issue so the convergence is not a major issue. In fact, for the inviscid case, the scheme (2.39) is very similar to the finite volume conservative method [30]

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(\frac{1}{2}\left(u_j^n\right)^2 - \frac{1}{2}\left(u_{j-1}^n\right)^2\right), \tag{2.40}$$

where u_j^n now represents the average value of u over an interval $[x_j, x_{j+1}]$:

$$u_j^n = \frac{1}{\Delta x}\int_{x_j}^{x_{j+1}} u\left(x, t_n\right) dx. \tag{2.41}$$

The descriptor “conservative " applies to the way the scheme is derived. Conservation of u can be expressed as

$$\frac{1}{\Delta x}\int_{x_j}^{x_{j+1}} u(x, t_{n+1})dx = \frac{1}{\Delta x}\int_{x_j}^{x_{j+1}} u(x, t_n)dx - \frac{1}{\Delta x}\left[\int_{t_n}^{t_{n+1}} f(u(x_j, t))dt - \int_{t_n}^{t_{n+1}} f(u(x_{j+1}, t))dt\right]. \tag{2.42}$$

The equality (2.42) implies that the average value of u at t_{n+1} is a function of the average value at t_n and the cumulative flux f through the boundaries at x_j and x_{j+1} over the time interval $[t_n, t_{n+1}]$ - the value of u is conserved. Conservative schemes mimic (2.42), and take the form

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(F(u_j^n, u_{j+1}^n) - F(u_{j-1}^n, u_j^n)\right). \tag{2.43}$$

Despite numerical similarities for smooth solutions, the finite volume (2.40) and finite difference (2.38) schemes are not algebraically equivalent. However, an algebraic equivalent can be found for the inviscid case, (2.38) as

$$u_j^{n+1} = \frac{1}{\Delta x}\left[\left(\Delta x - u_j^n \Delta t\right) u_j^n + \left(\Delta t u_j^n\right) u_{j-1}^n\right] \tag{2.44}$$

that is Burgers specific version of the Courant-Isaacson-Rees method [31] with $u_j^n > 0$ for all j. This method is non conservative, but treats u_j^n as an average volume of u over the interval $[x_j, x_{j+1}]$ in spirit of finite volume schemes. The propagation velocity of u_j^n is c in the linear case and u_j^n in the nonlinear. The average value u_{j-1}^n on the interval $[x_{j-1}, x_j]$ will be assumed to be propagating at speed u_j^n into the next interval. The result is that the value of u_j^{n+1} will be u_{j-1}^n on the interval $\left[x_j, x_j + \Delta t u_j^n\right]$. By similar logic, the value on interval $\left[x_j + \Delta t u_j^n, x_{j+1}\right]$ will be u_j^n. Combining the propagating u in both intervals and averaging one obtains the scheme (2.44). Through basic algebraic manipulation (2.44) can be shown to be identical to the inviscid version, (2.38). By setting $F(U,W) = \left(\nu/\left(\Delta x^2\right)\right)(-U+W)$ in (2.43), diffusion can also be accounted in equivalence between (2.44) and (2.38).

The boundary conditions will be set to be periodic

$$u(x=0,t_n) = u_0^n = u_N^n = u(x=L,t_n), \tag{2.45}$$

where L is domain length. With the combination of (2.38) and (2.45), the scheme can be expressed as

$$\begin{bmatrix} u_0^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \end{bmatrix} = \vec{u}^{n+1} = [X]\vec{u}^n, \tag{2.46}$$

where $[X]$ is a square matrix and a function of $\vec{u}^n$. The eigenvalues of the matrix can be investigated to verify for stability as seen in Figure 2.2.

It is worth making a final comment regarding stability. It is well known [31] that upwinding for an advection term (linear or nonlinear) is equivalent to adding a diffusion term with a grid dependent diffusivity. This generally leads to an increase in stability of the numerical scheme, and this is consistent with what we observed in this thesis.

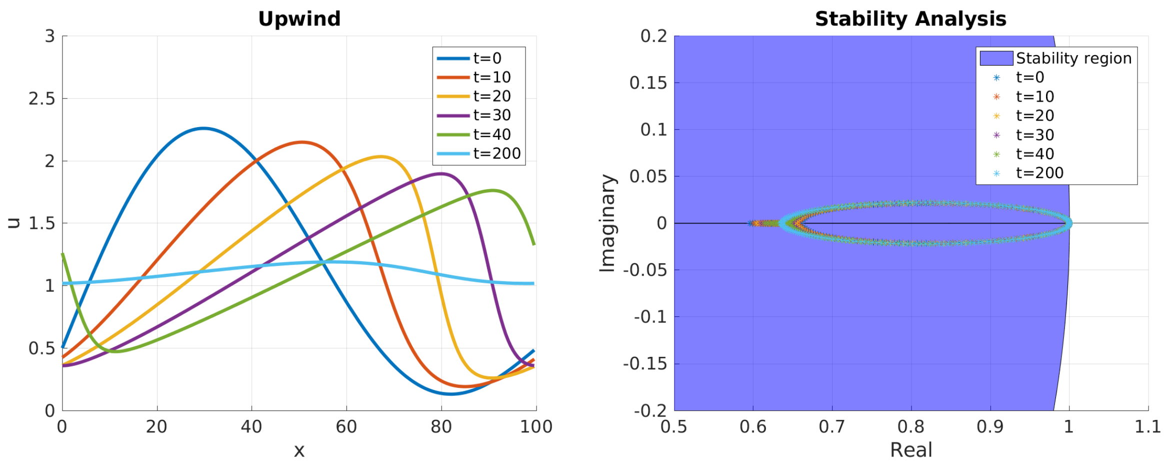

Figure 2.2: In the left panel, the periodic Burgers equation is solved for using scheme (2.38) and plotted for various output times. In the right panel, the eigenvalues of $[X]$ from (2.46) are plotted for various timesteps. All eigenvalues are less than one in absolute value and are therefore in the stability region. The numerical method is thus stable. For both figures: $N = 200$, $L = 100$, $\Delta x = 0.5$, $\Delta t = 0.01$, $\nu = 2$, $u(x, t = 0) = \sin((\pi/L)x) + \sin((2\pi/L)x) + 0.5$.

Chapter 3

Kalman Filter

3.1 Introduction

The Kalman filter, originally introduced in the early 1960s [19] [20], is an algorithm that extracts a signal from noisy measurements based on an assumed model. The filter has a wide range of applications including economics and spacecraft navigation [34]. In this thesis, the Kalman filter will be a vital component of the algorithms employed for sensor placement.

3.2 Discrete-time Kalman filter

In this section, the discrete-time Kalman filter (**dt-KF**) will be derived based on [36]. The derivation will start with the introduction of least squares estimation.
Assume $\vec{x} \in \mathbb{R}^n$ is a constant vector and $\vec{y} \in \mathbb{R}^k$ is a k element measurement vector. Furthermore, assume that $\vec{x}$ is an unknown state that one is trying to estimate using the measurements. Let each element of the measurement vector be a linear combination of the entries in $\vec{x}$:

$$y_1 = C_{11}x_1 + \ldots + C_{1n}x_n + v_1 = C_1\vec{x} + v_1 \tag{3.1}$$

$$\vdots \tag{3.2}$$

$$y_k = C_{k1}x_1 + \ldots + C_{kn}x_n + v_k = C_k\vec{x} + v_k, \tag{3.3}$$

where v_i is added, mean zero, measurement noise. Measurements (3.1 - 3.3) can be abbreviated as

$$\vec{y} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_k \end{bmatrix} \vec{x} + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix} = C\vec{x} + \vec{v}. \tag{3.4}$$

The matrix C, known as the observation matrix, extracts entries from the state and converts them into k measurements. The estimate of $\vec{x}$ is defined to be $\vec{\hat{x}}$ so that $\vec{r}$ is the residual, written explicitly as,

$$\vec{r} = \vec{y} - C\vec{\hat{x}}. \tag{3.5}$$

The residual can be interpreted as the error between the measurements and noiseless measurements of $\vec{x}$ and ideally would be zero. A choice for estimate $\vec{\hat{x}}$ is one that minimizes the scalar cost J - the squared norm of the residual

$$J = r^T r = \left(\vec{y} - C\vec{\hat{x}}\right)^T \left(\vec{y} - C\vec{\hat{x}}\right), \tag{3.6}$$

where superscript T denotes the transpose. It can be shown that the estimate that minimizes J is obtained through the method of least squares where in the case of a linear system the estimate is given by $\vec{\hat{x}} = (C^T C)^{-1} C^T \vec{y}$ [36]. The disadvantage of least squares estimation is that it requires all $k+1$ measurements to recompute the estimate when a new measurement is recorded. For large k, the new measurement presents a computational issue. The issue can be addressed by using a recursive least squares estimator. Consider an already computed estimate $\vec{\hat{x}}_k$ after k measurements and that an additional measurement is obtained

$$y_{k+1} = C_{k+1}\vec{x} + v_{k+1}. \tag{3.7}$$

Via the recursive least squares estimator, the current estimate $\vec{\hat{x}}_{k+1}$ is given by

$$\vec{\hat{x}}_{k+1} = \vec{\hat{x}}_k + K_{k+1}\left(y_{k+1} - C_{k+1}\vec{\hat{x}}_k\right). \tag{3.8}$$

To compute (3.8) one only needs the previous estimate $\vec{\hat{x}}_k$ and the current measurement y_{k+1}. Before specifying how K_{k+1} in (3.8) is derived, first consider the expected mean of the estimation error and substitute in (3.7) and (3.8):

$$\begin{aligned}
\mathrm{E}\left[\left(\vec{x}-\hat{\vec{x}}_{k+1}\right)\right] &= \mathrm{E}\left[\vec{x}-\left(\hat{\vec{x}}_k+K_{k+1}\left(y_{k+1}-C_{k+1}\hat{\vec{x}}_k\right)\right)\right] && (3.9)\\
&= \mathrm{E}\left[\vec{x}-\left(\hat{\vec{x}}_k+K_{k+1}\left(\left(C_{k+1}\vec{x}+v_{k+1}\right)-C_{k+1}\hat{\vec{x}}_k\right)\right)\right] && (3.10)\\
&= \mathrm{E}\left[\left(\vec{x}-\hat{\vec{x}}_k\right)-K_{k+1}C_{k+1}\left(\vec{x}-\hat{\vec{x}}_k\right)-K_{k+1}v_{k+1}\right] && (3.11)\\
&= \left(I-K_{k+1}C_{k+1}\right)\mathrm{E}\left[\left(\vec{x}-\hat{\vec{x}}_{k-1}\right)\right]-K_{k+1}\mathrm{E}\left[v_{k+1}\right]. && (3.12)
\end{aligned}$$

Assume that the noise is zero mean

$$\mathrm{E}\left[v_i\right]=0, \forall i \in \mathbb{N}, i>0$$

with variance R_k. In addition, due to the estimator's recursive nature $\hat{\vec{x}}_0$ must be defined despite the absence of a measurement at $k=0$. If one chooses $\hat{\vec{x}}_0$ such that

$$\begin{aligned}
\mathrm{E}\left[\left(\hat{\vec{x}}_0-\vec{x}\right)\right] &= 0 && (3.13)\\
\hat{\vec{x}}_0 &= \mathrm{E}\left[\left(\vec{x}\right)\right] && (3.14)
\end{aligned}$$

then by (3.12)

$$\begin{aligned}
\mathrm{E}\left[\left(\hat{\vec{x}}_k-\vec{x}\right)\right] &= 0 && (3.15)\\
\hat{\vec{x}}_k &= \mathrm{E}\left[\left(\vec{x}\right)\right] && (3.16)
\end{aligned}$$

holds. In other words, the mean assumption of the noise and initial estimate and (3.12) indicate that the expected value of $\hat{\vec{x}}_k$ is equal to $\vec{x}$. This makes the estimator *unbiased*.

The current estimate is updated based on balancing between previous estimate and current measurement using the matrix K_{k+1}. The matrix, commonly referred to as the gain, is specified by choosing an optimality criterion that the estimate would need to satisfy. The previous cost J (3.6) can no longer be used as it requires the use of all measurements. Consider the new scalar cost J_{k+1} - the sum of the variances of the error entries at the $(k+1)^{\text{th}}$ measurement:

$$\begin{aligned}
J_{k+1} &= \mathrm{E}\left[\left(\vec{x}-\hat{\vec{x}}_{k+1}\right)^T\left(\vec{x}-\hat{\vec{x}}_{k+1}\right)\right] && (3.17)\\
&= \operatorname{Trace}\left(\mathrm{E}\left[\left(\vec{x}-\hat{\vec{x}}_{k+1}\right)\left(\vec{x}-\hat{\vec{x}}_{k+1}\right)^T\right]\right) && (3.18)\\
&= \operatorname{Trace}\left(\hat{P}_{k+1}\right), && (3.19)
\end{aligned}$$

where $\hat{P}_{k+1}$ is the error covariance matrix and Trace is the operator summing the diagonal entries of a square matrix. The estimate that minimizes (3.17) is computed as

$$K_k = \hat{P}_{k-1} C_k^T \left(C_k \hat{P}_{k-1} C_k^T + R_k \right)^{-1} \tag{3.20}$$

$$\vec{\hat{x}}_k = \vec{\hat{x}}_{k-1} + K_k \left(\vec{y}_k - C_k \vec{\hat{x}}_{k-1} \right) \tag{3.21}$$

$$\hat{P}_k = \left(I - K_k C_k \right) \hat{P}_{k-1} \tag{3.22}$$

with $\vec{\hat{x}}_0 = \mathrm{E}\left[\vec{x}\right]$ as in (3.13) and initial estimate covariance $P_0 = \mathrm{E}\left[\left(\vec{x} - \vec{\hat{x}}_0\right)\left(\vec{x} - \vec{\hat{x}}_0\right)^T\right]$[36].

The unknown state $\vec{x}$ defined as a constant vector is a strong restriction which is addressed by the Kalman filter. Consider that $\vec{x}$ now evolves based on the following transition equation

$$\vec{x}_k = A\vec{x}_{k-1} + \vec{w}_k, \tag{3.23}$$

where k represents the order of the measurement. The k^{th} measurement

$$\vec{y}_k = C\vec{x}_k + \vec{v}_k. \tag{3.24}$$

is taken at time t_k, a constant Δt seconds after the previous $(k-1)^{\mathrm{th}}$ measurement. The system matrix A is a constant $n \times n$ matrix, C is still the observation matrix, $\vec{v}_k$ and $\vec{w}_k$ are *sensor* and *process* noise respectively. The system matrices A,Q,R,C can be changed to be time step dependent in the Kalman filter formulation, but this will be avoided here for the purposes of clarity. The noise vectors have the following properties:

$$\vec{w}_k \sim \mathcal{N}(\vec{0}, Q) \tag{3.25}$$

$$\vec{v}_k \sim \mathcal{N}(\vec{0}, R) \tag{3.26}$$

$$\mathrm{E}\left[\vec{w}_k \vec{w}_j^T\right] = Q\delta_{k,j} \tag{3.27}$$

$$\mathrm{E}\left[\vec{v}_k \vec{v}_j^T\right] = R\delta_{k,j} \tag{3.28}$$

$$\mathrm{E}\left[\vec{v}_k \vec{w}_j^T\right] = 0. \tag{3.29}$$

The notation $\vec{x} \sim \mathcal{N}(\vec{y}, Z)$ means that $\vec{x}$ is Gaussian with mean $\vec{y}$ and covariance Z. Properties (3.25) and (3.26) define the process and sensor noise as zero mean and Gaussian with covariance Q and R respectively. The Kronecker delta $\delta_{k,j}$ in (3.27) and (3.28) restricts the noise to be time independent while (3.29) implies that process and sensor noise are independent.

In the recursive least squares estimator incoming measurements modified the covariance matrix P_k as seen in (3.20). The Kalman filter will also modify the covariance matrix based on the transition equation (3.23). Consider the mean of the state at t_k:

$$\begin{aligned} \mathrm{E}\left[\vec{x}_k\right] &= \bar{\vec{x}}_k \qquad (3.30)\\ &= A\mathrm{E}(\vec{x}_{k-1}) + E\left(\vec{w}_k\right) \qquad (3.31)\\ &= A\bar{\vec{x}}_{k-1}. \qquad (3.32) \end{aligned}$$

Combining (3.30) with (3.23) and (3.27) it can then be shown that

$$P_k = AP_{k-1}A^T + Q. \qquad (3.33)$$

The effect of the transition equation in (3.33) combined with the measurement update of the recursive least squares estimator in (3.20-3.22) to derive the dt-KF. Let $\hat{\vec{x}}_k^-$ be the *a priori* estimate based on all measurements before time t_k:

$$\hat{\vec{x}}_k^- = \mathrm{E}\left[\vec{x}_k | \vec{y}_1, \vec{y}_2, \ldots, \vec{y}_{k-1}\right], \qquad (3.34)$$

where the right hand side term is a conditional probability - the expected value of $\vec{x}_k$ given the measurements. Let $\hat{\vec{x}}_k^+$ be the *a posteriori* estimate based on all measurements up to and including time t_k:

$$\hat{\vec{x}}_k^+ = \mathrm{E}\left[\vec{x}_k | \vec{y}_1, \vec{y}_2, \ldots, \vec{y}_k\right] \qquad (3.35)$$

Both $\hat{\vec{x}}_k^-$ and $\hat{\vec{x}}_k^+$ are estimates of $\vec{x}_k$. Since $\hat{\vec{x}}_k^+$ includes additional information, the intuition is that it will be a better estimate. With these two estimates there are associated covariance matrices:

$$\begin{aligned} \hat{P}_k^- &= \mathrm{E}\left[\left(\vec{x}_k - \hat{\vec{x}}_k^-\right)\left(\vec{x}_k - \hat{\vec{x}}_k^-\right)^T\right] \qquad (3.36)\\ \hat{P}_k^+ &= \mathrm{E}\left[\left(\vec{x}_k - \hat{\vec{x}}_k^+\right)\left(\vec{x}_k - \hat{\vec{x}}_k^+\right)^T\right]. \qquad (3.37) \end{aligned}$$

The material, terminology and motivation for the discrete-time Kalman filter has now been covered and the filter will now be formally defined.

Definition 3.1. Discrete-time Kalman filter
Assume

1. The linear system

$$\vec{x}_k = A\vec{x}_{k-1} + \vec{w}_k \tag{3.38}$$
$$\vec{y}_k = C\vec{x}_k + \vec{v}_k \tag{3.39}$$
$$\mathrm{E}\left(\vec{w}_k\vec{w}_j^T\right) = Q\delta_{k,j} \tag{3.40}$$
$$\mathrm{E}\left(\vec{v}_k\vec{v}_j^T\right) = R\delta_{k,j} \tag{3.41}$$
$$\mathrm{E}\left(\vec{w}_k\vec{v}_k^T\right) = 0, \tag{3.42}$$

where $\vec{x}_k \in \mathbb{R}^n$ and $\vec{y}_k \in \mathbb{R}^m$ are state and measurement at time t_k respectively. Observation matrix $C \in \mathbb{R}^{mxn}$ and state transition matrix $A \in \mathbb{R}^{nxn}$. Process noise $\vec{w}_k$ and sensor noise $\vec{v}_k$ are zero mean and Gaussian with variances $Q \in \mathbb{R}^{nxn}$ and $R \in \mathbb{R}^{mxm}$.

2. The estimate is initialized as

$$\vec{\hat{x}}_0^+ = \mathrm{E}\left[\vec{x}_0\right] \tag{3.43}$$
$$\hat{P}_0^+ = \mathrm{E}\left[\left(\vec{x}_0 - \vec{\hat{x}}_0^+\right)\left(\vec{x}_0 - \vec{\hat{x}}_0^+\right)^T\right]. \tag{3.44}$$

3. For each incoming measurement at time step $k = 1, 2, \ldots$ the following two phases are computed:

$$\textbf{predict phase} \tag{3.45}$$
$$\hat{x}_k^- = A\hat{x}_{k-1}^+ \tag{3.46}$$
$$\hat{P}_k^- = A\hat{P}_k^+A^T + Q \tag{3.47}$$

update phase

$$K_k = \hat{P}_k^- C^T \left(C\hat{P}_k^- C^T + R\right)^{-1} \tag{3.48}$$
$$\vec{\hat{x}}_k^+ = \vec{\hat{x}}_k^- + K_k\left(\vec{y}_k - C\vec{\hat{x}}_k^-\right) \tag{3.49}$$
$$\hat{P}_k^+ = (I - K_kC)\hat{P}_k^-, \tag{3.50}$$

where the estimate at t_k is $\vec{\hat{x}}_k^+$. $K_k \in \mathbb{R}^{nxm}$ is known as the Kalman gain.

Then (3.43 - 3.44) and (3.46 - 3.50) describe the discrete-time Kalman filter which computes an optimal unbiased linear estimate that minimizes the mean square error of estimate:

$$\mathbf{MMSE} = \mathrm{E}\left[\left(\vec{x}_k - \hat{\vec{x}}_k^+\right)^T \left(\vec{x}_k - \hat{\vec{x}}_k^+\right)\right]. \tag{3.51}$$

This is equivalent to minimizing the trace of $\hat{P}_k^+$ [15, Chapter 3],[4, Chapter 3],[7, Chapter 5],[6, Chapter 5].

If one sets A to the identity matrix and Q to zero in Definition 3.1 then the Kalman filter is equivalent to the recursive least squares estimator.

Example 3.1. Let x be the position of a car with initial position $x_0 = 0$ and let z be the fixed velocity equal to $z_0 = 1$ m s^{-1}. Let position measurements be taken every 0.5 seconds ($\Delta t = 0.5$), then the transition equation is

$$\begin{bmatrix} x \\ z \end{bmatrix}_k = \begin{bmatrix} 1 & (\Delta t) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ z_{k-1} \end{bmatrix} + \vec{w}_k \tag{3.52}$$

with measurements

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ z_k \end{bmatrix} + v_k = x_k + v_k. \tag{3.53}$$

Define the noise covariance matrices as

$$Q = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{3.54}$$

$$R = 0.3. \tag{3.55}$$

A sample realization of the filter with the defined parameters is shown in Figure 3.1.

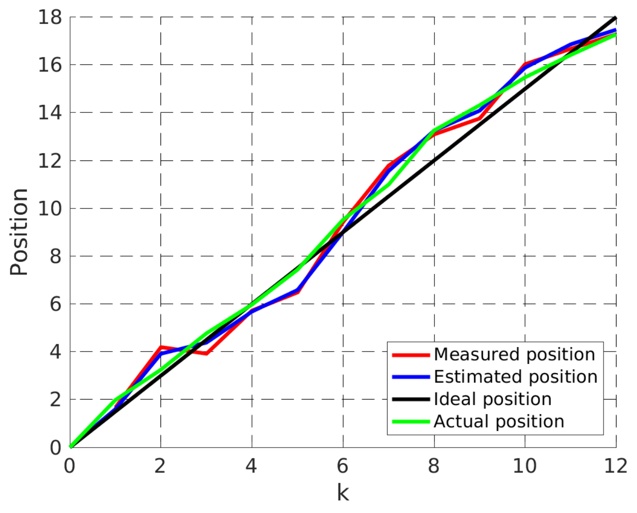

Figure 3.1: The graph shows four curves useful in understanding and assessing the performance of the dt-KF. Comparing the ideal position (without any process noise) to actual position shows the effect of Q. Comparing measured position to actual position shows the effect of R. Finally, comparing estimated position to actual position demonstrates the Kalman filter's performance. For examples where initial estimate does not match the state or other interesting examples please refer to [27].

3.3 Continuous-time Kalman Filter

The continuous-time Kalman filter (**ct-KF**) is different from the dt-KF as the state and measurements now evolve continuously in time. The ct-KF, also known as the Kalman-Bucy filter [8], can be derived from the dt-KF when one takes the limit as the time step decreases to zero. The details of this can be found in [36] and [42, Chapter 3]. The definition of ct-KF is similar to the definition of dt-KF in Definition 3.1. We note that the notation below, while standard for ODEs, is somewhat misleading when the stochastic component is included. While some aspects of the numerical treatment of stochastic DEs were discussed in the previous chapter, here we use the naive, ODE notation. The filter will now be defined.

Definition 3.2. Continuous-time Kalman filter
Assume

1. The linear system is given by

$$\frac{d}{dt}\left(\vec{x}(t)\right) = \dot{\vec{x}} = A\vec{x}(t) + \vec{w}(t), \tag{3.56}$$

$$\vec{y}(t) = C\vec{x}(t) + \vec{v}(t), \tag{3.57}$$

   where the observation matrix $C \in \mathbb{R}^{m\text{x}n}$ and $A \in \mathbb{R}^{n\text{x}n}$ are constant. The state $\vec{x}(t) \in \mathbb{R}^n$ and measurement $\vec{y}(t) \in \mathbb{R}^m$ have additive process noise $\vec{w}(t)$ and sensor noise $\vec{v}(t)$. The two noises are uncorrelated with each other in addition to being zero mean white noise:

$$\mathrm{E}\left[\vec{w}(t)\vec{w}(\tau)^T\right] = Q\delta(t-\tau) \tag{3.58}$$

$$\mathrm{E}\left[\vec{v}(t)\vec{v}(\tau)^T\right] = R\delta(t-\tau). \tag{3.59}$$

   The continuous-time impulse response $\delta(t-\tau)$ has a value of ∞ at $t=\tau$ and zero elsewhere with an area of 1. The impulse response in (3.58) and (3.59) signifies that the noise is infinitely correlated with itself at $t=\tau$, but has zero correlation with itself when $t \neq \tau$ [36, Page 231]. The matrices Q and R represent the intensity of the process and sensor noise respectively.

2. The filter's initial estimate $\hat{\vec{x}}(0)$ and covariance $P(0)$ are given by

$$\hat{\vec{x}}(0) = \mathrm{E}\left(\vec{x}_0\right), \tag{3.60}$$

$$P(0) = \mathrm{E}\left[\left(\vec{x}(0) - \hat{\vec{x}}(0)\right)\left(\vec{x}(0) - \hat{\vec{x}}(0)\right)^T\right]. \tag{3.61}$$

3. Using the initial conditions (3.60-3.61), the estimate $\hat{\vec{x}}(t)$, Kalman gain $K(t)$ and covariance $P(t)$ can be computed by solving

$$\dot{\hat{\vec{x}}}(t) = A\hat{\vec{x}}(t) + K\left(y(t) - C\hat{\vec{x}}(t)\right) \tag{3.62}$$

$$K(t) = P(t)C^TR^{-1} \tag{3.63}$$

$$\dot{P}(t) = -P(t)C^TR^{-1}CP(t) + AP(t) + P(t)A^T + Q. \tag{3.64}$$

   Equation (3.64) is referred to as the Differential Riccati Equation (**DRE**).

Then (3.60-3.61) and (3.62-3.64) describe the Kalman-Bucy filter which computes an optimal unbiased linear estimate that minimizes the mean square error of estimate[8, Chapter 4] [5] [42, Chapter 3],[6, Chapter 9]:

$$\textbf{MMSE} = \mathrm{E}\left[\left(\vec{x}(t) - \hat{\vec{x}}(t)\right)^T \left(\vec{x}(t) - \hat{\vec{x}}(t)\right)\right] = \mathrm{Trace}(P(t)). \tag{3.65}$$

In [36, Chapter 8] the ct-KF is defined as minimizing the cost function

$$J_e = \int_0^{t_f} \mathrm{E}\left[\left(\vec{x}(t) - \hat{\vec{x}}(t)\right)^T \left(\vec{x}(t) - \hat{\vec{x}}(t)\right)\right] dt \tag{3.66}$$

with the filter defined as it is in Definition 3.2.

Example 3.2. The transition equations (3.52) and (3.53) from example (3.1) are reformulated for the continuous case:

$$\frac{d}{dt}\left(\begin{bmatrix} x \\ z \end{bmatrix}\right) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + w(t) \tag{3.67}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + v(t) \tag{3.68}$$

Recall in this thesis that Δt is the time span between incoming measurements for the dt-KF (the dt-KF can be used for purely discrete systems with no mention of time). The equivalence between the continuous and discrete versions as $\Delta t \to 0$ is determined by the way the sensor and process noise covariance and system matrices are scaled [37]. Let the continuous version of sensor and process noise be R_c, Q_c and let R_d, Q_d be the discrete version. In addition, let A_c be the continuous transition matrix and let A_d be the discrete transition matrix, then the system matrices can be related as follows:

$$R_d = \frac{1}{\Delta t} R_c \tag{3.69}$$

$$Q_d = (\Delta t)\, Q_c \tag{3.70}$$

$$A_d = \exp\left(A_c \Delta t\right). \tag{3.71}$$

Example 3.3. The discrete approximation of the ct-KF from example (3.2) is given by:

$$A_d = \exp\left(A_c \Delta t\right) = \exp\left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \Delta t\right) = \begin{bmatrix} 1 & (\Delta t) \\ 0 & 1 \end{bmatrix} \tag{3.72}$$

which matches equation (3.43) in the discrete example (3.1) as expected. The noise matrices are now:

$$Q_c = \frac{1}{\Delta t}\begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix} \tag{3.73}$$

$$R_c = (\Delta t)\,(0.3) = 0.15. \tag{3.74}$$

The details of comparing the two are outside of the scope of this thesis, but are important for consideration when numerically approximating the ct-KF with the dt-KF. Another computational consideration is that for linear systems both the ct-KF and dt-KF do not require the measurement $\vec{y}$ to compute the Kalman covariance P. This decoupling means that the Kalman gain $K(t)$ can be computed in advance. The independence of the covariance P from the measurement will change for the nonlinear case.
Computing $P(t)$ for the ct-KF involves solving a differential equation. For a system with a sufficiently large number of states, large savings in computational effort can be achieved by using constant gain K instead of solving the DRE [36]. The constant gain is obtained through the steady state assumption where $\dot{P} = 0$.
Given that A,C,Q and R are constant, $P(t)$ may eventually reach a steady state covariance P_{ss} when $\dot{P} = 0$. The zero derivative assumption reduces the DRE to the Continuous Algebraic Riccati Equation (**CARE**):

$$-P_{ss}C^T R^{-1} C P_{ss} + AP_{ss} + P_{ss}A^T + Q = 0, \tag{3.75}$$

where Kalman gain K is now constant. A result that guarantees a steady state Kalman filter being stable [36] is provided by the following theorem:

Theorem 3.1. *The CARE has a unique positive semidefinite solution P_{ss} if and only if both conditions hold:*

- *(A, G) is stabilizable meaning there exists a matrix L such that the matrix $A+GL$ has eigenvalues with negative real parts where $GG^T = Q$*

- *(A, C) is detectable or equivalently (A^T, C^T) stabilizable*

The corresponding steady-state Kalman filter is stable; that is, all eigenvalues of $(A - KC)$ have negative real parts.

Similar steady results hold for the discrete case with more details found in chapter 4 of [1].

Example 3.4. Using A,C,Q,R from example (3.2) and G

$$G^T G = Q = \begin{bmatrix} \sqrt{0.1} & 0 \\ 0 & \sqrt{0.1} \end{bmatrix}, \tag{3.76}$$

(A, G) is stabilizable:

$$A + GL = A + G \begin{bmatrix} L_1 & L_2 \\ L_3 & L_4 \end{bmatrix} = \begin{bmatrix} 0.1L_1 & 0.1L_2 + 1 \\ 0.1L_3 & 0.1L_4 \end{bmatrix}. \tag{3.77}$$

Choose $L_1 = -10, L_2 = -10, L_3 = 0, L_4 = -10$ then

$$A + GL = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.78}$$

which has all eigenvalues with negative real parts. Therefore it is, stabilizable.
(A, C) detectable:

$$A^T + C^T L = A^T + C^T \begin{bmatrix} L_1 & L_2 \end{bmatrix} = \begin{bmatrix} L_1 & L_2 \\ 1 & 0 \end{bmatrix}. \tag{3.79}$$

Choose $L_1 = -1, L_2 = -1$ to prove (A, C) detectable.

The steady state matrix (computed by Matlab's CARE function) is

$$P_{ss} = \begin{bmatrix} 1.285 & 0.946 \\ 0.946 & 1.440 \end{bmatrix} \tag{3.80}$$

which results in a stable steady state Kalman filter.

The ct-KF requires the measurements to be continuous in time. This is not always the case as in many cases the measurements are made at discrete points in time. To allow for the state to evolve continuously and for discrete measurements, the dt-KF and ct-KF are combined to form the continuous-discrete Kalman filter **cd-KF**. It is now formally defined:

Definition 3.3. Continuous-Discrete Kalman filter
Assume

1. The linear system is given by

$$\frac{d}{dt}\left(\vec{x}(t)\right) = \dot{\vec{x}} = A\vec{x}(t) + \vec{w}(t) \tag{3.81}$$

   with measurements

$$\vec{y}_k = C\vec{x}_k + \vec{v}_k \tag{3.82}$$

   taken at t_k with fixed intervals between each measurement. Observation matrix $C \in \mathbb{R}^{mxn}$ and $A \in \mathbb{R}^{nxn}$ are constant. The state $\vec{x}(t) \in \mathbb{R}^n$ and measurement $\vec{y}_k \in \mathbb{R}^m$ have additive process noise $\vec{w}(t)$ and sensor noise $\vec{v}_k$ respectively. The two noises are uncorrelated with each other in addition to being Gaussian with the properties

$$\mathrm{E}\left[\vec{w}(t)\vec{w}(\tau)^T\right] = Q\delta(t-\tau) \tag{3.83}$$
$$\mathrm{E}\left(\vec{v}_k\vec{v}_j^T\right) = R\delta_{k,j}. \tag{3.84}$$

2. The estimate is initialized as

$$\hat{\vec{x}}(0) = \mathrm{E}\left[\vec{x}(0)\right] \tag{3.85}$$
$$\hat{P}(0) = \mathrm{E}\left[\left(\vec{x}(0) - \hat{\vec{x}}(0)\right)\left(\vec{x}(0) - \hat{\vec{x}}(0)\right)^T\right]. \tag{3.86}$$

3. For each incoming measurement at time step $k = 1, 2, \ldots$ the following two phases

are computed:

$$\textbf{predict phase} \tag{3.87}$$

initial conditions:

$$\vec{\hat{x}}(t_{k-1}) = \vec{\hat{x}}_{k-1}^{+} \tag{3.88}$$

$$\hat{P}(t_{k-1}) = P_{k-1}^{+} \tag{3.89}$$

differential equations :

$$\frac{d}{dt}\left(\hat{x}(t)\right) = A\vec{\hat{x}}(t) \tag{3.90}$$

$$\frac{d}{dt}\left(P(t)\right) = AP(t) + P(t)A^{T} + Q \tag{3.91}$$

solution:

$$\vec{\hat{x}}_{k}^{-} = \vec{\hat{x}}(t_k) \tag{3.92}$$

$$\hat{P}_{k}^{-} = P(t_k) \tag{3.93}$$

update phase

$$K_k = \hat{P}_{k}^{-}C^{T}\left(C\hat{P}_{k}^{-}C^{T} + R\right)^{-1} \tag{3.94}$$

$$\vec{\hat{x}}_{k}^{+} = \vec{\hat{x}}_{k}^{-} + K_k\left(\vec{y}_k - C\vec{\hat{x}}_{k}^{-}\right) \tag{3.95}$$

$$\hat{P}_{k}^{+} = (I - K_kC)\hat{P}_{k}^{-}, \tag{3.96}$$

where the estimate at t_k is $\vec{\hat{x}}_{k}^{+}$. $K_k \in \mathbb{R}^{nxm}$ is known as the Kalman gain.

The equations describe the continuous-discrete Kalman filter which computes an optimal unbiased linear estimate that minimizes the mean square error of estimate [42, Page 194].

For sensor placement investigation it will be assumed that measurements are taken at discrete points in time meaning that cd-KF and dt-KF will be used.

3.4 Kalman Filter Implementation

The Kalman filter version used in this thesis will rely on incoming measurements for which the time between successive measurements is fixed. This specification restricts the Kalman filter to run a predict phase followed by an update phase for every measurement as seen in the dt-KF Definition (3.1). The introduction to the intuition behind the phases as well as basic coded examples can be found in [28]. In brief, the predict phase propagates both the estimate and covariance all based on the system and assumed process noise. The update phase adjusts the predict phase's estimate based on the current measurement using the Kalman gain [4, Chapter 3]. In this section the implementation details of the update and predict phase will be explained and the extended Kalman filter is introduced.

3.4.1 Update Phase

Consider the update phase

$$K_k = \hat{P}_k^- C^T \left(C P_k^- C^T + R\right)^{-1} \tag{3.97}$$

$$\vec{\hat{x}}_k^+ = \vec{\hat{x}}_k^- + K_k \left(\vec{y}_k - C\vec{\hat{x}}_k^-\right) \tag{3.98}$$

$$\hat{P}_k^+ = (I - K_k C)\hat{P}_k^-, \tag{3.99}$$

as seen in dt-KF Definition (3.1) and cd-KF Definition (3.3). Programming (3.97 - 3.99) is straightforward. However, one has to be careful as “one consequence of round-off error is that the computed $[P_k^+]$ may be non-Hermitian" [18] which would violate the property of the covariance matrix being symmetric. One potential solution for this is to add a step after (3.99) that would average the covariance $\hat{P}_k^+$ and its transpose.

$$\hat{P}_k^+ = \frac{1}{2}\left(\hat{P}_k^+ + \left(\hat{P}_k^+\right)^T\right). \tag{3.100}$$

Unfortunately, (3.100) does not address rounding errors potentially leading to the matrix being non positive semidefinite [18] thus violating the definition of a covariance matrix. This can be resolved by computing the square root factor matrix or the Cholesky factor

$$\left(P_k^+\right)^{\frac{1}{2}}, \tag{3.101}$$

where

$$\hat{P}_k^+ = \left(\hat{P}_k^+\right)^{\frac{1}{2}} \left(\left(\hat{P}_k^+\right)^{\frac{1}{2}}\right)^*, \tag{3.102}$$

where superscript $*$ represents the conjugate transpose. Given that the factor can be computed, then $\hat{P}_k^+$ can be reconstructed as in (3.102) while still maintaining positive semidefinite property since

$$\vec{x}^T P_k^+ \vec{x} = \vec{x}^T \left(\left(\hat{P}_k^+\right)^{\frac{1}{2}} \left(\left(\hat{P}_k^+\right)^{\frac{1}{2}} \right)^* \right) \vec{x} \tag{3.103}$$

$$= \left(\vec{x}^T \left(\hat{P}_k^+\right)^{\frac{1}{2}} \right) \left(\left(\left(\hat{P}_k^+\right)^{\frac{1}{2}} \right)^* \vec{x} \right) \tag{3.104}$$

$$= \vec{z}^* \vec{z} \tag{3.105}$$

$$\geq 0. \tag{3.106}$$

Other potential sources of errors as well as detailed dicussion on various implementations and their motivations can be found in [14, Chapter 7]. If one assumes that the initial covariance $\hat{P}_0^+$ is positive definite then by the Cholesky factorization the factor exists and is lower triangular such that (3.102) holds [13, Chapter 4.2.3]. Provided that the square root factor is propagated consistently with respect to the update and predict phase of the filter then the initial positive definite covariance and a basic recursion argument guarantees that each $\hat{P}_k^+$ will be positive semidefinite for all k. The QR decomposition (i.e. the decomposition of a matrix A into product QR where Q unitary and R upper triangular) will be the basis behind the update phase for propagating the square root factor consistently. Consider the square root factor

$$\begin{bmatrix} R^{\frac{1}{2}} & C\left(\hat{P}_k^-\right)^{\frac{1}{2}} \\ 0 & \left(\hat{P}_k^-\right)^{\frac{1}{2}} \end{bmatrix} \tag{3.107}$$

of the matrix

$$\begin{bmatrix} R + C\hat{P}_k^- C^T & C\left(\hat{P}_k^-\right) \\ \left(\hat{P}_k^-\right)^T C^T & \hat{P}_k^- \end{bmatrix}. \tag{3.108}$$

When (3.107) is multiplied by some unitary matrix it will equal a lower triangular square root factor matrix

$$\begin{bmatrix} R^{\frac{1}{2}} & C\left(P_k^-\right)^{\frac{1}{2}} \\ 0 & \left(P_k^-\right)^{\frac{1}{2}} \end{bmatrix} \Theta = \begin{bmatrix} X & 0 \\ Y & Z \end{bmatrix}. \tag{3.109}$$

The unitary matrix, in order to compute the lower triangular square factor, is specified by the QR decomposition. The square root factor implies that (3.108) must equal

$$\begin{bmatrix} XX^* & XY^* \\ YX^* & YY^* + ZZ^* \end{bmatrix}. \tag{3.110}$$

Through algebraic manipulation it can be shown the following holds (for details see appendix (A)):

$$Z = \left(P_k^+\right)^{\frac{1}{2}} \tag{3.111}$$

$$YX^{-1} = K_k. \tag{3.112}$$

Provided that $\hat{P}_k^-$ is in square root form, as in (3.109), then after the update phase one will compute $\hat{P}_k^+$ in square root form. It is left to demonstrate that the predict phase can also propagate the covariance matrix in square root form to conclude that the implementation of the Kalman filter guarantees the positive semidefitness of the covariance.

3.4.2 Predict Phase

The predict phase is responsible for propagating the estimate and covariance matrix based on the model and assumed process noise. The update phase was introduced first as it is the same for both continuous ($\dot{x} = Ax$) and discrete ($x_{k+1} = Ax_k$) models. The predict phase is where the implementations of the two differ.

A significant portion of the propagation within the predict phase involves adding matrices together. In theorem (3.2) it is demonstrated that adding two matrices with square root factors produces a matrix with a square root factor.

Theorem 3.2. *If adding matrices $Z \in \mathbb{R}^{m\times m}$ and $Y \in \mathbb{R}^{m\times m}$ which both have square root factors*

$$Y = Y^{\frac{1}{2}}\left(Y^{\frac{1}{2}}\right)^* \in \mathbb{R}^{m\times m} \tag{3.113}$$

$$Z = Z^{\frac{1}{2}}\left(Z^{\frac{1}{2}}\right)^* \in \mathbb{R}^{m\times m}, \tag{3.114}$$

then the sum equals a matrix that also has a square root factor.

Proof:

$$Y+Z = Y^{\frac{1}{2}}\left(Y^{\frac{1}{2}}\right)^{*} + Z^{\frac{1}{2}}\left(Z^{\frac{1}{2}}\right)^{*} \tag{3.115}$$
$$= Y^{\frac{1}{2}}\Theta\Theta^{*}\left(Y^{\frac{1}{2}}\right)^{*} + Z^{\frac{1}{2}}\Theta\Theta^{*}\left(Z^{\frac{1}{2}}\right)^{*} \tag{3.116}$$
$$= \begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix}\Theta\Theta^{*}\begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix}^{*} \tag{3.117}$$
$$= \left(\begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix}\Theta\right)\left(\begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix}\Theta\right)^{*} \tag{3.118}$$
$$= XX^{*}, \tag{3.119}$$

where $\Theta \in \mathbb{R}^{mxm}$ is a unitary matrix ($\Theta\Theta^{*} = I$). The last line (3.119) signifies that the sum of two matrices with square root factors can be represented as a matrix with square root factors. One can specify Θ by taking the QR decomposition that factors some m-by-n matrix A into product QR where Q is m-by-m and orthogonal while R is m-by-n and upper triangular [13, Chapter 5.2]. Consider the factor from (3.119)

$$\begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix}\Theta = \begin{bmatrix}X & 0\end{bmatrix} \tag{3.120}$$
$$\begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix} = \begin{bmatrix}X & 0\end{bmatrix}\Theta^{*} \tag{3.121}$$
$$\begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix}^{*} = \Theta\begin{bmatrix}X^{*} \\ 0^{T}\end{bmatrix}, \tag{3.122}$$

then the QR factorization where $Q = \Theta$ and $R = \begin{bmatrix}X & 0\end{bmatrix}^{*}$ of the matrix $\begin{bmatrix}Y^{\frac{1}{2}} & Z^{\frac{1}{2}}\end{bmatrix}^{*} \in \mathbb{R}^{mx2m}$ can be used to determine the sum of Z and Y.

Discrete Case

From dt-KF Definition (3.1) the predict phase is

$$\hat{x}_k^{-} = A\hat{x}_{k-1}^{+} \tag{3.123}$$
$$\hat{P}_k^{-} = A\hat{P}_{k-1}^{+}A^{T} + Q. \tag{3.124}$$

By following example (3.2) the line (3.124) can be expressed as

$$\left[A\left(\hat{P}_{k-1}^{+}\right)^{\frac{1}{2}} \quad (Q)^{\frac{1}{2}}\right]\Theta = \left[\left(\hat{P}_k^{-}\right)^{\frac{1}{2}} \quad 0\right] \tag{3.125}$$

by using the QR decomposition. Thus the predict phase (3.124) can be described as (3.125) which transform a square root covariance to another square root covariance matrix.

Continuous Case

The state will evolve continuously while the measurements will be taken at discrete intervals in time and therefore the cd-KF will be used. The ct-KF implementation will not be discussed as continuous measurements will not be considered in the thesis. The predict phase for cd-KF from Definition (3.3) is

$$\text{input:}$$

$$\vec{\hat{x}}(t_{k-1}) = \vec{\hat{x}}^{+}_{k-1} \tag{3.126}$$

$$\hat{P}(t_{k-1}) = P^{+}_{k-1} \tag{3.127}$$

$$\text{differential equations :}$$

$$\frac{d}{dt}\left(\vec{\hat{x}}(t)\right) = A\vec{\hat{x}}(t) \tag{3.128}$$

$$\frac{d}{dt}\left(P(t)\right) = AP(t) + P(t)A^{T} + Q \tag{3.129}$$

$$\text{output:}$$

$$\vec{\hat{x}}^{-}_{k} = \vec{\hat{x}}(t_k) \tag{3.130}$$

$$P^{-}_{k} = P(t_k) \tag{3.131}$$

For solving (3.128) Runge-Kutta (**RK4**) method will be used to evolve $\vec{\hat{x}}(t)$ from measurement at time t_{k-1} to the next at t_k ($\Delta t = t_k - t_{k-1}$). Fixed size time steps are taken of size h where $h = t_{k-1|n+1} - t_{k-1|n}$ is the n^{th} RK4 step in the current predict step with m total steps ($mh = \Delta t$). The scheme is computed as follows

$$\vec{k}_1 = hf(\vec{x}_{k-1|n}, t_{k-1|n}) \tag{3.132}$$

$$\vec{k}_2 = hf\left(\vec{x}_{k-1|n} + \frac{k_1}{2}, t_{k-1|n} + \frac{h}{2}\right) \tag{3.133}$$

$$\vec{k}_3 = hf\left(\vec{x}_{k-1|n} + \frac{k_2}{2}, t_{k-1|n} + \frac{h}{2}\right) \tag{3.134}$$

$$\vec{k}_4 = hf\left(\vec{x}_{k-1|n} + k_3, t_{k-1|n} + h\right) \tag{3.135}$$

$$\vec{x}_{k-1|n+1} = \vec{x}_{k-1|n} + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}, \tag{3.136}$$

where $f(\vec{x},t) = A\vec{x}$. For every timestep in the RK4 method, the covariance matrix will also be evolved by solving (3.129) where $J(t)$ for the linear model is equal to A. Equation (3.129) over the Δt interval can be written in the form

$$P(t_k) = \phi(t_k)P_{k-1}^{+}\phi(t_k)^T + \int_{t_{k-1}}^{t_k} \phi(t)G(t)QG(t)^T\phi(t)^T dt \tag{3.137}$$

$$\frac{d\phi(t)}{dt} = J(t)\phi(t) \text{ and } \phi(t_{k-1|1}) = \Phi_1 = I. \tag{3.138}$$

One can solve (3.138) using RK4 with the same timesteps as for (3.128) to obtain a set of $\Phi_n = \phi(t_{k-1|n})$ [11]. The integral (3.137) can be approximated by quadrature using the Trapezoidal rule [35]

$$P(t_k) = P_k^- \approx \Phi_m P_{k-1}^{+}\Phi_m^T + h\left(\frac{1}{2}\Phi_1 Q\Phi_1^T + \Phi_2 Q\Phi_2^T + \ldots + \frac{1}{2}\Phi_m Q\Phi_m^T\right). \tag{3.139}$$

In example (3.2) with the use of QR decomposition it was shown that the sum of two matrices with square root factors equals a matrix that can also be expressed in square root form. Hence, given that $\hat{P}_{k-1}^{+}$ and Q can be expressed in square root form it can be shown that the predict phase maintains positive semidefitness.

3.4.3 Implementation

Previously, the math behind each iteration of the predict-update phases of the Kalman filter was described. In this section the specific implementation will be discussed. The math described is closest to what is known as the Square Root Covariance Filter (**SRCF**) [39] [14]. In that implementation, similar to the update phase, a unitary matrix Θ exists such that

$$\begin{bmatrix} R^{\frac{1}{2}} & C\left(\hat{P}_{k-1}^{+}\right)^{\frac{1}{2}} & 0 \\ 0 & A\left(\hat{P}_{k-1}^{+}\right)^{\frac{1}{2}} & Q^{\frac{1}{2}} \end{bmatrix}\Theta = \begin{bmatrix} X & 0 & 0 \\ Y & Z & 0 \end{bmatrix}, \tag{3.140}$$

where

$$\hat{\vec{x}}_k^{+} = A\hat{\vec{x}}_{k-1} - YX^{-}\left(C\hat{\vec{x}}_{k-1} - y_k\right) \tag{3.141}$$

$$\hat{P}_k^{+} = ZZ^{*}. \tag{3.142}$$

The matrix Θ can be obtained through the QR decomposition that itself can be obtained through the use of the Householder transformation [35]. However, the implementation used in this thesis splits (3.140) into QR decomposition for both update and predict phase separately. The reason for this is to allow the RK4 algorithm to take multiple steps (3.139) ($m > 1$) and the reuse of update phase code between cd-KF and dt-KF. Despite the change from SRCF, the discrete predict and update phases are not original work and in [18] are introduced separately as lead up to (3.140). The predict phase of the cd-KF takes inspiration mainly from [17] where the update phase was implemented identically while the quadrature rule was used to sum up Φ's as in (3.139) which was computed using a different Runge Kutta scheme. In addition, [11] also refers to using (3.139) with RK4 in predict phase while [25] solves the predict phase's DE's directly (3.129) while still using the triangularization technique to advance after each timestep.

The runtime of single iteration of the Kalman filter is $O(n^3)$ [39], [18] where n is the state count. For m total measurements this leads to runtime $O(mn^3)$. Sample Matlab code can be found in appendix (B) and additional C++ code and other filter implementations can be found on Github with link in appendix (B).

3.5 Extended Kalman Filter

The Extended Kalman filter (**EKF**) is an extension of the existing definitions. The EKF handles the nonlinear case where the state evolves as

$$\dot{\vec{x}}(t) = g(\vec{x}) + \vec{w}(t) \tag{3.143}$$

in the continuous case and

$$\vec{x}_{k+1} = f(\vec{x}_k) + \vec{w}_k, \tag{3.144}$$

for the discrete case where f and g are nonlinear functions of the state x and w is the process noise. The EKF algorithm linearizes f and g through the Jacobian J about the current estimate and modifies the predict phase for the discrete case as

$$\hat{\vec{x}}_k^- = f(\hat{\vec{x}}_{k-1}^+) \tag{3.145}$$

$$\hat{P}_k^- = J\hat{P}_k^+ J^T + Q, \tag{3.146}$$

where J is the Jacobian of f evaluated at $\hat{\vec{x}}_{k-1}^+$. The EKF for the discrete case will be referred to as the discrete-discrete Extended Kalman filter (**dd-EKF**). In the linear case, when $\vec{x}_{k+1} = A\vec{x}_k + \vec{w}_k$ then $J = A$, the dd-EKF reduces to the previously defined dt-KF.

For the continuous case, the predict phase is

$$\frac{d}{dt}\left(\hat{\vec{x}}_k(t)\right) = g(\hat{\vec{x}}_k(t)) \tag{3.147}$$

$$\frac{d}{dt}\left(P_k(t)\right) = J(t)P_k(t) + P_k(t)J(t)^T + Q, \tag{3.148}$$

where $J(t)$ is the Jacobian of g evaluated at $\hat{\vec{x}}(t)$. The EKF for the continuous case will be known as the **cd-EKF** and when g is a linear function of $\vec{x}$ then cd-EKF reduces to the cd-KF. More detailed introductions of the EKF for continuous and discrete cases can be found in [6] [36].

For the EKF the Jacobian is computed around the current estimate, which in the update phase is a function of the current measurement. This means that unlike the regular Kalman filter, the covariance computations are not decoupled from the state estimate [6], and hence cannot be computed offline. In addition EKF is "numerically scarcely affordable in high-dimensional systems" [4]. The EKF needs to compute and store the a priori and a posteriori covariance matrices which is computational taxing for large n whereas other filters such as the Ensemble Kalman filter (EnKF) do not require to compute the covariance matrices.

The EKF linearizes about the current estimate and propagates the covariance with the use of the Jacobian. This first order approximation has the potential of introducing unmodeled errors that violate some basic assumptions about the predictions errors - unbiased estimate and having covariances match the computed by filter [6]. The approximation factor prevents the EKF of having the MMSE guarantee as it does for the linear case in Definitions 3.1, 3.2 and 3.3. In fact, there "are almost no useful analytical results on the performance of the EKF. A considerable amount of experimation and 'tuning' is needed to get a reasonable filter. However, this has been done in numerous different applications" [18] such as aerospace navigation [34]. Tuning can be described as adhoc and in [10] can possibly involve inflating the assumed process noise.

The EKF is not the only filter designed for nonlinear estimation. In fact, other filters such as the Unscented Kalman filter (UKF) and EnKF exist and are related to the Kalman filter. For example, the UKF can capture mean and covariance to a higher order (second) for any nonlinearity when compared to the EKF [16]. In "the less frequent case, EKF loses track of truth, while the EnKF still follows it" [4].

The previous two paragraphs are meant to remind the reader that the Kalman filter is an algorithm and its performance needs to be checked. The "limits of successful use of the linearization techniques implicit in the EKF can be obtained only via extensive Monte Carlo simulations for consistency verification" [6].

Chapter 4

Sensor placement introduction

When choosing to place a sensor in location A over location B one expects, and at the very least hopes, that the measurements obtained in A will be better or more useful. The criterion 'better' and 'useful' are vague and this is addressed by choosing a cost function to provide a quantitative comparison between various locations. The cost is then minimized over the set of all candidate sensor locations with the help of some algorithm. Finally, the performance of the chosen sensor is verified to confirm its optimality.

This chapter is divided into two sections. In the first, the chosen cost function will be motivated and discussed. In the second, methods of verifying the cost function will be discussed along with other issues. The presentation of the concepts will be supplemented with detailed but simple examples of linear systems. Results for nonlinear systems will be presented in the next chapter.

4.1 Cost Function

Many sensor placement techniques use observability based as outlined in the review of sensor placement [38]. Borrowing notation from chapter 3, the famous result states that the Kalman Observability Matrix

$$\begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{4.1}$$

has rank n if and only if the system is observable. A system is said to be observable if it is possible to determine the state $\vec{x}(t)$ from the measurements $\vec{y}(t)$. The observability is a binary measure - either it is observable or not. Quantitative observability based measures exist, such as maximizing the eigenvalues of the solution to a Lyapunov equation in order to "maximize the output energy generated by a given state" [38], [44].

The approach considered in this thesis is not based on observability, but on another contribution of Kalman - the Kalman filter. Kalman filter based sensor placement is a well known approach that "aims at minimizing state-estimation errors" [38]. In an earlier review [24] sensor placement is discussed for distributed parameter systems - systems with an infinite-dimensional state space. A significant portion of optimal sensor techniques for optimal state estimation discussed in [24] involve reducing the infinite-dimensional to a finite-dimensional system by truncation. This will also be done in this thesis.

The Kalman filter based approach involves minimizing the trace of the error covariance operator. In particular, Kumar and Seinfeld [26] focused on minimizing the Kalman trace in linear systems, but for effective computations used a suboptimal criterion by setting an upperbound on the covariance. In addition, Seinfeld and Yu in [43] considered optimal sensor placement of the finite-dimensional approximation using the steady sensor placement to be reviewed in (4.4) and extended it to a suboptimal approach for multiple sensors. The Kalman based approaches discussed in [26] and [43] and others reviewed in [24] differ little from the concepts introduced here. This thesis investigates the Kalman trace with a brute force approach by evaluating the trace over many candidate sensor locations. In addition, the approach is extended to a nonlinear system.

4.1.1 Steady State Kalman Filter

The first cost function is based on the steady state Kalman filter introduced in chapter 3 in (3.75). Previous work using this approach goes back to early 1970s [43] [9], with more recent work covered in the past few years [44],[22] and [23]. Given that the assumptions of Theorem 3.1 are satisfied then the covariance matrix $P(t)$ given by the Differential Riccati Equation will converge to the matrix provided by solving the Algebraic Riccati equation:

$$-P_{ss}C^TR^{-1}CP_{ss} + AP_{ss} + P_{ss}A^T + Q = 0, \tag{4.2}$$

where P_{ss} is the steady state covariance which is dependent on the observation matrix C, along with the system matrix A and noise covariances Q and R. For a linear system with noise conditions as outlined in continuous and discrete Definitions ((3.1),(3.2),(3.3)), the Kalman filter minimizes the mean square error, or equivalently the trace of covariance, at time t, or step k, respectively. The Kalman filter covariance will tend to the steady state covariance P_{ss} as $t \to \infty$ whose trace will therefore also be minimized. The matrix P_{ss} is dependent on the observation matrix C. In return, C is directly dependent on the sensor location x_0 (later on, x and x_0 will be used interchangeably indicate sensor location - x). By varying x_0 the matrix P_{ss} is directly modified and therefore the trace. The cost function to be minimized for the steady state approach is

$$J_{ss} = \text{Trace}\left(P_{ss}\right). \tag{4.3}$$

The optimal sensor placement location x_0 is considered the one that minimizes the trace over all candidate locations considered:

$$\begin{aligned} &\min\ J_{ss} \\ &\text{subject to} \\ -P_{ss}C^T R^{-1} C P_{ss} + A P_{ss} + P_{ss} A^T + Q = 0& \\ &C = f(x_0) \\ &x_0 \in X, \end{aligned} \tag{4.4}$$

where $f(x_0)$ signifies that the observation matrix C is a function of the sensor placement x_0. The sensor placement x_0 is chosen from a global set of candidate positions. The matrices A, Q and R are fixed for a given problem.

Example 4.1. Consider the two term truncation of the heat equation

$$u_t = \frac{1}{5} u_{xx} \tag{4.5}$$

$$u(x = 0, t) = u(x = \pi, t) = 0 \tag{4.6}$$

$$u(x, t) \approx \sum_{n=1}^{2} a_n(t) \sin(nx). \tag{4.7}$$

The PDE (4.5) is approximated through (4.7) and with added process noise is expressed in control theoretic notation:

$$\frac{d}{dt}\left(\begin{bmatrix} a_1(t) \\ a_2(t) \end{bmatrix}\right) = \begin{bmatrix} -0.2 & 0 \\ 0 & -0.8 \end{bmatrix} \begin{bmatrix} a_1(t) \\ a_2(t) \end{bmatrix} + \vec{w}(t), \tag{4.8}$$

where $\vec{w}(t)$ is the zero mean process noise term with covariance

$$Q = 10^{-8} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}. \tag{4.9}$$

The entries of the state $\vec{x}$ are a_1 and a_2. The sensor will measure the average value of u in a small region around $x = x_0$:

$$y(t) = \frac{1}{2\delta}\int_{x_0-\delta}^{x_0+\delta} u(x,t)dx + v(t) \tag{4.10}$$

$$= \frac{1}{2\delta}\begin{bmatrix}\int_{x_0-\delta}^{x_0+\delta} \sin(x)\,dx & \int_{x_0-\delta}^{x_0+\delta} \sin(2x)\,dx\end{bmatrix}\begin{bmatrix}a_1\\a_2\end{bmatrix} + v(t) \tag{4.11}$$

$$= C\begin{bmatrix}a_1\\a_2\end{bmatrix} + v(t), \tag{4.12}$$

where $v(t)$ is the sensor noise with variance 10^{-6} and $\delta = 0.01$. Selecting x_0 at discrete points in the domain

$$x_0 \in \{0.1, 0.15, 0.2, \ldots, 3.1\} \tag{4.13}$$

and computing J_{ss} for each candidate location one can plot (4.1). The location that minimizes (4.3) over all candidate locations is $x_0 = 1.55$ with $J_{ss} = 1.0785 \times 10^{-7}$.

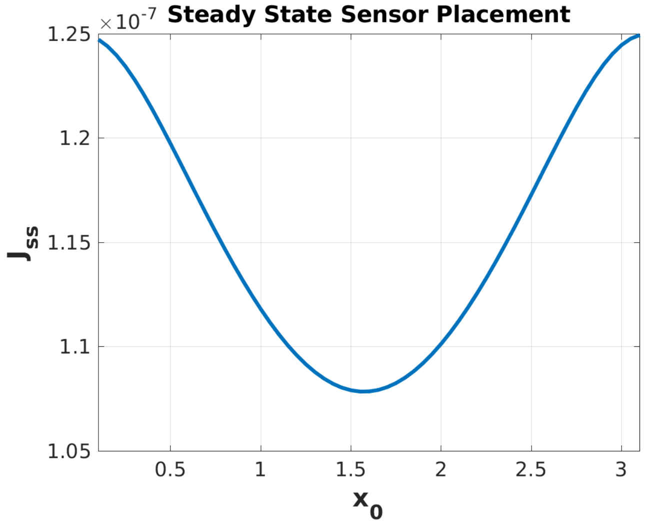

Figure 4.1: Trace of the steady state Kalman covariance is computed for all x_0. The location that is the optimal sensor placement is $x = 1.55$ as it minimizes the cost function J_{ss} in (4.4)

When placing a sensor at location $x_0 = \pi/2$ then J_{ss} is 1.0784×10^{-7} which is less than the cost computed at $x = 1.55$. The location $\pi/2$ is the node for the second mode.

It may seem counter intuitive to place a sensor based on the minimal cost at the node as it would not be able to pick up the second mode's amplitude. These concerns are addressed in later examples of this chapter suggesting a potential problem in the steady state based sensor placement approach.

The specific choice of process noise in example (4.1) may seem peculiar and in this case has no particular physical motivation. In [43] it is shown that given that if a PDE has a dynamical noise term $\xi(x,t)$ that is space and time dependent, then the noise covariance can be expressed using the modes and in the limit of mode count the covariance of ξ "can be approximated arbitrarily closely in the mean square sense" by the mode covariance. A general method to construct the modal covariance is shown in [43] and a specific example is provided in example 10.20 of [33] for the heat equation. Convergence of the finite-dimensional approximation to the infinite-dimensional Kalman filter is important but out of the scope of this thesis - further specialized discussion can be found in [12]. A recent example discussing convergence for finite time is in [41] where the importance of the initial covariance $P(0)$ being nuclear (i.e. the singular values are summable) is highlighted in Fig 1 and 2 of [41]. For nonnuclear initial covariances, such as the identity matrix, the filter does not converge as mode count N is increased. For infinite time refer to [44].

4.1.2 Transient Kalman Filter

The cost function considered in this thesis is an extension of (4.3) and the associated optimization problem (4.4). The focus is now on what happens before steady state is reached, or during the transient. Another, common phrase for the 'transient' is 'finite-time horizon'. This change requires the definition of an initial covariance $P(0)$ at time zero and the computation of the covariance over some time range of interest. In order to minimize a scalar as in (4.4) it is chosen to minimize the average trace over a specific time span $[t_s, t_f]$:

$$J_{[t_s,t_f]} = \frac{1}{t_f - t_s}\int_{t_s}^{t_f} \text{Trace}\,(P(t))\,dt. \tag{4.14}$$

The timespan will be chosen in an ad-hoc manner. The cost (4.14) has been used before as mentioned in sensor placement review [38]. The steady state covariance sensor approach depends on solving for P_{ss} when the derivative of P in the differential Riccati equation (DRE) is set to zero. In the transient approach considered in this thesis, the DRE will not be solved as the measurements are assumed to be no longer continuous in time. The state still evolves continuously so the covariance will be computed using the continuous-discrete Kalman filter introduced in Definition (3.3).

$$\begin{aligned}
&\min J_{[t_s,t_f]} \\
&\text{subject to} \\
P(0) &= \mathrm{E}[\left(\vec{x}(0)-\vec{\hat{x}}(0)\right)\left(\vec{x}(0)-\vec{\hat{x}}(0)\right)^T] \\
\hat{P}(t_{k-1}) &= P^+_{k-1} \\
\frac{d}{dt}\left(\hat{x}(t)\right) &= A\vec{\hat{x}}(t) \\
\frac{d}{dt}\left(P(t)\right) &= AP(t)+P(t)A^T+Q \\
\hat{P}^-_k &= P(t_k) \\
K_k &= \hat{P}^-_k C^T\left(C\hat{P}^-_k C^T+R\right)^{-1} \\
\hat{P}^+_k &= (I-K_kC)\hat{P}^-_k \\
C &= f(x_0) \\
x_0 &\in X
\end{aligned} \tag{4.15}$$

The transient problem (4.15) is expressed for the continuous-discrete Kalman filter (cd-KF). The difference in optimal location between the continuous Kalman filter based steady state approach and the cd-KF transient approach will be explored in the next example.

Example 4.2. The same system as in example 4.1 is used. For the linear case the covariance $P(t)$ is dependent only on the initial covariance $P(0)$ which is specified to be

$$P(0) = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}. \tag{4.16}$$

In Figure 4.2 the trace values for $x = 0.85$ and $x = 1.6$ over the interval $t \in [0, 20]$ are plotted.

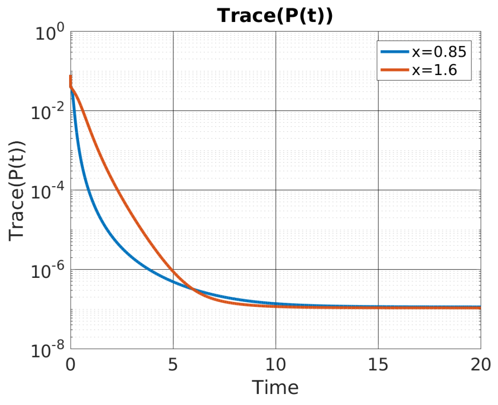

Figure 4.2: The trace of the Kalman filter covariance is plotted for two different sensor locations. Around $t = 6$ the trace at $x = 1.6$, which is initially larger then that at $x = 0.8$, stops being larger than that at $x = 0.85$.

In Figure 4.2 it is shown that for a fixed sensor location, the value of the trace is changing. This alone is not interesting but when contrasted against the other location suggests that the optimal location given by (4.15) can change depending on the time interval. One can plot multiple curves together for all candidate x_0 locations and use a log colour scale as seen in Figure 4.3.

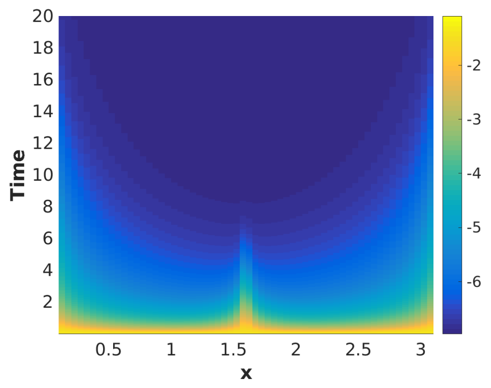

Figure 4.3: The log of the trace of the Kalman covariance for various sensor locations for the same candidate set considered in example (4.1). As time increases, the trace decreases.

Applying minimization problem (4.15) on the results in Figure 4.3 one will find that that the optimal location for the sensor is $x = 1.0$ for $t_s = 0$ and t_f in the range $(0, 10000]$. At $t = 100$, the computed covariance $P(t)$ varies very little with an effective 'steady state' reached where the max relative error between P_{ss} from previous example trace and $P(t)$ trace for all sensor locations is on the order of 10^{-5}. This low relative error is despite P_{ss} depending on ct-KF and $P(t)$ depending on cd-KF. The discussion of approximating ct-KF with cd-KF will not be discussed further. The assumption for all remaining results is that measurements are discrete in time. An interesting observation is that the optimal steady state based sensor placement is highly unoptimal sensor location in the transient case for low t_f as seen by the protrusion in the middle of Figure 4.3. This observation suggests care should be taken in using steady state based approaches.

The transient approach considered in this thesis is of a brute force nature meaning that a candidate-by-candidate location test is conducted. In example (4.2), 61 candidates were evaluated for 10000 timesteps each using a continuous-discrete Kalman filter defined in Definition (3.3). For low state size $N = 2$ this is a computational cost of $2s \times 61 = 122s$ in serial execution. Despite the highly parallelizable nature of the approach, the computation time is very sensitive to candidate count and especially sensor count. The "brute-force approach is indirect and not solvable in time polynomial" in M where M is sensor count [38]. This is an undesirable property for a sensor placement strategy - which is the reason why only single sensor placement will be investigated.

4.2 Verification

Verifying the performance of a sensor location is important in gauging how effective the cost function is at determining optimality. The cost function in this thesis is based on the trace of the covariance computed by the Kalman filter as outlined previously. The filter computes a covariance $P(t)$ used for the cost and the estimate $\hat{x}$ to be used for assessment.

The covariance, in the continuous case, can be expressed in a few ways

$$\text{Trace}\,(P(t)) = \text{Trace}\left(\text{E}\left[\left(\vec{\hat{x}}(t) - \vec{x}(t)\right)\left(\vec{\hat{x}}(t) - \vec{x}(t)\right)^T\right]\right) \tag{4.17}$$

$$= \text{E}\left[\left(\vec{\hat{x}}(t) - \vec{x}(t)\right)^T\left(\vec{\hat{x}}(t) - \vec{x}(t)\right)\right] \tag{4.18}$$

$$= \text{E}\left[\left\|\vec{\hat{x}}(t) - \vec{x}(t)\right\|^2\right]. \tag{4.19}$$

Similar expressions to (4.17) - (4.19) exist for the discrete case. The last expression (4.19) in words is the expected value of squared norm of the error between the estimate and true state. The finite sample approximation of (4.19) is

$$\frac{1}{K}\sum_{i=1}^{K}\left\|\vec{\hat{x}}_i(t) - \vec{x}_i(t)\right\|^2, \tag{4.20}$$

where i stands for the i^{th} run of K total Monte-Carlo simulations (not to be confused for the k^{th} time step in the discrete case. In the linear case, the computation of the covariance is independent of sampled process and sensor noise as seen in Definitions (3.1) and (3.2) and therefore a single run of the Kalman filter is sufficient to compute (4.19) for fixed initial conditions. The independence does not hold for the computation of the estimate since on a computer the noise is sampled and therefore many trials are necessary in order to compute (4.20). The square root of (4.20) is known as the root mean-square error (**RMSE**):

$$\text{RMSE} = \sqrt{\frac{1}{K}\sum_{i=1}^{K}\left\|\vec{\hat{x}}_i(t) - \vec{x}_i(t)\right\|^2}. \tag{4.21}$$

The scalar (4.21) is commonly used to asses the performance of estimators. There is a strong connection between (4.21) and (4.19), however they are not be confused with each other. "The former is a ruler used to measure the performance of estimators in the evaluation process," while the latter is the "mean-square error (MSE) a theoretical optimality criterion" that the Kalman filter minimizes. "Mathematical tractability is a crucial consideration for an optimality criterion, but not for a measure for performance evaluation. As a ruler, it is much more important for a measure for performance evaluation to be impartial, free of distortion, and with a clear interpretation, among other things"[32].

Example 4.3. Using the same parameters as in previous examples (4.2) and (4.1) and setting $K = 100$ the finite sample equivalent of Figure 4.3 is displayed in Figure 4.4. The added process and sensor noise is sampled with aforementioned covariances as is the initial condition $\vec{x}(0)$ for true state with mean

$$\vec{\bar{x}}(0) = \begin{bmatrix} 0 \\ 3 \end{bmatrix} \tag{4.22}$$

and covariance $P(0)$ from (4.16).

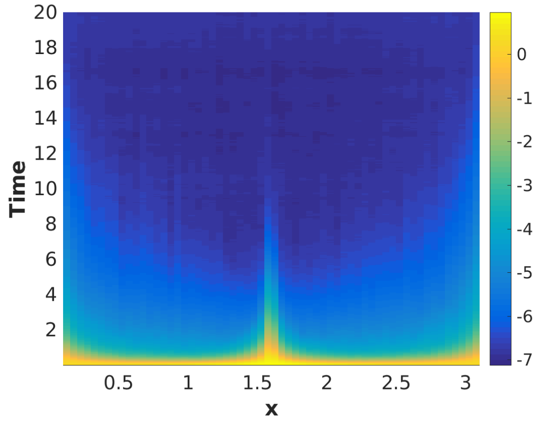

Figure 4.4: The log of the square of RMSE (4.21) is plotted for various sensor locations. The expression (4.20) approximates (4.19) well.

A strong assumption for the minimization problem (4.15) is the restriction that both the initial covariance and estimate are correct:

$$P(0) = \mathrm{E}\left[\left(x(0) - \hat{x}(0)\right)\left(x(0) - \hat{x}(0)\right)^T \right] \tag{4.23}$$

$$\hat{x}(0) = \mathrm{E}\left[x(0)\right]. \tag{4.24}$$

Wrong initial conditions, where (4.23 - 4.24) are not satisfied will not be considered except in the following example because it adds an additional degree of freedom to the investigation.

Example 4.4. Using the same parameters as in example (4.3) except the initial estimate is set incorrectly - representing a wrong initial condition guess. The true state will still evolve with

$$\vec{x}(0) = \begin{bmatrix} a_1(0) \\ a_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \end{bmatrix} \tag{4.25}$$

while the Kalman filter's estimate is initialized as

$$\vec{\hat{x}}(0) = \begin{bmatrix} \hat{a}_1(0) \\ \hat{a}_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.26}$$

with the same initial covariance. The sensor will be placed in the middle of the domain $x_0 = \frac{\pi}{2}$, the node of the mode 2 wave. As expected, when compared to a correct initial condition guess the incorrect guess will perform poorly as seen in Figure 4.5.

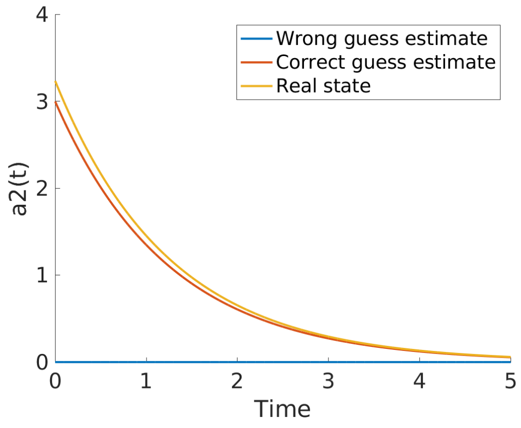

Figure 4.5: The incorrect case initializes estimate of a_2 as zero. Since the sensor is located at the node of second mode, it cannot pick up its amplitude. Therefore, the incorrect estimate remains unchanged and only converges to the real state once the real state has diffused to zero. The wrong guess estimate's a_2 value is on on top of the Time axis.

Figure 4.5 shows the importance of a correct initial guess. In Figure 4.6 the square RMSE and Trace($P(t)$) are plotted to show how the Kalman trace based cost function is no longer reflected in actual simulation once the initial condition is guessed incorrectly.

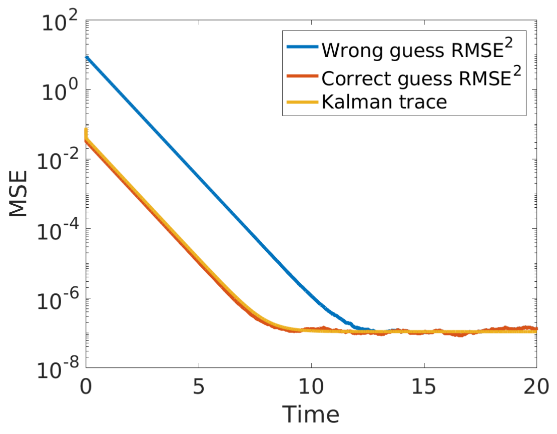

Figure 4.6: The finite sample approximation of Kalman trace is no longer close to the real trace when the initial guess is incorrect.

The previous example (4.4), although contrived, demonstrates the importance of the restriction on the initial estimate - having to match the mean of the real state at time zero. The results are not a surprise as the incorrect initial condition was a deviation from Kalman definitions in Definition (3.2) and (3.1). However, the RMSE used in example (4.4) is a useful tool and will be used for nonlinear systems where the guarantee of minimizing the mean square error is no longer present.

Chapter 5

Nonlinear sensor placement

The previous chapter served as an introduction to Kalman based sensor placement techniques for linear systems. In this chapter, the transient approach is extended to nonlinear systems. By definition, for such systems, the regular Kalman filter can no longer be used and a common solution is to instead use the Extended Kalman filter introduced in chapter 3. For the EKF, the covariance now depends on the computed estimate, as for example in the discrete case introduced in Definition 3.1 the predict phase is

$$\vec{\hat{x}}_k^- = f(\vec{\hat{x}}_{k-1}^+) \tag{5.1}$$

$$\hat{P}_k^- = J\hat{P}_k^+ J^T + Q, \tag{5.2}$$

where J is the Jacobian of f evaluated at the estimate $\vec{\hat{x}}_{k-1}^+$. In return, in the update phase, the estimate

$$\vec{\hat{x}}_k^+ = \vec{\hat{x}}_k^- + K_k \left(\vec{y}_k - C\vec{\hat{x}}_k^- \right) \tag{5.3}$$

is a function of the measurement and thus depends on sensor and process noise. The previous equations referred to the discrete Kalman filter, but the same holds for the continuous-discrete Extended Kalman filter. The cost functions will be expressed with respect to the continuous-discrete formulation. Since the noise is sampled, then like the RMSE in (4.21) the Kalman covariances will be averaged to account for the variance as part of the sensor placement algorithm. The original transient cost function for the continuous-discrete case is

$$J_{[t_s,t_f]} = \frac{1}{t_f - t_s} \int_{t_s}^{t_f} \text{Trace}\left(P(t)\right) dt \tag{5.4}$$

and is now modified to

$$J^N_{[t_s,t_f]} = \frac{1}{t_f - t_s}\left(\sum_{n=1}^{N}\int_{t_s}^{t_f}\text{Trace}\left(P_n(t)\right)dt\right), \tag{5.5}$$

where n stands for the n^{th} run of N total Monte-Carlo simulations (and not the discrete Kalman filter covariance $\hat{P}_n^+$ at n^{th} step). An analogous cost function can be formulated for the discrete Extended Kalman filter. For the n^{th} simulation, the initial condition of the true state is sampled from the specified initial covariance $P(0)$ and mean $\vec{\hat{x}}(0)$ and solved for with the specified process noise. The measurements are then generated using the observation matrix C with added sensor noise. The N measurements are then used as input for the EKF that is initialized with $P(0)$ and $\vec{\hat{x}}(0)$. The output of the EKF is N sets of estimates and covariances. The covariances are averaged using (5.5). By varying the sensor locations x_0 from a finite candidate set X and therefore varying the observation matrix C the cost (5.5) can be minimized to find the optimal sensor location. The above is summarized as

$$\begin{gathered}
\min J^N_{[t_s,t_f]} \\
\text{subject to} \\
P(0) = \mathrm{E}[\left(\vec{x}(0) - \vec{\hat{x}}(0)\right)\left(\vec{x}(0) - \vec{\hat{x}}(0)\right)^T] \\
\vec{\hat{x}}(0) = \mathrm{E}[\vec{x}(0)] \\
\vec{\hat{x}}(t_{k-1}) = \vec{\hat{x}}^+_{k-1} \\
\hat{P}(t_{k-1}) = P^+_{k-1} \\
\frac{d}{dt}\left(\vec{\hat{x}}(t)\right) = A\vec{\hat{x}}(t) \\
\frac{d}{dt}\left(P(t)\right) = AP(t) + P(t)A^T + Q \\
\vec{\hat{x}}^-_k = \vec{\hat{x}}(t_k) \\
P^-_k = P(t_k) \\
C = f(x_0) \\
x_0 \in X
\end{gathered} \tag{5.6}$$

The observation matrix C is a function of the sensor location. The first section in this chapter will approach the problem with a modal truncation, and the second with a spatial discretization.

5.1 Modal Discretization

The modal truncation introduced in (2.29) of chapter 2 can be expressed as

$$\frac{d}{dt}(\vec{a}) = f(\vec{a}) + \vec{n}(t), \tag{5.7}$$

where $\vec{a} \in \mathbb{R}^N$ represents the mode coefficients and $\vec{n}(t) \in \mathbb{R}^N$ is the process noise term. For modal discretization, the state $\vec{x}$ will be referred to as $\vec{a}$. For illustration, N will be set to 10. It will be shown that the sensor placement problem is initial condition dependent. Two initial conditions will be considered:

$$u_1(x, t=0) = \sin\left(\left(\frac{\pi}{L}\right)x\right) + \frac{1}{2}\sin\left(\left(\frac{2\pi}{L}\right)x\right) + \frac{1}{2}\sin\left(\left(\frac{3\pi}{L}\right)\right) \tag{5.8}$$

$$u_2(x, t=0) = \sin\left(\left(\frac{\pi}{L}\right)x\right) \tag{5.9}$$

for Burgers defined on the domain $[0, \pi]$. The initial conditions are plotted in Figure 5.1.

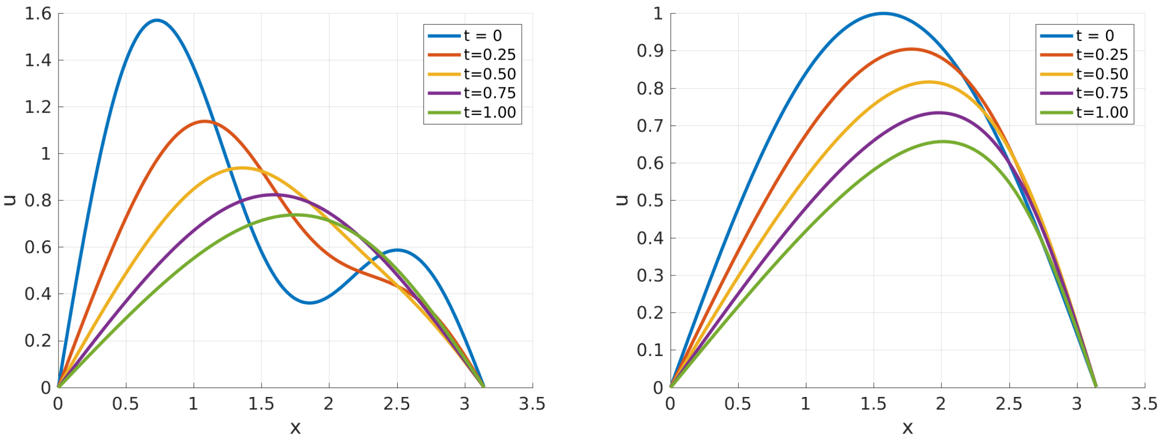

Figure 5.1: From left to right: initial condition u_1 and states at later times and u_2 and states at later times as defined in (5.8-5.9).

The process noise covariance is set to

$$Q = 10^{-12}\text{diag}\left(1, \frac{1}{2^2}, \frac{1}{3^2}, \ldots, \frac{1}{6^2}, 0, 0, 0\right), \tag{5.10}$$

where diag is a diagonal matrix with the argument as the diagonal. The sensor noise is

$$R = 10^{-8} \tag{5.11}$$

with measurements sampled every 10^{-6} time units. The initial variance is set to

$$P(0) = 10^{-4}\text{diag}\left(1, \frac{1}{2^2}, \frac{1}{3^2}, 0, \dots, 0\right). \tag{5.12}$$

In the nonlinear case, the true state's initial condition will be sampled using (5.12) with mean (5.8) or (5.9) to justify setting the EKF with initial covariance (5.12). A total of 30 samples will be taken for each initial condition. For each sample, noisy measurements will be taken for each candidate sensor location. The candidate sensors locations are

$$0.1, 0.105, \dots, 3.1 \tag{5.13}$$

for a total of 61 locations. The sample count and sensor count will lead to running the filter for a total of $30 \times 61 = 1830$ runs of the EKF with a total of 10^6 measurements for each run.

In Figure 5.2 the trace is shown for initial condition (5.8) and in Figure 5.3 for initial condition (5.9).

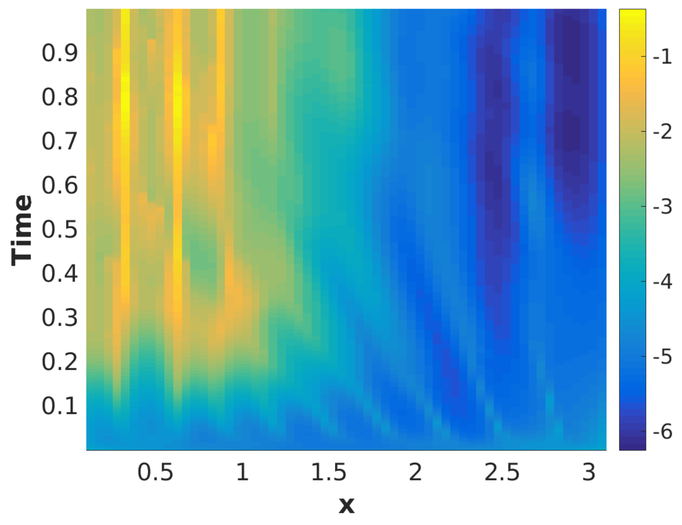

Figure 5.2: The trace of the Kalman trace is computed over time 0 to 1 for various sensor locations on the x axis. The log of the trace is plotted for initial condition u_1.

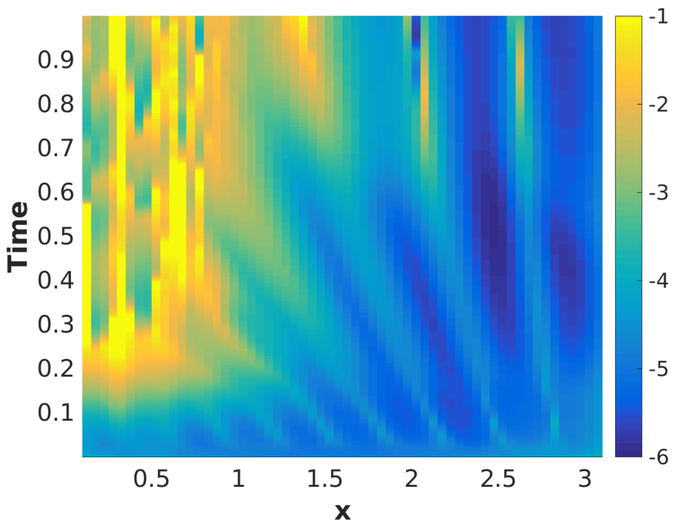

Figure 5.3: The trace of the Kalman trace is computed over time 0 to 1 for various sensor locations on the x axis. The log of the trace is plotted for initial condition u_2.

The cost is minimized at different locations as shown in Figure 5.4.

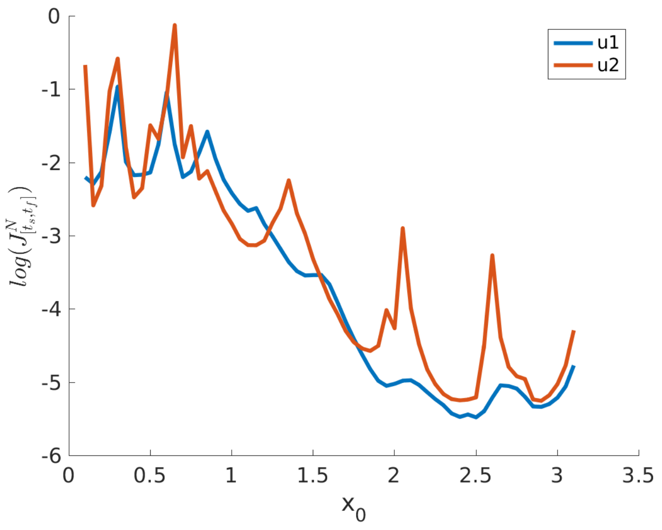

Figure 5.4: The log of $J^N_{[t_s,t_f]}$ is plotted for both initial conditions. The cost function is minimized at different locations: for u_1 it is at $x = 2.5$ with a cost of 3.3309×10^{-6} while for u_2 it is at $x = 2.9$ with a cost of 5.6026×10^{-6}.

5.2 Spatial Discretization

For the next investigation, the discretization is changed to the finite difference scheme (2.38) introduced in chapter 2. A common approach for sensor placement studies involves first approximating a PDE and then running a sensor placement algorithm on the finite dimensional system. This will not be done here and instead a spatial discretization is used where the state is a vector representing u at the grid points at time step n with transition between timesteps given by

$$\vec{U}^{n+1} = f\left(\vec{U}^n\right), \tag{5.14}$$

where f in the case of Burgers is nonlinear. Gaussian, zero mean noise is added to (5.14):

$$\vec{U}^{n+1} = f\left(\vec{U}^n\right) + \vec{w}_n. \tag{5.15}$$

Convergence to the solution to the PDE and of the Kalman filter will not be an issue of concern and system (5.15) with its noise will be treated as the true representation of some system. This means that in the case that f is linear, the transient and steady state approach will lead to the optimal sensor placement with respect to the problem of minimizing the mean square error for that system. In the nonlinear case where the Kalman covariance P is propagated through the Jacobian (an approximation), the trace, or equivalently the mean square error, will have to be validated through multiple simulations. For this investigation, a single initial condition is selected

$$u_0(x, t = 0) = \sin\left(\left(\frac{\pi}{L}\right) x\right) + \sin\left(\left(\frac{2\pi}{L}\right) x\right) + 0.5 \tag{5.16}$$

and is plotted along with space time plot of its evolution (i.e. the trajectory of the system) in Figure 5.5. The grid count or equivalently the state count is set to $M = 100$. The process noise is selected such that

$$Q = 0.001^2 I_M, \vec{w}_n \sim \mathcal{N}(0, Q) \tag{5.17}$$

and initial variance of $\vec{U}^0$ is

$$P(0) = 0.01^2 I_M, \tag{5.18}$$

where I_M is the $M \times M$ identity matrix. N - simulation count - is set to 150.

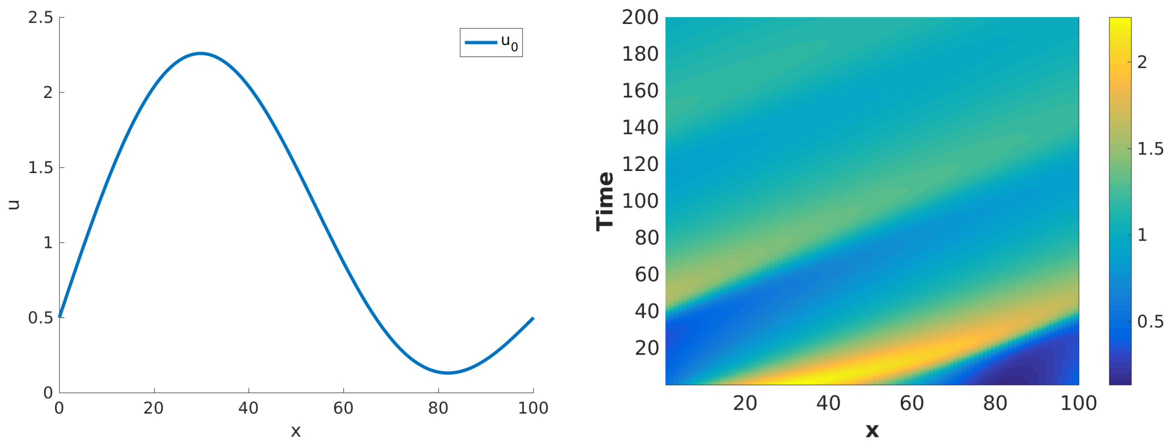

Figure 5.5: Left panel: The initial condition u_0 from (5.16) of the periodic domain and average trajectory is plotted. Right panel: A space time plot of the evolution of the initial condition (i.e. the trajectory of the system). The colour and associated colour bar show the magnitude of the state at a given time - the brighter the colour the higher the magnitude. By the end of simulation time (T = 200) the waveform has mostly diffused out.

One possible physical intuition for sensor placement is that placing a sensor where the waveform has the highest values would be more effective than a location with lower values. This is because the relative error between the noisy sensor measurement and wave height would be smaller in the former location and therefore the sensor would be able to pick up variations much more easily. Using a different color map than for the state trajectory in Figure 5.5, the relative error is plotted between the sensor measured value at a given grid point and the actual value at the grid point for all simulation times in Figure 5.6.

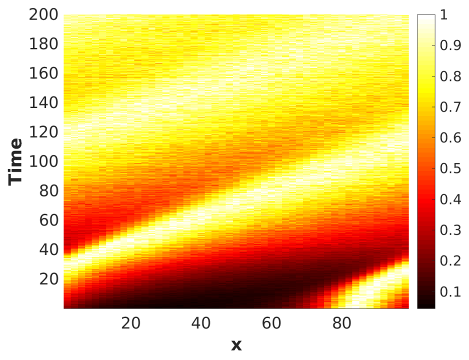

Figure 5.6: The average relative error between noisy measurement and true state value at each location is plotted and normalized with respect to the largest relative error at a given time. The noise is zero mean with constant variance 0.1^2. The higher the ratio, the smaller the u_j^n at that location. The initial condition is (5.16).

Combining the relative error Figure 5.6 and state trajectory Figure 5.5 together into Figure 5.7 for comparison, it is confirmed that locations where the state has a high value correspond to locations where the relative error is low. Same shaded regions appear in both figures to facilitate a comparison.

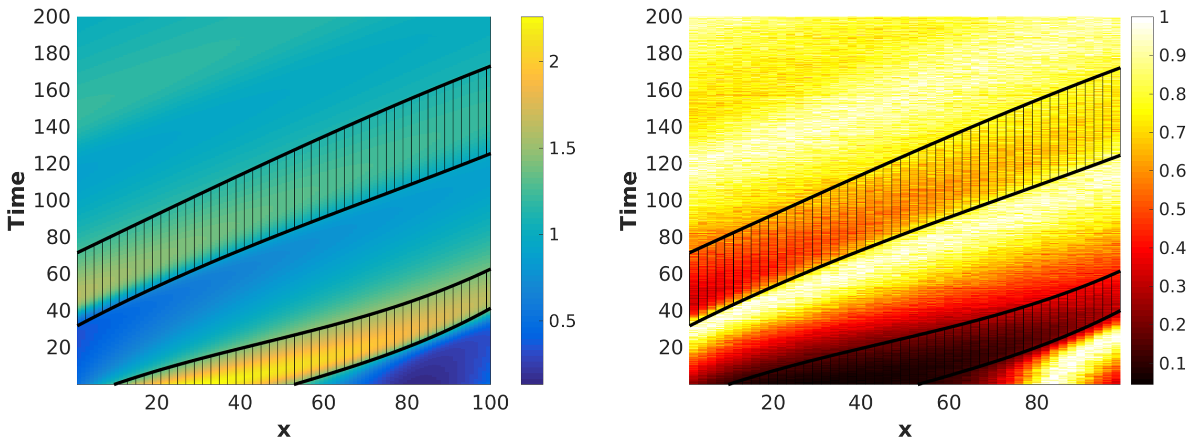

Figure 5.7: On the left is the average state trajectory with shaded region showing area where the state is relatively high compared to other locations at the same time. On the right, the normalized relative error between noisy sensor measurement and true state from Figure 5.6. The locations where the state has a high value is where the relative error is the lowest - this is not a surprise and is expected. The initial condition is (5.16).

Every second grid point will be considered as a candidate sensor location where the domain is $x \in (0, 100)$:

$$x_0 \in \{1, 3, \ldots, 99\}. \tag{5.19}$$

The average computed Kalman trace is plotted in Figure 5.8 with the same shading used as in previous figures.

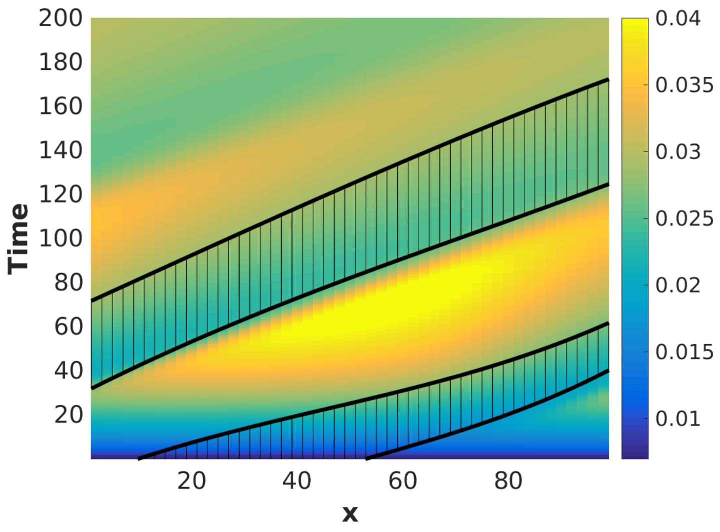

Figure 5.8: Kalman trace is computed over time 0 to 200 for various sensor locations on the x axis. The color represents the magnitude of the trace at a specific time and location.

The shaded region overlay in the state trajectory from Figure 5.7 matches where the state is the highest in value at a given time and from Figure 5.6 where the sensor measured state relative error is the lowest is also the region where the the Kalman trace is lowest in value. This matches the hypothesis made earlier, although is not a definite claim.

From Figure 5.8, the highest trace occurs during $t = 71.67$ at $x_0 = 57$ with $\text{Trace}\,(P(t)) = 0.0206$ while at the same time, the minimum trace of 0.0116 is achieved at $x_0 = 31$. It is challenging to have a physical interpretation of the trace so sample estimates based on the two sensor locations are plotted alongside the true state in Figure 5.9. The estimate at $x_0 = 57$ does not approximate the true state as well as at $x_0 = 31$ on domain $[20, 50]$. By plotting the individual trace components corresponding to the grid points in Figure 5.10 the $x_0 = 57$ location has a higher expected error rate in the $[20, 50]$ subdomain.

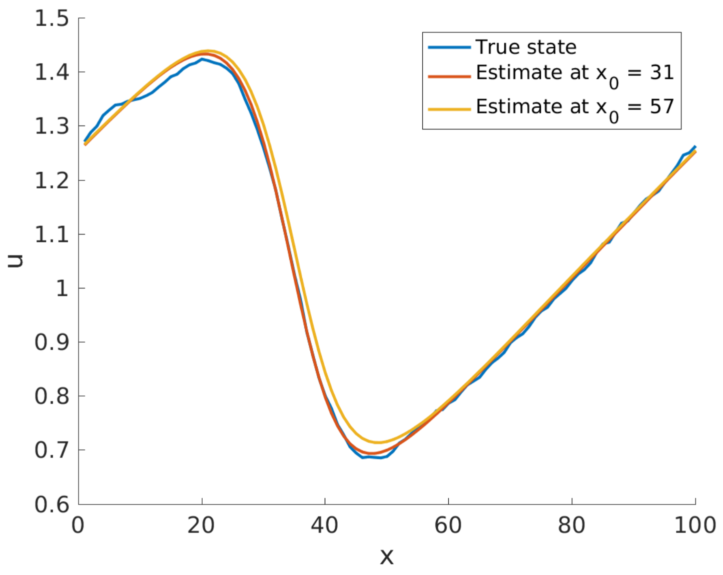

Figure 5.9: The true state is plotted along with estimate with sensor at $x_0 = 31$ and estimate with sensor at $x_0 = 57$, all at $t = 71.67$. Note: small gap for x in range $[20, 50]$ between estimate with sensor at $x = 57$ and true state signifying that estimator with sensor location at $x_0 = 31$ is performing better. The initial condition is (5.16).

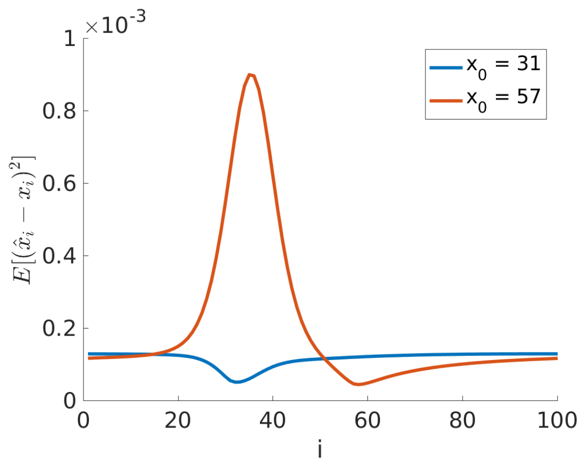

Figure 5.10: The diagonal entries of the Kalman covariance are plotted for two different sensor locations at $t = 71.67$. The i^{th} entry corresponds to the $x = i$ grid point. The true state is expected to be much better approximated by the sensor at $x_0 = 31$ in the domain $[20, 50]$. The initial condition is (5.16).

Despite Figure 5.9 and 5.10 providing more information as to why at a particular time one location performs better than the other, it is difficult to do this for all time and locations and summarize results effectively which is why Figure 5.8 is used instead. This prevents one from knowing exactly why a trace is lower at one compared to another location. For physical interpretations this can be a limiting factor. By strictly looking at the trace covariance, in this context, it is implied that only the cumulative error matters, and not some physical property. The less the expected error, the better. In the following Figure 5.11, the sampled equivalent of Figure 5.8 is plotted. The sampled plot is similar to that of the Kalman trace in shape but differs in magnitude suggesting slight divergence.

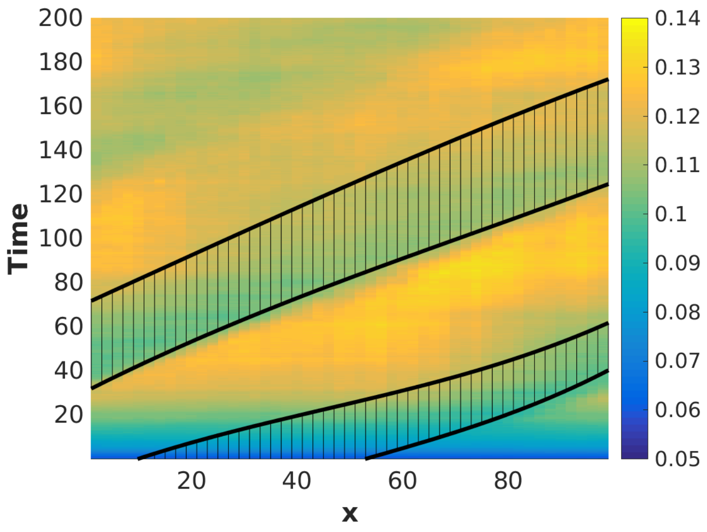

Figure 5.11: The squared RMSE for all 150 simulation runs. The plot is very similar in shape to the trace plot in Figure 5.8. However, the scale is not the same.

The similarity of the plots leads to the conclusion that the Kalman trace data approximates the sensor placement results reasonably well. With this assurance, the sensor placement problem (5.6) for nonlinear systems is conducted. For the time averaged Kalman cost function, t_s will be set to zero and t_f to final simulation time of 200. The results are plotted in Figure 5.12. The location that minimizes the problem is at $x_0 = 5$ with an average trace of 0.013538.

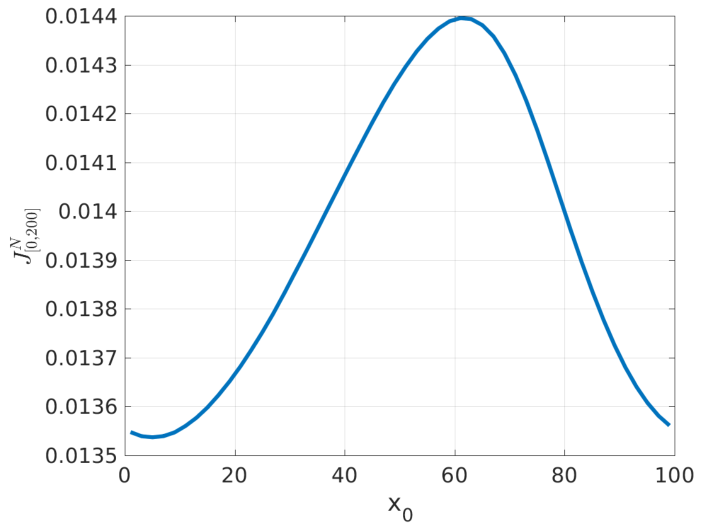

Figure 5.12: All possible $J^1_{[0,200]}50$ scores given the sensor candidates. The location that satisfies (5.6) is at $x_0 = 5$ with a score of 0.013538.

The chosen t_f and t_s are ad hoc in nature, but can be specified in a certain physical context. Since Figures 5.11 and 5.8 are similar, a natural question to ask is to why not simply use the RMSE as a cost function instead. For each run of the EKF one produces both the sample estimate $\hat{\vec{x}}$ and $P(t)$ which are used respectively for the average based costs RSME and $J^N_{[t_s,t_f]}$. If one runs N simulations then one has the choice of using the RMSE or trace of the covariance as costs. The reason to use $J^N_{[t_s,t_f]}$ is because of its rate of convergence, requiring a smaller N and therefore less computation. In Figure 5.13 the log of relative error is plotted between the average Kalman trace with $N = 150$ and N set to $N = 30, 60, 90, 120$. The relative error at $N = 30$ is low suggesting that lower N can be used for solving the sensor placement problem. For RMSE, the convergence is much slower and noisier. In addition, as a preliminary test, the unscented Kalman Filter was run in place of the EKF and the computed trace was found to have a relative error of maximum 10^{-4} and general filter performance was the same.

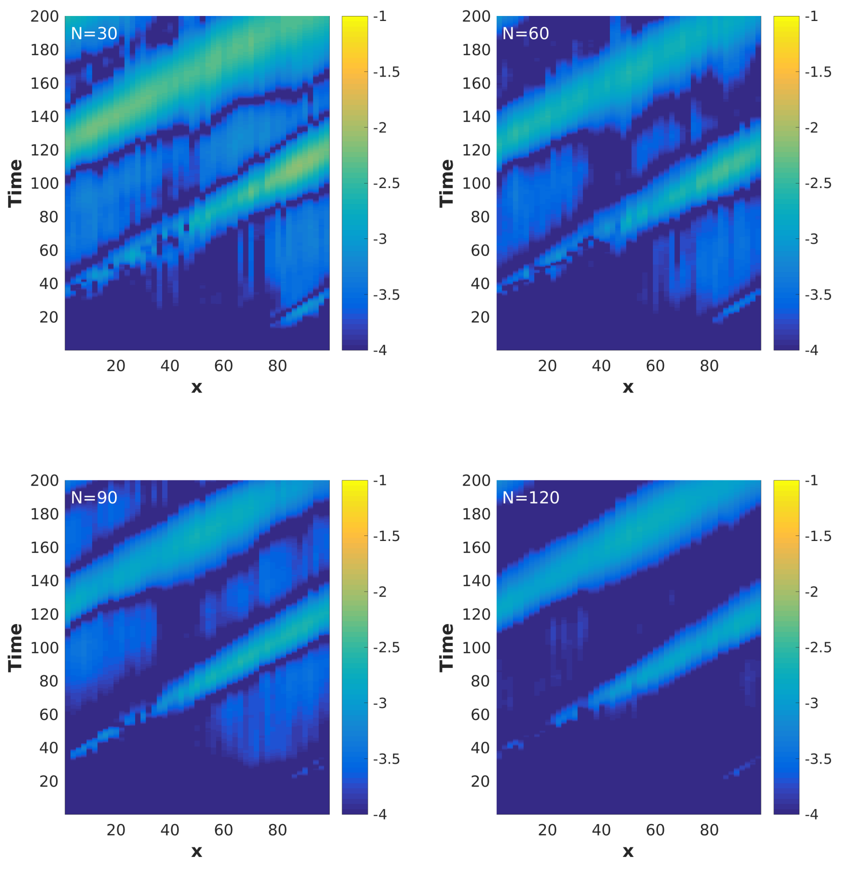

Figure 5.13: Relative error log plot between average Kalman trace using $N = 150$ as reference solution and varying N. The relative error stays low throughout.

Another question concerns scalability of the specific algorithm presented in this thesis. What if systems with larger state space are to be investigated? The Matlab's version of the EKF implementation averaged computation time for varying M is included below in Figure 5.14. The brute force algorithm is not scalable to larger state count and multiple sensor count.

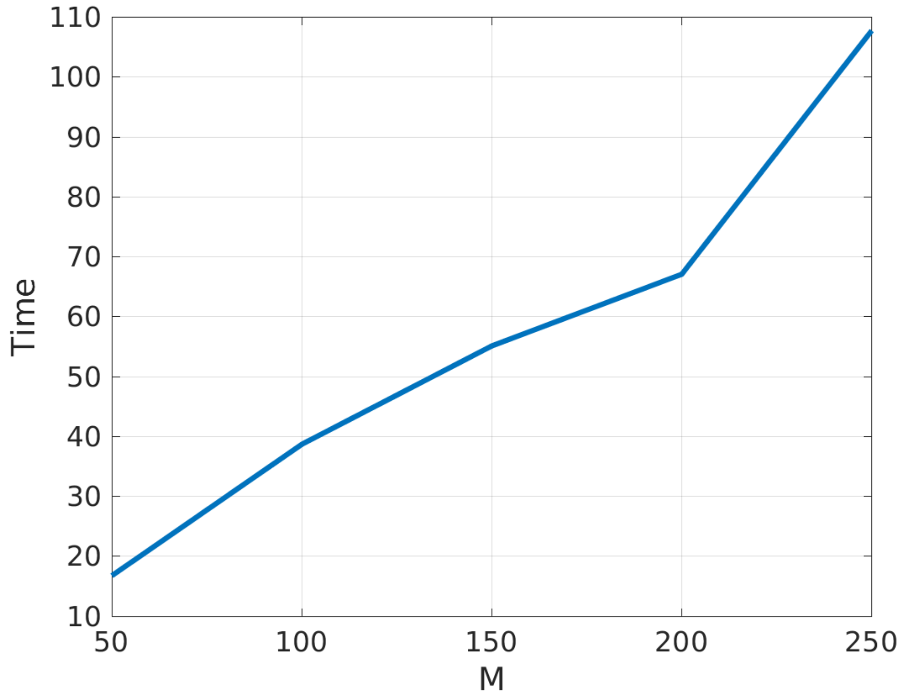

Figure 5.14: For state space size M and 20000 measurements the average computation time in seconds is shown for Matlab implementation for the EKF. For the investigation the sample size is $N = 150$ with 50 candidate locations meaning that for $M = 50$ the computation time on a single core is approximately 1.5 days while for $M = 250$ is 9.5 days. If sensor count is set to 2 (choosing an optimal pair of sensors) then $\binom{50}{2}$ total candidate pairs need to be investigated instead of 50 - a significant computational cost.

To investigate this sensor placement approach presented in this thesis further, non brute force type solutions to the optimization problem should be considered or more efficient Riccati solvers.

Chapter 6

Conclusion

In chapter 4 it was found that for linear systems, steady state based optimal sensor placement results may differ from those based on the transient. Examples in the introduction to sensor placement provided a clear explanation of the approach as well as some potential issues as outlined in example 4.4 where the initial estimate was set to an incorrect value. The results chapter explored the extension of the transient approach to a nonlinear system, namely Burgers equation. Different initial conditions are found to lead to different sensor placements.

There are various concerns that cast a shadow on the transient approach. Discussing them is appropriate for the consideration of future potential investigations on this topic. Parameters such as process, sensor noise covariances and the diffusivity coefficient of Burgers equation were set in an ad hoc manner. This created an additional degree of freedom when considering how to approximate the model. The Euler-Maruyama method was used for the modal discretization, but required a very small time step and low process noise covariance. When considering a modal discretization for shallow water, this scheme had extremely poor performance, preventing both an effective sensor placement strategy and the study of convergence for Burgers equation. Implicit methods exist such as the θ-Euler-Maruyama method [33], but were not investigated in depth. Perhaps an approach of generating the true state $\vec{x}(t)$ via a spatial discretization and then running the Kalman filter with a modal based state is a good middle ground similar to what is done in [21] for ease of numerics and to facilitate a convergence study.

One may reasonably assume that these parameter/discretization based issues will be resolved when a specific system is selected and a good numerical scheme chosen for investigation. However, for the transient sensor placement approach, the value of a few parameters remain unresolved: initial covariance and estimate. In the linear case, the trace of the covariance is minimized if the initial estimate is the expected value of the true state and the initial covariance is the error covariance between the estimate and true state, all at time zero. Although mathematically these concepts are clear and well defined with lots of theory behind them, it is hard to extend them to a clear physical context (which ideally is the end goal of a sensor placement investigation). How would one know what the expected value of the initial state is? Example 4.3 showed that incorrectly setting the initial estimate can have significant consequences.

Setting the initial estimate of the filter is related to what initial condition of the true state is being studied in general. For nonlinear systems the computed trace is initial condition dependent as shown in modal example of chapter 5. This means that coming up with a global sensor placement is difficult without being more specific in what initial conditions are studied. Perhaps initial conditions are too specific and another cost function that is independent of it should be studied.

Possible divergence of the Kalman covariance must also be reviewed. Divergence occurs when the error between the computed estimate and true state is outside the range predicted by the covariance matrix P. Computational issues, incorrect parameters or the propagation of the covariance through a first-order linearization within the EKF can lead to the divergence of the filter. Since the cost function for sensor placement is based on P, divergence must be investigated more thoroughly to ensure more confidence in results. Other filters such as the unscented Kalman filter or the ensemble Kalman filter are known to not diverge when the EKF does and the ensemble can scale to a larger state count. This suggests that other algorithms could be investigated for sensor placement. The divergence as well as potential of using other, better filters makes one reconsider the word 'optimal' that is often associated with the phrase 'sensor placement'. In the case of this thesis, although the sensor is 'optimal' with respect to the function to be minimized, the use of the word becomes questionable when considering the global context.

The highlighted issues in past few paragraphs need to be considered and accounted for to have a more thorough and conclusive sensor placement investigation.

References

[1] Brian D.O. Anderson and John B. Moore. *Optimal Filtering*. Courier Corporation, 2012.

[2] Dale A. Anderson, John C. Tannehill, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. Taylor & Francis, 1997.

[3] M. Antuono et al. “Two-dimensional modal method for shallow-water sloshing in rectangular basins”. In: *Journal of Fluid Mechanics* 700 (2012), pp. 419–440.

[4] Mark Asch, Marc Bocquet, and Maëlle Nodet. *Data Assimilation: Methods, Algorithms, and Applications*. Vol. 11. SIAM, 2016.

[5] Michael Athans and Edison Tse. “A direct derivation of the optimal linear filter using the maximum principle”. In: *IEEE Transactions on Automatic Control* 12.6 (1967), pp. 690–698.

[6] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. John Wiley & Sons, 2004.

[7] Robert Grover Brown, Patrick Y.C. Hwang, et al. *Introduction to Random Signals and Applied Kalman Filtering*. Vol. 3. Wiley New York, 1992.

[8] Richard S. Bucy and Peter D. Joseph. *Filtering for Stocahstic Processes With Applications to Guidance*. Vol. 326. Chelsea Publishing Company, 1987.

[9] Ruth F. Curtain and Akira Ichikawa. “Optimal location of sensors for filtering for distributed systems”. In: *Distributed Parameter Systems: Modelling and Identification*. Springer, 1978, pp. 236–255.

[10] R. Fitzgerald. “Divergence of the Kalman filter”. In: *IEEE Transactions on Automatic Control* 16.6 (1971), pp. 736–747.

[11] Paul Frogerais, Jean-Jacques Bellanger, and Lotfi Senhadji. “Various ways to compute the continuous-discrete extended Kalman filter”. In: *IEEE Transactions on Automatic Control* 57.4 (2012), p. 1000.

[12] Alfredo Germani, L. Jetto, and M. Piccioni. “Galerkin approximation for optimal linear filtering of infinite-dimensional linear systems”. In: *SIAM journal on control and optimization* 26.6 (1988), pp. 1287–1305.

[13] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Vol. 3. JHU press, 1989.

[14] Mohinder S. Grewal. *Kalman Filtering*. Springer, 2011.

[15] Andrew C. Harvey. *Forecasting, Structural Time Series Models and the Kalman filter*. Cambridge university press, 1990.

[16] Simon Haykin. *Kalman Filtering and Neural Networks*. Vol. 47. John Wiley & Sons, 2004.

[17] John Bagterp Jorgensen et al. “A computationally efficient and robust implementation of the continuous-discrete extended Kalman filter”. In: *American Control Conference, 2007. ACC'07*. IEEE. 2007, pp. 3706–3712.

[18] Thomas Kailath, Ali H. Sayed, and Babak Hassibi. *Linear estimation*. EPFL-BOOK-233814. Prentice Hall, 2000.

[19] Rudolph E. Kalman and Richard S. Bucy. “New results in linear filtering and prediction theory”. In: *Journal of basic engineering* 83.1 (1961), pp. 95–108.

[20] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[21] Wei Kang and Liang Xu. “Optimal placement of mobile sensors for data assimilations”. In: *Tellus A: Dynamic Meteorology and Oceanography* 64.1 (2012), p. 17133.

[22] Khan, Tawsif. “Optimal Sensor Location for the Estimation of a Linear Dispersive Wave Equation”. In: (2015). URL: http://hdl.handle.net/10012/9507.

[23] Tawsif Khan, Kirsten Morris, and Marek Stastna. “Computation of the Optimal Sensor Location for the Estimation of an 1-D Linear Dispersive Wave Equation”. In: *2015 American Control Conference (ACC)*. IEEE. 2015, pp. 5270–5275.

[24] Carlos S. Kubrusly and Helios Malebranche. “Sensors and controllers location in distributed systems-A survey”. In: *Automatica* 21.2 (1985), pp. 117–128.

[25] Maria V. Kulikova and Gennady Yu Kulikov. “Square-root accurate continuous-discrete extended Kalman filter for target tracking”. In: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, pp. 7785–7790.

[26] Sudarshan Kumar and J. Seinfeld. “Optimal location of measurements for distributed parameter estimation”. In: *IEEE Transactions on Automatic Control* 23.4 (1978), pp. 690–698.

[27] R.R. Labbe. *Kalman and Bayesian Filters in Python*. 2014.

[28] Roger Labbe. *Kalman and Bayesian Filters in Python*. URL: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python.

[29] Randall J. LeVeque. *Finite Fifference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Vol. 98. Siam, 2007.

[30] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Vol. 31. Cambridge university press, 2002.

[31] Randall J. LeVeque. *Numerical Methods for Conservation Laws*. Vol. 132. Springer, 1992.

[32] X. Rong Li and Zhanlue Zhao. “Measures of performance for evaluation of estimators and filters”. In: *Signal and Data Processing of Small Targets 2001*. Vol. 4473. International Society for Optics and Photonics. 2001, pp. 530–542.

[33] Gabriel J. Lord, Catherine E. Powell, and Tony Shardlow. *An Introduction to Computational Stochastic PDEs*. 50. Cambridge University Press, 2014.

[34] Leonard A. McGee and Stanley F. Schmidt. “Discovery of the Kalman filter as a practical tool for aerospace and industry”. In: (1985).

[35] William H. Press et al. “Numerical Recipes in C++”. In: *The art of scientific computing* 2 (1992), p. 1002.

[36] Dan Simon. *Optimal State Estimation: Kalman, H infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[37] M.W.A. Smith and A.P. Roberts. “An exact equivalence between the discrete-and continuous-time formulations of the Kalman filter”. In: *Mathematics and COmputers in Simulation* 20.2 (1978), pp. 102–109.

[38] Marc Van De Wal and Bram De Jager. “A review of methods for input/output selection”. In: *Automatica* 37.4 (2001), pp. 487–510.

[39] Michel Verhaegen and Paul Van Dooren. “Numerical aspects of different Kalman filter implementations”. In: *IEEE Transactions on Automatic Control* 31.10 (1986), pp. 907–917.

[40] Gerald Beresford Whitham. *Linear and Nonlinear Waves*. 1974.

[41] Xueran Wu, Birgit Jacob, and Hendrik Elbern. “Optimal control and observation locations for time-varying systems on a finite-time horizon”. In: *SIAM Journal on Control and Optimization* 54.1 (2016), pp. 291–316.

[42] Lihua Xie, Dan Popa, and Frank L. Lewis. *Optimal and Robust Estimation: With an Introduction to Stochastic Control Theory.* CRC press, 2007.

[43] Thomas K. Yu and John H. Seinfeld. “Observability and optimal measurement location in linear distributed parameter systems”. In: *International journal of control* 18.4 (1973), pp. 785–799.

[44] Minxin Zhang and Kirsten Morris. “Sensor choice for minimum error variance estimation”. In: *IEEE Transactions on Automatic Control* 63.2 (2018), pp. 315–330.

APPENDICES

Appendix A

QR for Kalman Update phase

Consider $R \in \mathbb{R}^{m\text{x}m}$, $C \in \mathbb{R}^{m\text{x}n}$, $\left(P_k^-\right)^{\frac{1}{2}} \in \mathbb{R}^{n\text{x}n}$ that together make up the matrix

$$\begin{bmatrix} R^{\frac{1}{2}} & C\left(P_k^-\right)^{\frac{1}{2}} \\ 0 & \left(P_k^-\right)^{\frac{1}{2}} \end{bmatrix} \in \mathbb{R}^{(m+n)\text{x}(m+n)}. \tag{A.1}$$

The matrices are originally part of the Kalman filter's update phase

$$K_k = \hat{P}_k^- C^T \left(C P_k^- C^T + R\right)^{-1} \tag{A.2}$$
$$\hat{x}_k^+ = \hat{x}_k^- + K_k\left(y_k - C\hat{x}_k^-\right) \tag{A.3}$$
$$\hat{P}_k^+ = (I - K_k C)\hat{P}_k^-. \tag{A.4}$$

as defined in definition (3.2). In (3.109) it was stated that (A.1) can be triangulized as follows

$$\begin{bmatrix} R^{\frac{1}{2}} & C\left(P_k^-\right)^{\frac{1}{2}} \\ 0 & \left(P_k^-\right)^{\frac{1}{2}} \end{bmatrix} \Theta = \begin{bmatrix} X & 0 \\ Y & Z \end{bmatrix} \tag{A.5}$$

where $\Theta \in \mathbb{R}^{(m+n)\text{x}(m+n)}$ is unitary, $X \in \mathbb{R}^{m\text{x}m}$, $Y \in \mathbb{R}^{n\text{x}m}$ and $Z \in \mathbb{R}^{n\text{x}n}$ such that

$$\left(\begin{bmatrix} R^{\frac{1}{2}} & C\left(P_k^-\right)^{\frac{1}{2}} \\ 0 & \left(P_k^-\right)^{\frac{1}{2}} \end{bmatrix} \Theta\right)\left(\begin{bmatrix} R^{\frac{1}{2}} & C\left(P_k^-\right)^{\frac{1}{2}} \\ 0 & \left(P_k^-\right)^{\frac{1}{2}} \end{bmatrix} \Theta\right)^* = \begin{bmatrix} R + CP_k^- C^T & C\left(P_k^-\right) \\ \left(P_k^-\right)^T C^T & P_k^- \end{bmatrix} \tag{A.6}$$

$$\left(\begin{bmatrix} X & 0 \\ Y & Z \end{bmatrix}\right)\left(\begin{bmatrix} X & 0 \\ Y & Z \end{bmatrix}\right)^* = \begin{bmatrix} XX^* & XY^* \\ YX^* & YY^* + ZZ^* \end{bmatrix} \tag{A.7}$$

$$\begin{bmatrix} R + CP_k^- C^T & C\left(P_k^-\right) \\ \left(P_k^-\right)^T C^T & P_k^- \end{bmatrix} = \begin{bmatrix} XX^* & XY^* \\ YX^* & YY^* + ZZ^* \end{bmatrix}. \tag{A.8}$$

From (A.8) the following is true

$$XX^* = R + CP_k^- C^T \tag{A.9}$$
$$YY^* + ZZ^* = P_k^- \tag{A.10}$$
$$XY^* = CP_k^- \tag{A.11}$$
$$YX^* = \left(P_k^-\right)^T C^T. \tag{A.12}$$

Take (A.10) and expand to obtain

$$YY^* + ZZ^* = P_k^- \tag{A.13}$$
$$ZZ^* = P_k^- - YY^* \tag{A.14}$$
$$ZZ^* = P_k^- - Y\left(X^*\left(X^*\right)^{-1}\right)\left(X^{-1}X\right)Y^* \tag{A.15}$$
$$ZZ^* = P_k^- - YX^*\left(\left(X^*\right)^{-1}X^{-1}\right)XY^* \tag{A.16}$$
$$ZZ^* = P_k^- - YX^*\left(XX^*\right)^{-1}XY^*. \tag{A.17}$$

Sub in (A.9),(A.11) and (A.12) into (A.17):

$$ZZ^* = P_k^- - YX^*\left(R + CP_k^- C^T\right)^{-1}XY^* \tag{A.18}$$
$$ZZ^* = P_k^- - \left(P_k^-\right)^* C^T\left(R + CP_k^- C^T\right)^{-1}CP_k^- \tag{A.19}$$
$$ZZ^* = P_k^- - P_k^- C^T\left(R + CP_k^- C^T\right)^{-1}CP_k^- \tag{A.20}$$
$$ZZ^* = P_k^+ = (I - K_k C)P_k^- \text{ where } K_k = P_k^- C^T(CP_k^- C^T + R)^{-1}. \tag{A.21}$$

The product ZZ^* matches (A.4) meaning that

$$Z = \left(P_k^+\right)^{\frac{1}{2}}. \tag{A.22}$$

Computing Z only produces the covariance matrix. To compute the gain (A.2) in order to compute (A.3), Y is solved for:

$$YY^* + ZZ^* = P_k^- \tag{A.23}$$
$$YY^* = P_k^- C^T\left(R + CP_k^- C^T\right)^{-1}CP_k^- \tag{A.24}$$
$$YY^* = P_k^- C^T\left(\left(\left(R + CP_k^- C^T\right)^{-1}\right)^{\frac{1}{2}}\right)^*\left(\left(\left(R + CP_k^- C^T\right)^{-1}\right)^{\frac{1}{2}}\right)CP_k^- \tag{A.25}$$
$$YY^* = \left(P_k^- C^T\left(\left(\left(R + CP_k^- C^T\right)^{-1}\right)^{\frac{1}{2}}\right)^*\right)\left(P_k^- C^T\left(\left(\left(R + CP_k^- C^T\right)^{-1}\right)^{\frac{1}{2}}\right)^*\right)^*. \tag{A.26}$$

Using (A.1) and (A.26) solve for YX^{-1}

$$YX^{-1} = \left(P_k^- C^T \left(\left(\left(R + CP_k^- C^T\right)^{-1}\right)^{\frac{1}{2}}\right)^*\right)\left(\left(R + CP_k^- C^T\right)^{\frac{1}{2}}\right)^{-1}. \tag{A.27}$$

However, since

$$P^{-1} = \left(P^{\frac{1}{2}}\left(P^{\frac{1}{2}}\right)^*\right)^{-1} = \left(\left(P^{\frac{1}{2}}\right)^*\right)^{-1}\left(\left(P^{\frac{1}{2}}\right)\right)^{-1} \tag{A.28}$$

then

$$YX^{-1} = P_k^- C^T \left(R + CP_k^- C^T\right)^{-1} = K_k. \tag{A.29}$$

gives us the Kalman gain (A.2).

Appendix B

Kalman implementation

B.1 Discrete-Discrete Extended Kalman Filter

The following three files summarize the discrete-discrete Extended Kalman Filter. The code can also be found at with other filter implementation in both Matlab and C++ at `https://github.com/mannyray/KalmanFilter`.

Listing B.1: ddekf.m

```
function [estimates, covariances ] = ddekf(f_func,jacobian_func,
   dt_between_measurements,start_time,state_count,sensor_count,
   measurement_count,C,Q_root,R_root,P_0_root,x_0, measurements)
%Runs discrete—discrete Extended Kalman filter on data. The initial
%estimate and covariances are at the time step before all the
%measurements — be wary of the off—by—one error. If f_func is a
%linear function the code is equivalent to discrete—discrete Kalman
%filter.
%[estimates, covariances] = ddekf(f_func,jacobian_func,state_count,
   sensor_count,measurement_count,C,Q_root,R_root,P_0_root,x_0,
   measurements)
%INPUT:
%       f_func: x_{k+1} = f_func(x_k,t) where x_k is the state. The
%       function's second argument is time t for cases when the function
%       changes with time. The argument can be also used an internal
%       counter variable for f_func when start_time is set to zero and
%       dt_between_measurements is set to 1.
%
%       jacobian_func(x,t): jacobian of f_func with state x at time t
%
%       dt_between_measurements: time distance between incoming
%       measurements. Used for incrementing time counter for each
%       successive measurement with the time counter initialized with
%       start_time. The time counter is fed into f_func(x,t) as t.
%
%       start_time: the time of first measurement
%
%       state_count: dimension of the state
%
%       sensor_count: dimension of observation vector
%
%       C: observation matrix of size 'sensor_count by state_count'
%
%       R_root: The root of sensor error covariance matrix R where
%       R = R_root*(R_root'). R_root is of size 'sensor_count by
%       sensor_count'. R_root = chol(R)' is one way to derive it.
%
%       Q_root: The root of process error covariance matrix     Q where
%       Q = Q_root*(Q_root'). Q_root is of size 'state_count by
%       state_count'. Q_root = chol(Q)' is one way to derive it.
%
%       P_0_root: The root of initial covariance matrix P_0 where
%       P_0 = P_0_root*(P_root'); P_0_root is of size 'state_count by
%       state_count'. % P_0_root = chol(P_0)' is one way to derive it.
%
%       x_0:Initial state estimate of size 'state_count by 1'
%
%       measurements: ith column is ith measurement. Matrix of size
%       'sensor_count by measurement_count'
%
%OUTPUT:
%       estimates: 'state_count by measurement_count+1'
%       ith column is ith estimate. first column is x_0
%
```

```
%       covariances: cell of size 'measurement_count+1' by 1
%       where each entry is the P covariance matrix at that time

        %make sure input is valid
        assert(size(P_0_root,1)==state_count &&...
         size(P_0_root,2)==state_count);
        assert(size(C,1)==sensor_count && size(C,2)==state_count);
        assert(size(Q_root,1)==state_count &&...
         size(Q_root,2)==state_count);
        assert(size(x_0,1)==state_count && size(x_0,2)==1);
        assert(size(R_root,1)==sensor_count &&...
         size(R_root,2)==sensor_count);
        test = f_func(x_0,start_time);
        assert(size(test,1)==state_count && size(test,2)==1);
        test = jacobian_func(x_0,start_time);
        assert(size(test,1)==state_count && size(test,2)==state_count);
        assert(size(measurements,1)==sensor_count &&...
         size(measurements,2)==measurement_count);

        x_km1_p = x_0;
        P_root_km1_p = P_0_root;

        estimates = zeros(state_count,measurement_count + 1);
        covariances = cell(measurement_count + 1, 1);

        estimates(1:state_count,1) = x_0;
        covariances{1,1} = P_0_root*P_0_root';

        current_time = start_time;

        for k=1:measurement_count
                %loop through all measurements and follow through predict
                %and update phase
                [x_k_m,P_root_km] = ddekf_predict_phase(f_func,...
                        jacobian_func,current_time,P_root_km1_p,x_km1_p,...
                        Q_root);
                [x_k_p,P_root_kp] = ddekf_update_phase(R_root,...
                        P_root_km,C,x_k_m,measurements(:,k));

                x_km1_p = x_k_p;
                P_root_km1_p = P_root_kp;

                %store results
                estimates(:,k+1) = x_k_p;
                covariances{k+1,1} = P_root_kp*(P_root_kp');

                current_time = current_time + dt_between_measurements;
        end
end
```

Listing B.2: ddekf_predict_phase.m

```
function [estimate, covariance_sqrt] = ddekf_predict_phase(f_func,
   jacobian_func,t,P_0_sqrt,x_0,Q_root)
%runs predict portion of update of the dd-ekf
%[estimate, covariance_sqrt] ddekf_predict_phase(f_func,jacobian_func,t,
   P_0_sqrt,x_0,Q_root)
%INPUT:
%       f_func: x_{k+1} = f_func(x_k,t) where x_k is the state. The
%       function's second argument is time t for cases when the function
%       changes with time.
%
%       jacobian_func: the jacobian of f_func at state x and time t
%       jacobian_func(x,t)
%
%       t: current time
%
%       X_0: estimate of x at start_time
%
%       P_0_sqrt: square root factor of covariance P at
%       start_time. P_0 = P_0_sqrt*(P_0_sqrt')
%
%       Q_root: square root of process noise covariance matrix
%       Q where Q = Q_root*(Q_root');
%
%OUTPUT:
```

```
%       estimate: estimate of x after predict phase
%
%       covariance_sqrt: square root of covariance P after predict phase
%       P = covariance_sqrt*(covariance_sqrt')

        x = x_0;
        state_count = length(x);

        estimate = f_func(x,t);
        jac = jacobian_func(x,t);
        [~, covariance_sqrt] = qr([jac*P_0_sqrt,Q_root]');
        covariance_sqrt = covariance_sqrt';
        covariance_sqrt = covariance_sqrt(1:state_count,1:state_count);
end
```

Listing B.3: ddekf_update_phase.m

```
function [estimate_next, covariance_sqrt] = ddekf_update_phase(R_root,
   P_root,C,estimate,measurement)
%runs update portion of the dd-ekf.
%[estimate_next,covariance_sqrt] = ddekf_update_phase(R_root,P_root,C,
   estimate,measurement)
%INPUT:
%       R_root:sensor error covariance matrix where
%       R=R_root*(R_root')
%
%       P_root:covariance matrix from the predict phase
%       P_predict = P_root*(root')
%
%       C: observation matrix
%
%       estimate: estimate from predict phase
%
%       measurement: measurement picked up by sensor
%
%OUTPUT:
%       estimate_next: estimate incorporating measurement
%
```

```
%       covariance_sqrt: square root of covariance at the
%       end of update phase. Actual covariance is
%       covariance_sqrt*(covariance_sqrt')

        measurement_count = size(C,1);
        state_count = size(C,2);

        tmp = zeros(state_count+measurement_count, state_count+
           measurement_count);
        tmp(1:measurement_count,1:measurement_count) = R_root;
        tmp(1:measurement_count,(measurement_count+1):end) = C*P_root;
        tmp((1+measurement_count):end,(1+measurement_count):end) = P_root;

        [Q,R] = qr(tmp');
        R = R';

        X = R(1:measurement_count,1:measurement_count);
        Y = R((1+measurement_count):end,(1:measurement_count));
        Z = R((1+measurement_count):end,(1+measurement_count):end);

        %Kalman_gain = Y*inv(X);
        estimate_next = estimate + Y*(X\(measurement - C*estimate));
        covariance_sqrt = Z;
end
```

B.2 Continuous-Discrete Extended Kalman Filter

The continuous-discrete Extended Kalman filter shares the same update phase with the discrete-discrete Extended Kalman filter. The header is modified as follows

Listing B.4: cdekf.m

```
function [estimates, covariances ] = cdekf(f_func,jacobian_func,
   dt_between_measurements,rk4_steps, start_time, state_count, sensor_count
   , measurement_count,C,Q_root,R_root,P_0_root,x_0, measurements)
%Runs continuous-discrete Extended Kalman filter on data. The initial
%estimate and covariances are at the time step before all the
```

```
%measurements — be wary of the off—by—one error. If f_func is a
%linear function the code is equivalent to continuous—discrete Kalman
%filter.
%[estimates, covariances] = cdekf(f_func,jacobian_func,
   dt_between_measurements,rk4_steps, start_time, state_count, sensor_count
   , measurement_count,C,Q_root,R_root,P_0_root,x_0, measurements)
%INPUT:
%       f_func: \dot{x} = f_func(x,t) where x is the state
%       and lhs is the derivative of x at time t. The derivative is a
%       function of both the time and state.
%
%       jacobian_func(x,t): jacobian of f_func with state x at time t
%
%       rk4_steps: amount of steps taken in predict phase in rk4 scheme
%
%       dt_between_measurements: time distance between incoming
%       measurements
%
%       start_time: the time of first measurement
%
%       state_count: dimension of the state
%
%       sensor_count: dimension of observation matrix
%
%       C: observation matrix of size 'sensor_count by state_count'
%
%       R_root: The root of sensor error covariance matrix R where
%       R = R_root*(R_root'). R_root is of size 'sensor_count by
%       sensor_count'. R_root = chol(R)' is one way to derive it.
%
%       Q_root: The root of process error covariance matrix     Q where
%       Q = Q_root*(Q_root'). Q_root is of size 'state_count by
%       state_count'. Q_root = chol(Q)' is one way to derive it.
%
%       P_0_root: The root of initial covariance matrix P_0 where
%       P_0 = P_0_root*(P_root'); P_0_root is of size 'state_count by
%       state_count'. % P_0_root = chol(P_0)' is one way to derive it.
%
```

```
%       x_0:Initial state estimate of size 'state_count by 1'
%
%       measurements: ith column is ith measurement. Matrix of size
%       'sensor_count by measurement_count'
%
%OUTPUT:
%       estimates: 'state_count'X'measurement_count+1'
%       ith column is ith estimate. first column is x_0
%
%       covariances: cell of size 'measurement_count+1' by 1
%       where each entry is the P covariance matrix at that time.
%       Time is computed based on dt_between_measurements
```

The code follows *ddekf.m* except it is now the predict phase is different

Listing B.5: cdekf_predict_phase.m basicstyle

```
          [x_k_m,P_root_km] = cdekf_predict_phase(f_func,...
                              jacobian_func,h,
                                 dt_between_measurements,
                                 current_time,...
                              P_root_km1_p,x_km1_p,Q_root);
```

with the predict phase given by ‘cdekf_predict_phase.m‘.

Listing B.6: cdekf_predict_phase.m

```
function [estimate, covariance_sqrt] = cdekf_predict_phase(f_func,
   jacobian_func,h,dt_between_measurements,start_time,P_0_sqrt,x_0,Q_root)
%runs predict portion of update of the extended kalman filter
%[estimate, covariance_sqrt] cdekf_predict_phase(f_func,jacobian_func,h,
   dt_between_measurements,start_time,P_0_sqrt,x_0,Q_root)
%INPUT:
%       f_func: encodes relationship \dot{x} = f_func(x,t) for state x
%       and time t.
%
%       jacobian_func: the jacobian of f_func at state x and time t
%
%       h: fixed timestep used by RK4 to cover dt_between_measurements
%       to evolve estimate as well covariance matrix. h must be less
%       than delT
%
%       dt_between_measurements: distance between incoming measurements
%
%       start_time: start time over which solving RK4
%       (finish time is start_time + dt_between_measurements)
%
%       x_0: estimate of x at start_time
%
%       P_0_sqrt: square root factor of covariance P at
%       start_time.     P_0 = P_0_sqrt*(P_0_sqrt')
%
%       Q_root: square root of process noise covariance matrix
%       Q where Q = Q_root*(Q_root');
%
%OUTPUT:
%       estimate: estimate of x at dt_between_measurements + start_time
%
%       covariance_sqrt: square root of covariance P after the
%       update phase. P = covariance_sqrt*(covariance_sqrt')

        current_time = start_time;
        finish_time = start_time + dt_between_measurements;
        x = x_0;
        state_count = length(x);

        Phi = eye(state_count);
        Phi_sum_root = zeros(state_count,state_count);

        while(current_time < finish_time)
                if(current_time + h > finish_time)
                        h = finish_time - current_time;
                end

                %RK4 for estimate
                k1 = h.*f_func(x,current_time);
                k2 = h.*f_func(x + k1./2,current_time+h/2);
                k3 = h.*f_func(x + k2./2,current_time+h/2);
                k4 = h.*f_func(x + k3, current_time + h);
                x = x + k1./6 + k2./3 + k3./3 + k4./6;

                %RK4 for phi
                jac_a  = jacobian_func(x,current_time);
                %k_phi_1 = h.*jac_A*(Phi);
                %k_phi_2 = k_phi_1 + (0.5*h).*jac_A*(k_phi_1);
                %k_phi_3 = k_phi_1 + (0.5*h).*jac_A*(k_phi_2);
                %k_phi_4 = k_phi_1 + h.*jac_A*(k_phi_3);
                %Phi_next = Phi + k_phi_1./6 + k_phi_2./3 + k_phi_3./3 +
                %k_phi_4./6;
                %since jac_A is constant at current time the commented
                   lines
                %can be replaced with the stuff below
                %with th
                %following few lines
                jac_a_2 = jac_a*jac_a;
                jac_a_3 = jac_a_2*jac_a;
                jac_a_4 = jac_a_3*jac_a;
                Phi_next = (eye(state_count) + (h).*jac_a + (1/2*h^2).*
                   jac_a_2 +...
            (1/6*h^3).*jac_a_3 + (1/24*h^4).*jac_a_4)*Phi;

                %Trapezoid rule. add phi and phi_next together
                [~, R_tmp1] = qr([Phi*Q_root,Phi_next*Q_root]');
                R_tmp1 = sqrt(h*0.5).*R_tmp1';
                R_tmp1 = R_tmp1(1:state_count, 1:state_count);

                %Add previous sum to overall phi sum
                [~, R_tmp1] = qr([R_tmp1,Phi_sum_root]');
                R_tmp1 = R_tmp1';
                R_tmp1 = R_tmp1(1:state_count, 1:state_count);
                Phi_sum_root = R_tmp1;

                Phi = Phi_next;
                current_time = current_time + h;
        end

        [~, R_tmp1] = qr([Phi*P_0_sqrt,Phi_sum_root]');
        covariance_sqrt = R_tmp1';
        covariance_sqrt = covariance_sqrt(1:state_count,1:state_count);

        estimate = x;
end
```