LQG AND GAUSSIAN PROCESS TECHNIQUES

FOR FIXED-STRUCTURE WIND TURBINE CONTROL

LQG AND GAUSSIAN PROCESS TECHNIQUES

FOR FIXED-STRUCTURE WIND TURBINE CONTROL

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op woensdag 17 oktober 2018 om 10:00 uur

door

Hildo Janteunis BIJL

ingenieur in de lucht- en ruimtevaart,
geboren te Heerhugowaard, Nederland.

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. M. Verhaegen
prof.dr.ir. J.W. van Wingerden

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof.dr.ir. M. Verhaegen,	Technische Universiteit Delft, promotor
Prof.dr.ir. J.W. van Wingerden,	Technische Universiteit Delft, promotor
Onafhankelijke leden:	
Dr. M. Deisenroth,	Imperial College London
Prof.dr. C.E. Rasmussen,	University of Cambridge
Prof.dr. T.M. Heskes,	Radboud Universiteit Nijmegen
Prof.dr.ir. G.A.M. van Kuik,	Technische Universiteit Delft
Prof.dr. R. Babuska,	Technische Universiteit Delft
Overige leden:	
Prof.dr. T.B. Schön,	Uppsala University

Prof.dr. T.B. Schön heeft in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

This dissertation has been completed in partial fulfilment of the requirements of the Dutch Institute of Systems and Control (DISC) for graduate studies.

This research is supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO) and partly funded by the Ministry of Economic Affairs (project number 12173, SMART-WIND).

Keywords: Gaussian processes, regression, machine learning, optimization, system identification, automatic control, wind energy, smart rotor

Printed by: Ridderprint BV, Ridderkerk

Front & Back: Photo by Master Wen (Unsplash), design by Ridderprint BV, Ridderkerk

ISBN 978-94-6299-501-7

A digital version of this dissertation is available at:
http://repository.tudelft.nl/
For further information and source code, also check out:
http://www.hildobijl.com/Research.php

CONTENTS

SUMMARY

Wind turbines are growing bigger to become more cost-efficient. This does increase the severity of the vibrations that are present in the turbine blades, both due to predictable effects like wind shear and tower shadow, and due to less predictable effects like turbulence and flutter. If wind turbines are to become bigger and more cost-efficient, these vibrations need to be reduced.

This can be done by installing trailing-edge flaps to the blades. Because of the variety of circumstances which the turbine should operate in, this results in large uncertainties. As such, we need methods that can take stochastic effects into account. Preferably we develop an algorithm that can learn from online data how the flaps affect the wind turbine and how to optimally control them.

A simple prior analysis can be done using a linearized version of the system. In this case it is important to know not only the expected cost (damage) that will be incurred by the wind turbine in various situations, but also the spread of this cost. This can for instance be done by looking at the variance of the cost function. Various expressions are available to analytically calculate this variance. Alternatively, we can prescribe a degree of stability for the system.

Due to the limitations of linear approximations of systems, it is more effective to apply nonlinear regression methods. A promising one is Gaussian Process (GP) regression. Given a training set $(X, \boldsymbol{y})$ it can predict function values $f(\boldsymbol{x}_*)$ for test points $\boldsymbol{x}_*$. It has its basis in Bayesian probability theory, which allows it to not only make this prediction, but also give information (the variance) about its accuracy.

The usual way in which GP regression is applied has a few important limitations. Most importantly, it is computationally intensive, especially when applied to constantly growing data sets. In addition, it has difficulties dealing with noise present in the training input points $\boldsymbol{x}$. There are methods to solve either of these issues, but these tricks generally do not work well together, or their combination requires many computational resources. However, by making the right approximations, like Taylor expansions and at times even linearizations, Gaussian process regression can be applied efficiently, in an online way, to data sets with noisy input points. This enables GP regression to be used for system identification problems like online non-linear black-box modeling.

Another limitation is that it can be difficult to find the optimum of a Gaussian process. The reason is that the optimum of a Gaussian process is not a fixed point but a random variable. The distribution of this optimum cannot be calculated analytically, but we can use particle methods to approximate it. We can subsequently use this principle to efficiently explore an unknown nonlinear function, trying to locate its optimum. To do so, we sample a point $\boldsymbol{x}$ from the optimum distribution, measure what the function value $f(\boldsymbol{x})$ at this point is, update the Gaussian process approximation of the function, update

the optimum distribution and repeat this process until the distribution has converged. Finding the optimum of a function like this has shown to have competitive performance at keeping the cumulative regret low, compared to similar algorithms. In addition, it allows wind turbines to tune the gains of a fixed-structure controller so as to optimize a nonlinear cost function like the damage equivalent load.

All these improvements are a step forward in the application of Gaussian process regression to wind turbine applications. But as is always the case with research, there are still many things left to improve further.

SAMENVATTING

Windturbines worden steeds groter om zo goedkoop mogelijk energie te kunnen produceren. Dit verergert wel de trillingen die in de bladen aanwezig zijn, zowel vanwege voorspelbare effecten als windscheer en torenschaduw, maar ook door minder voorspelbare effecten als turbulentie en flutter. Als windturbines groter moeten worden, met een hogere kostenefficiëntie, dan moeten deze trillingen verminderd worden.

Dit kan bereikt worden door verstelbare flappen aan de achterkant van de bladen te zetten. Door de grote variëteit aan omstandigheden die de windturbine zal ondervinden, resulteert dit in grote onzekerheden. Hierdoor hebben we methoden nodig die om kunnen gaan met deze stochastische effecten. Bij voorkeur ontwikkelen we een algoritme dat van online informatie kan leren hoe de flappen de windturbine beïnvloeden en hoe ze vervolgens optimaal bestuurd kunnen worden.

Een simpele analyse vooraf kan gedaan worden via een linearisatie van het systeem. In dit geval is het belangrijk om niet alleen te weten wat de verwachte kosten (schade) zijn die plaats zal vinden in verschillende situaties, maar ook de spreiding van deze kosten. Dit kan bijvoorbeeld gedaan worden door te kijken naar de variantie van de kostenfunctie. Verschillende uitdrukkingen voor deze variantie zijn beschikbaar. Aan de andere kant is het ook mogelijk om een bepaalde graad van stabiliteit voor het systeem voor te schrijven.

Door de beperkingen van lineaire systeembenaderingen is het beter om nonlineaire regressiemethoden te gebruiken. Een beloftevol voorbeeld is Gaussisch proces-regressie (GP-regressie). Gegeven een trainingsset $(X, \boldsymbol{y})$ kan deze methode functiewaarden $f(\boldsymbol{x}_*)$ voorspellen voor testpunten $\boldsymbol{x}_*$. Het heeft een basis in de Bayesiaanse kansrekening, waardoor het niet alleen in staat is om zo'n voorspelling te maken, maar ook informatie te geven (de variantie) over de nauwkeurigheid ervan.

De gebruikelijke manier om GP-regressie toe te passen heeft een paar belangrijke beperkingen. Bovenal is het rekentechnisch een erg complexe methode, vooral als het toegepast wordt op grote en groeiende datasets. Hiernaast heeft het ook moeite met het meerekenen van ruis aanwezig in de invoerpunten $\boldsymbol{x}$. Er zijn methoden om elk van deze problemen op te lossen, maar die trucs gaan over het algemeen niet goed samen, of vereisen erg veel rekenkracht. Echter, door de juiste benaderingen te maken, zoals Taylor-ontwikkelingen en af en toe zelfs een linearisatie, kan Gaussisch proces-regressie op een efficiënte online manier toegepast worden op datasets met stochasische invoerpunten. Dit stelt GP-regressie in staat om gebruikt te worden voor systeemidentificatietoepassingen, zoals online niet-lineaire zwarte doos-modelering.

Een andere beperking is dat Gaussische processen niet gemakkelijk geoptimaliseerd kunnen worden. De reden is dat het optimum van een Gaussisch proces niet een vast punt is, maar een stochastische variabele. De verdeling van dit optimum kan daarnaast

ook niet analytisch berekend worden, maar we kunnen het wel benaderen met numerieke methoden. Dit principe kunnen we vervolgens gebruiken om op efficiënte wijze onbekende nonlineaire functies te verkennen, op zoek naar het optimum. Om dit te doen kiezen we een willekeurig punt $\boldsymbol{x}$, volgens de verdeling van het optimum, meten we wat de bijbehorende functiewaarde $f(\boldsymbol{x})$ op dit punt is, rekenen we dit mee in de Gaussisch proces-benadering, updaten we de distributie van het optimum en herhalen we dit hele proces tot de verdeling geconvergeerd is. Deze wijze van het vinden van het optimum van een functie heeft laten zien dat het concurrerende prestaties heeft, ten opzichte van soortgelijke algoritmen, in het beperkt houden van de totaal opgetelde kosten tijdens het verkennen. Hiernaast stelt het windturbines in staat om de versterkingen van een vaste-structuur regelaar bij te stellen, om op die manier een niet-lineaire kostenfunctie als de schade-equivalente belasting te optimaliseren.

Al deze verbeteringen zijn een belangrijke stap vooruit in de toepassing van Gaussisch proces-regressie op windturbines. Maar zoals altijd het geval is met onderzoek, zijn er nog steeds tal van dingen om verder te verbeteren.

1

INTRODUCTION

On April 22, 2016 the Paris Agreement was signed. This historic agreement requires countries to reduce greenhouse gas emissions, with as main goal to fight global warming. At the time of writing, 195 countries have signed the treaty, with 146 having ratified it. Countries are expected to gradually phase out fossil fuels like coal, gas and oil, and use more renewable energy sources like solar, hydro and wind energy. It is for this reason that there has been a tremendous growth of interest in these types of energy, as well as a large desire to make them more cost-efficient.

When using renewable energy, it is generally wise to use a variety of sources. When one source provides little energy due to external factors – i.e., there is less sun, water or wind available at a given time – then other sources can potentially compensate. As a result, a lot of research is done in all kinds of renewable energy sources. In this thesis we focus on wind energy. We start by looking at how wind energy is developing, what challenges it faces and what is needed to overcome these.

1.1. FACTORS AFFECTING THE COST OF WIND ENERGY

The most important factor in wind energy is the cost of energy. How much money does it cost to produce one kWh of wind energy? And more importantly: what decisions should we make to lower this cost? It is important here to realize that this cost-efficiency can, roughly put, be calculated through

$$\text{Cost-efficiency} = \frac{\text{Total energy produced}}{\text{Total capital costs} + \text{Total operating costs}}. \tag{1.1}$$

There are many design choices that affect this cost-efficiency, and they do so in a variety of ways.

Probably the most important factor is the location of the wind turbine. As noted by Navalkar (2016), many wind turbines reside on land in flat wind-rich regions of low population density with a stable grid connection and a green government policy. The reason for this is clear: land-based turbines generally have a lower operating cost, because

inspection and maintenance is easier to perform. At the same time, the set-up costs are also lower: the turbine components make up 68% of the capital costs for onshore turbines, but 32% for offshore turbines (Moné et al., 2015). The difference is mainly caused by the additional infrastructure that is required for offshore turbines.

However, with most of the suitable onshore locations already taken (Gebraad, 2014), and with an increase in social pressure against such sites, the focus starts to shift more and more towards offshore wind energy (van Solingen, 2015). At the same time, these locations have some favorable properties of their own (Hau, 2006). Offshore wind turbines have more persistent wind speeds, enabling them to extract more power from the available wind (Pryor and Barthelmie, 2001, 2002). In addition, the lower surface roughness allows the wind speed above the sea increases more rapidly than it does on land. This allows for lower towers. Finally, turbulence levels are lower (Türk and Emeis, 2010), positively affecting the amount of fatigue damage incurred by such turbulence.

Another factor that influences the cost-efficiency is the way that turbines are grouped. When multiple wind turbines in turbine parks are grouped closely together, then costs are saved due to easier installation, lower cable lengths, fewer space requirements and such. On the flip side, wind turbines are subject to aerodynamic interaction, where the upwind wind turbines reduce the power that lower wind turbines can generate (Gebraad, 2014). This holds for both onshore and offshore wind turbines, but more strongly for the latter, because on sea turbine wakes recover more slowly as a result of less mixing of the air (Hansen et al., 2012).

The size of wind turbines also strongly affects the cost-efficiency. The power generated by a wind turbine scales quadratically with the rotor diameter (Bianchi et al., 2007, Wagner and Mathur, 2013). In other words, a wind turbine that is twice as large will generate four times as much power. As a result, larger wind turbines have a significantly higher energy production, while the costs do not scale up that way. This has resulted in wind turbines growing larger and larger, as is also shown by Figure 1.1. One of the main factors limiting this increase in size is fatigue damage. Bigger wind turbines are subject to more intense vibrations, and the resulting fatigue limits the lifetime of the turbine. To continue this trend of growing wind turbines, it is therefore crucial to reduce the fatigue damage sustained by the turbines. One way to do so would be through active control.

We will study active control of wind turbines more in-depth next, in Section 1.2. But first it is worthwhile to note its effects on the cost-efficiency. The conventional way of reducing the bending stresses present, is by adding more material to the turbine blades. When we instead use active control to reduce the bending stresses, it is possible to reduce the amount of materials required to produce wind turbines, effectively bringing down the total capital cost. At the same time, the lifetime of the turbine may increase due to a lower fatigue load, resulting in a higher total amount of energy produced. However, applying active control generally means that the wind turbine has more moving parts, which may require more maintenance, adding to the total operating costs.

It can be concluded that there are many factors that affect the cost-efficiency of the turbine and finding the 'perfect wind turbine' is something that will still take a lot of research. This especially holds when it comes to actively reducing the vibrations within wind turbines, which is what we look at next.

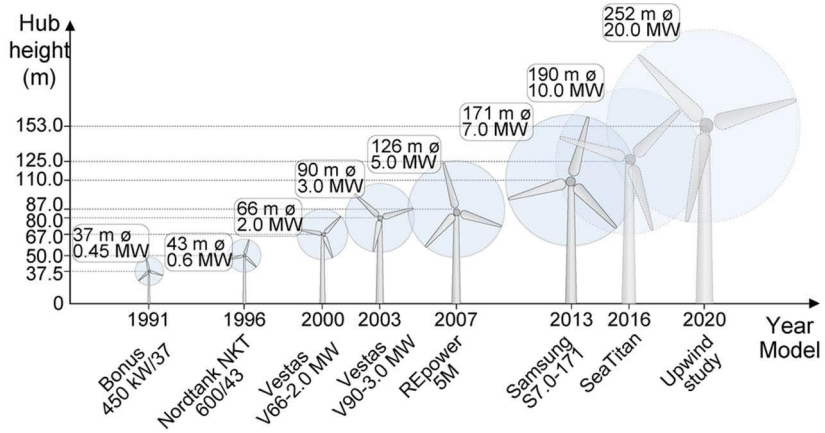

Figure 1.1: Development of the size of wind turbines over the years. The future sizes are a prognosis, whose validity strongly depends on whether the technical challenges related to larger wind turbines can be solved. Source: Rodrigues et al. (2016). Similar results are given by Philibert and Holttinen (2013).

1.2. TACKLING THE VIBRATIONS IN THE WIND TURBINE

Vibrations play a significant role within wind turbines. For example, the fluctuating bending moments at the root of the turbine blades cause fatigue damage. In addition, vibrations also move up the drive train, potentially causing damage in the generator. It is therefore crucial to reduce these vibrations.

1.2.1. THE CAUSE OF VIBRATIONS

The vibrations within wind turbines are caused by a variety of factors. Some of the factors are predictable. An important example is wind shear, which is defined as a variation in wind velocity occurring along a direction at right angles to the wind's direction. The most common one here is vertical wind shear (as opposed to horizontal wind shear) which is a variation in wind speed at different altitudes. In practice, higher altitudes often have higher wind speeds, as shown by Figure 1.2. This causes a fluctuating root bending moment within the blades. Another predictable factor is tower shadow. This is a reduction of the wind speed near the turbine tower and it also causes significant cyclic loads.

There are also less predictable causes of vibrations. These are mostly by fluctuations within the wind field. These variations can take the form of gusts (a sudden increase in the wind speeds), turbulence (rapid variations in air pressure and flow velocity) or other more stationary yet still unpredictable variations in the wind speed. In bad cases the turbine blade may also be subject to flutter, which is a dynamic instability of the blade, caused by positive feedback between the blade deflection/inclination and the aerodynamic force applied to it.

Altogether, these vibrations have the potential to cause significant fatigue damage in the turbine. If turbines are to grow larger, or have a longer lifetime, it is therefore crucial to reduce the vibrations. There are various ways to affect the workings of the turbine, and we will study them next.

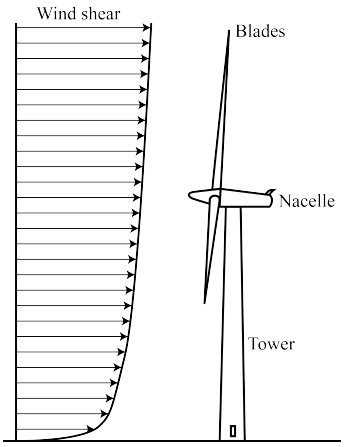

Figure 1.2: A wind turbine with the incoming wind visualized. Because the wind speed is lower on lower altitudes, the turbine experiences vertical wind shear. In addition, because the tower blocks the incoming wind, tower shadow occurs.

1.2.2. METHODS OF AFFECTING THE TURBINE STATE

There are three common ways of controlling a wind turbine. The first is by yawing the turbine. This is used to rotate it into the wind, but minor adjustments can also be made to direct the wind turbine wake away from other turbines downstream (Gebraad, 2014). The second is by using the generator torque to control the rotor speed of the wind turbine. This is generally down for below-rated wind conditions. At above-rated wind conditions, it is more common to adjust the pitch angle of the blades to keep the rotor speed limited, which is done through full-span pitch actuators.

Conventionally, the pitch actuators give all turbine blades the same pitch angle. However, instead of collectively controlling the pitch angle, it is also possible to apply Individual Pitch Control (IPC). This means that the pitch angle of the individual blades – each weighing dozens of tons – are adjusted several times during each rotation of the turbine. This technique has shown to be effective at reducing the first dominant peak of the load frequency spectrum, but is not capable at reducing higher frequency vibrations due to the limitations of the pitch actuator (Navalkar, 2016).

A more promising approach at reducing higher frequency vibrations is to install flaps on the trailing edge of the blades, as shown in Figure 1.3. These flaps can be actuated at much higher frequencies than what IPC can obtain. They can be actuated using small motors (Berg et al., 2012), using shape memory alloys (Hulskamp, 2011) or using piezo-actuators (Bak et al., 2007, Hulskamp, 2011). For a complete overview of control methods, see (Pechlivanoglou, 2012). There are also other methods of controlling the air flow, like the use of synthetic jets, microtabs or microflaps. These have been explored in the literature, but their application to turbine prototypes is still very scarce. These are hence still very novel technologies, whose feasibility still needs to be proven. At the same time, there is ample evidence that trailing-edge flaps are capable of reducing the vibrations (Castaignet et al., 2012, Andersen et al., 2006, Basualdo, 2005, Buhl et al., 2005). That is why we will consider trailing-edge flaps in this thesis.

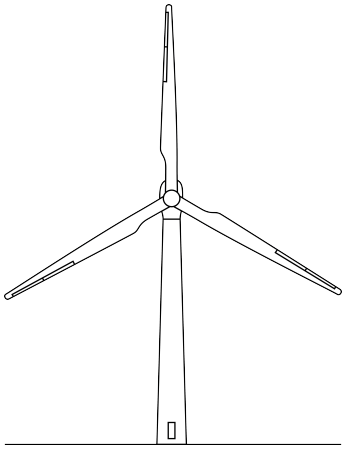

Figure 1.3: A wind turbine with trailing-edge flaps attached to the blades. Contrary to individual pitch control, these flaps are capable of reducing relatively high-frequency oscillations.

1.2.3. METHODS OF OBTAINING DATA

To be able to properly affect the state of the wind turbine, it is crucial to know what state it is in. After all, active control can very well amplify vibrations instead of reducing them. This requires measurements. In literature, a variety of data has been used, including pitot tubes to measure air pressure (Heinz et al., 2010), strain gauges to measure moments (Castaignet et al., 2012), accelerometers to measure motion, or LiDAR technology to measure the incoming airflow (Pechlivanoglou, 2012). This data can then be used to actuate the flap.

Although the measured data is not only necessary to estimate the state of the wind turbine. It is also crucial at determining its behavior. Although the dynamics of the wind turbine can be predicted reasonably well through first-principle modeling, they will never exactly match the dynamics of the real turbines (van der Veen et al., 2013). There are always minor variations in the manufacturing process, and these differences between turbines are likely to grow over time as the turbines age. As a result, a data-driven controller can very well outperform a model-based controller (Formentin et al., 2014). This is why it is crucial to obtain a sufficient amount of data to tune controllers.

1.2.4. CALCULATING THE FATIGUE LOAD

The goal of the control law should ideally be to reduce the fatigue load as much as possible. However, calculating the amount of 'fatigue' is not a trivial matter. There are two important matters: at what point of the blade do we want to minimize the fatigue? And how do we quantify the fatigue?

An important quantity for wind turbines, that can also be measured relatively easily, is the blade Root Bending Moment (RBM). As a result, it makes sense to only focus on the stresses present at the root of the blade. This is also the place where the bending moment is the largest. That is why, in this thesis, we will focus on the RBM.

To calculate the fatigue damage at the blade root, we need to take all fluctuations of

the RBM into account. This is not trivial, since the blade will be subject to lots of small oscillations, a few big ones, and many in-between. An example of the blade RBMs during the regular operation of a wind turbine is shown in Figure 1.4.

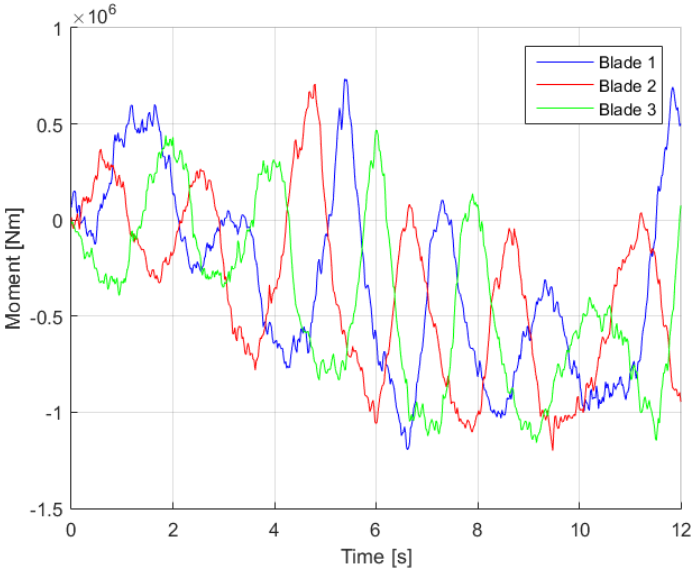

Figure 1.4: The root bending moments of the wind turbine in flapping direction. The values given are in the rotating reference frame. The numbers were produced by a TURBU simulation (van Engelen and Braam, 2004) using representative operating conditions. It can be seen that, next to the default oscillations of the turbine due to the rotation of the blades, there are also many smaller oscillations.

The first step is to analyze how many oscillations of which magnitude our blades have been subject to. This is done through rainflow counting (Niesłony, 2009). An explanation of this process is shown in Figure 1.5.

This gives us the magnitudes of all the individual oscillations encountered. The next step is to compare them with each other. To do this, we use the S-N Curve. Here, S is the stress magnitude of the oscillations that the blade encounters and N is the number of such oscillations that the material can survive before failing; also known as the life cycles. The S-N curve is generally plotted using a logarithmic scale for N. In this case, it takes a mostly linear shape for most materials. This means that N and S approximately satisfy a relation of the form $SN^m = \text{constant}$. That is, if the oscillation stress S becomes twice as big, the number of life cycles is reduced by a factor 2^m. Or alternatively, 2^m oscillations of stress S are equivalent to one oscillation of stress $2S$. This equivalence rule is known as Miner's rule (Wirsching et al., 1995) and the factor m is known as the Wöhler exponent. The exponent depends on the material used, with our glass fiber turbine blades having a Wöhler exponent of $m = 11$ (Savenije and Peeringa, 2009).

Using this equivalence principle, we can calculate the Damage Equivalent Load (DEL) (Freebury and Musial, 2000). This is the 1 Hz oscillatory load that would result in the same fatigue damage as the RBM that has been sustained. By calculating this quantity, it becomes easy to compare the load sustained during experiments with different durations.

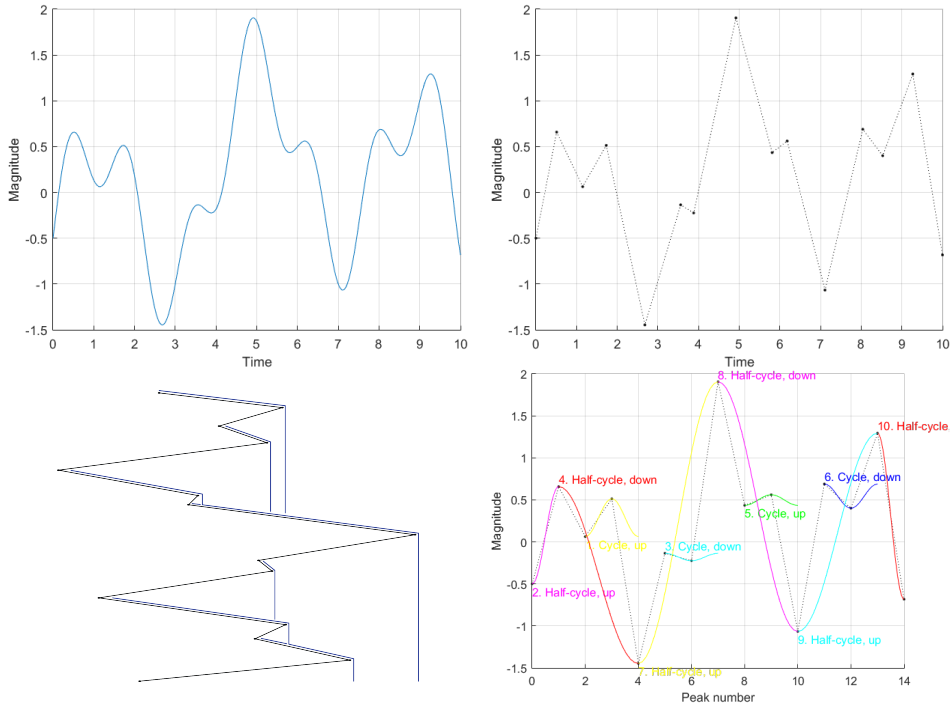

Figure 1.5: The rainflow procedure visualized. We start with a signal to analyze (top left). We then extract the peaks of this signal (top right). Everything else is irrelevant. We turn the plot sideways and let rain flow down from every rightward turning point (bottom left). When two flows meet, the shorter one (based on horizontal distance traveled) is cut off and the longer one continues. Based on these flow lengths, we can split the signal up into individual oscillations (bottom right). Here we should distinguish half oscillations from full oscillations. The bottom right figure was generated using the tools made by Niesłony (2009).

Concluding, the goal of our controller would be to minimize the DEL. Although in reality there are still more facets to it. After all, not only the damage sustained by the turbine matters, but also the control input required to obtain this amount of damage, and possibly other factors play a role as well. Optimal control of wind turbines is hence always a multi-objective optimization problem with a nonlinear objective function.

1.3. POSSIBLE CONTROL METHODOLOGIES

As mentioned before, our main challenge when designing a controller for wind turbine lies in dealing with the uncertainties present. This is partly due to uncertainties in the wind turbine model, but also the wind turbine environment is highly stochastic in nature. To adequately control the trailing-edge flaps, reducing the vibrations, we need methods to take into account these uncertainties.

1.3.1. APPLYING LINEAR METHODS

To improve wind turbine control methods, we can first look into providing improvements to currently existing methodologies. For that, we must examine the kind of controllers currently used.

As is the case in many industries, wind turbine producers traditionally have a strong preference for applying proven techniques, such as linear control laws (Bossanyi, 2000). This is despite the fact that wind turbines are inherently nonlinear systems. Control methodologies like Proportional-Integral-Derivative (PID) control, possibly with low-pass filters and notch filters, are commonly applied.

A probable step forward is to apply the slightly more sophisticated Linear-Quadratic-Gaussian (LQG) control. This is not applied as much yet, but as it falls within the realm of linear control laws, the industry is comparatively more likely to accept it. At the same time, research on applying it to wind turbines is common and recent (Hur and Leithead, 2017).

LQG has as advantage that it has a clear cost function to optimize. If we can let this cost function approximate the DEL, we can tune the controller parameters to reduce the fatigue damage sustained. Methods to optimize the LQG cost already exist. (For a full literature overview on this, see Chapter 2.) However, at the same time we should also take care that the cost does not exceed a certain threshold. If that happens, complete failure of the wind turbine may occur. This means that we should look into the statistical properties of the LQG cost. It leads to the first question of this thesis which we will study in Chapter 2.

Question 1a: *Within LQG control, how can we reduce the probability that the cost exceeds a given threshold?*

When using LQG control, we need accurate knowledge of both the system model and the system state. When the state is not known, it can be observed, for instance through a Kalman filter (Kalman and Bucy, 1961). Identically, when there is uncertainty in the system model, we can prescribe a degree of stability (Anderson and Moore, 1969). Combining these ideas has not been done before (see Chapter 3 for a complete literature overview) which leads to the second question of this thesis.

Question 1b: *In LQG control, how can we prescribe a degree of stability while also optimally applying a state estimator?*

Linear control methodologies do have significant limitations. Linear systems, even Linear-Parameter-Varying (LPV) systems, cannot fully describe all the wind turbine dynamics (Boukhezzar and Siguerdidjane, 2010). More importantly, it is impossible to use a linear control methodology to optimize a highly nonlinear cost function such as the DEL. In fact, many other methods like LPV-control (Tóth, 2010, Briat, 2015) and robust control (Green and Limebeer, 1995, Skogestad and Postlethwaite, 2005) also cannot incorporate a cost function like the DEL. For that reason, we need a method that can both approximate any type of nonlinear function, as well as deal with large degrees of uncertainty.

In this thesis we have chosen to apply Gaussian Process (GP) regression (Rasmussen and Williams, 2006). This technique has its basis in Bayesian probability theory, making it ideally suited to deal with stochastic effects. In addition, given the right covariance function (kernel) it can approximate any nonlinear function. A large part of this thesis (Chapters 4 to 6) will be devoted to investigating how it can be used to analyze and control wind turbines.

1.3.2. SYSTEM IDENTIFICATION METHODS

We start applying GP regression by analyze wind turbine models. When a wind turbine enters production, an important question is whether we can obtain a model of it. Due to the problems described before, we cannot solely use first-principle modeling. We need a data-based approach. And for economic reasons, the downtime of the wind turbine must remain limited. Ideally, the wind turbine continues to operate while the nonlinear system identification algorithm is running.

One option is to combine first-principle modeling with adaptive control (Landau et al., 2016), which comes down to performing online adjustments of various system parameters. Though effective, this method has limitations in the sense that it can only adjust to what falls within its model. A more powerful method is black-box modeling, where as few assumptions are made in advance about the model of the system that is to be identified. This leads to the third question this thesis is meant to answer.

Question 2: *How can Gaussian process regression be used for online black-box modeling of a nonlinear multi-variable system like a wind turbine?*

To answer this question, it is crucial that the identification process does not require too much computational power. If we require months of computations to obtain a model of the wind turbine, while the turbine itself is already up and running, then clearly the identification method is not suitable for our purposes. We will look into detail into this question in Chapter 4.

1.3.3. OPTIMIZING THE NONLINEAR COST FUNCTION

As mentioned before, our main goal is to minimize the DEL from Section 1.2.4. This is a complicated and highly nonlinear cost function. Existing control methods like linear-parameter-varying control and robust control control all optimize specific types of cost functions, and the DEL certainly does not fall among them. We could try to approximate it using GP regression.

At the same time, we should take into account the conservative nature of the wind turbine industry. An advanced nonlinear control law is unlikely to be adopted, so it is more worthwhile to restrict ourselves to fixed-structure control laws with only a handful of controller gains. We hence need to find a way to approximate and subsequently optimize the DEL with respect to these gains. Gaussian process regression is once more very suitable at dealing with nonlinear functions, especially when uncertainty is involved. This leads to the following research question, which we study in Chapter 5.

Question 3a: *How can Gaussian process regression be used to approximate and optimize the cost function of a system with a fixed-structure control law?*

In reality the situation is more involved. Instead of just doing experiments and subsequently optimizing the controller gains, we should tune the gains in an online way. After all, wind turbine downtime will only result in additional costs. In addition, we also want the wind turbine to incur as little damage as possible from the tuning process. This leads to the final and most important question this thesis is meant to answer.

Question 3b: *How can the gains of a fixed-structure control law be tuned online, in such a way as to minimize the damage sustained by the wind turbine?*

This question will be the focus of Chapter 6, where it will be studied in-depth.

1.4. STRUCTURE OF THIS THESIS

This thesis will answer the questions that have been posed, where each chapter tackles one individual subquestion. A full overview is shown in Figure 1.6.

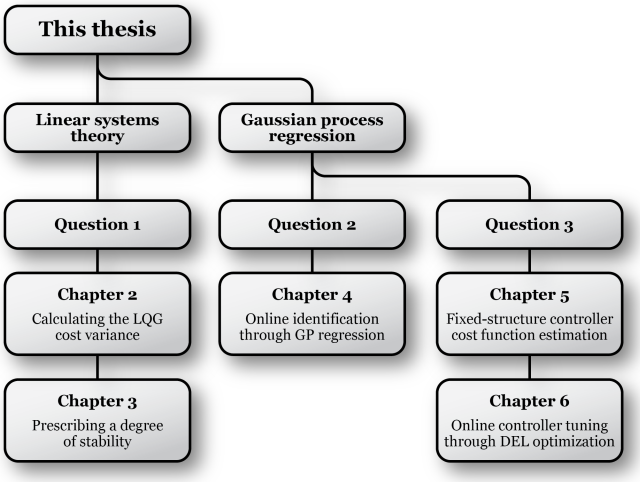

Figure 1.6: A flow chart of the thesis, showing which questions are answered in which chapter. One branch requires prior knowledge on linear systems theory, while the other branch assumes prior knowledge on Gaussian process regression.

- *Chapter 2* looks at the variance of the cost of LQG systems, deriving various analytic expressions to calculate it. It is based on Bijl et al. (2016), and the main contribution is a set of expressions to calculate the variance of the LQG cost function.
  Full citation: Hildo Bijl, Jan-Willem van Wingerden, Thomas B. Schön, and Michel Verhaegen. Mean and variance of the LQG cost function. Automatica, 67:216–223, May 2016b.

- *Chapter 3* studies whether it is possible to prescribe a degree of stability for LQG systems, and how the state can in that case be optimally observed. This chapter is based on Bijl and Schön (2017) and its contribution is an expression for the optimal observer gains when optimizing the discounted LQG cost.
  Full citation: Hildo Bijl and Thomas B. Schön. Optimal controller/observer gains of discounted-cost LQG systems. Submitted for publication to Automatica, 2017.
- *Chapter 4* aims to set up a system identification algorithm using Gaussian process regression. Specifically, it tries to take into account noisy measurements in an online way. This work is based on Bijl et al. (2017b) and contributes a novel online nonlinear black-box system identification algorithm.
  Full citation: Hildo Bijl, Thomas B. Schön, Jan-Willem van Wingerden, and Michel Verhaegen. System Identification through Online Sparse Gaussian Process Regression with Input Noise. IFAC Journal of Systems and Control, 2:1–11, December 2017.
- *Chapter 5* examines whether Gaussian process regression can be applied to approximate a nonlinear cost function of a system. To account for finite-time experiments, it does this through a reinforcement learning set-up. The contents are based on Bijl et al. (2014) and the main contribution is a method of combining GP regression and reinforcement learning.
  Full citation: Hildo Bijl, Jan-Willem van Wingerden, and Michel Verhaegen. Applying Gaussian processes to reinforcement learning for fixed-structure controller synthesis. In Proceedings of the 19th IFAC World Congress, pages 10391–10396, 2014b.
- *Chapter 6* looks at whether a fixed-structure controller can be tuned in an online way, so as to minimize the total amount of damage sustained by the wind turbine. The resulting Bayesian optimization algorithm has been described in Bijl et al. (2017a). The main contribution of this chapter is an algorithm of approximating the maximum distribution as well as its application to Thompson sampling for Bayesian optimization.
  Full citation: Hildo Bijl, Thomas B. Schön, Jan-Willem van Wingerden, and Michel Verhaegen. A sequential Monte Carlo approach to Thompson sampling for Bayesian optimization. Published on arXiv.org, 2017.

At the end, in Chapter 7, conclusions drawn throughout the thesis are summarized and recommendations for future work are given.

2

COST FUNCTION DISTRIBUTION FOR LQG SYSTEMS

Abstract — Linear-Quadratic-Gaussian (LQG) systems are well-understood and methods to minimize the expected cost are readily available. Less is known about the statistical properties of the resulting cost function. In this chapter we examine this LQG cost function, deriving analytic expressions for its mean and variance. These expressions are derived using two different methods, one using solutions to Lyapunov equations and the other using only matrix exponentials. Both the discounted and the non-discounted cost function are considered, as well as the finite-time and the infinite-time cost function. The derived expressions are successfully applied to an example system to reduce the probability of the cost exceeding a given threshold.

The wind turbine industry has a strong preference for linear control laws. That is why this chapter focuses on the linear control methodology of Linear-Quadratic-Gaussian (LQG) control. This technique is a step forward from the conventional PID controllers that are applied, and is receiving recent interest from the wind energy community (Hur and Leithead, 2017).

We start by studying related literature in Section 2.1, present the problem formulation in Section 2.2 and derive the expressions that solve this problem in Section 2.3. Section 2.4 then shows how the equations can be applied to LQG systems, which is subsequently done in Section 2.5. Supporting theorems for this chapter are given in Appendix A.

2.1. LITERATURE OVERVIEW

The LQG control paradigm is generally well-understood in literature. (See for instance Anderson and Moore (1990), Skogestad and Postlethwaite (2005), Bosgra et al. (2008), Åström (1970).) There are many methods available of calculating and minimizing the expected cost $\mathbb{E}[J]$. However, much less is known about the resulting distribution of the cost function J. Yet in many cases (like in machine learning applications, in risk analysis

2

and similar stochastic problems) knowledge of the full distribution of the cost function J, or at least knowledge of its variance $\mathbb{V}[J]$, is important. That is the focus of this chapter. We derive analytical expressions for both the mean $\mathbb{E}[J]$ and the variance $\mathbb{V}[J]$ of the cost function distribution for a variety of cases. This chapter is based on Bijl et al. (2016), which is the first publication containing expressions for the variance $\mathbb{V}[J]$.

The cost function J is usually defined as an integral over a squared non-zero-mean Gaussian process, turning its distribution into a generalized noncentral χ^2 distribution. This distribution does not have a known Probability Density Function (PDF), although its properties have been studied before in literature, for instance in Rice (1944), Schwartz (1970), Sain and Liberty (1971), and methods to approximate it are discussed in Mathai and Provost (1992), Davies (1980). However, none of this theory has been applied, or can directly be applied, to the LQG cost function.

In LQG control most methods focus on the expected cost $\mathbb{E}[J]$, but not all. For instance, Minimum Variance Control (MVC) (see Åström (1970)) minimizes the variance of the output $\boldsymbol{y}$, while Variance Constrained LQG (VCLQG) (see Collins and Selekwa (1999), Conway and Horowitz (2008)) minimizes the cost function subject to bounds on the variance of the state $\boldsymbol{x}$ and/or the input $\boldsymbol{u}$. Alternatively, in Minimal Cost Variance (MCV) control (see Kang et al. (2014), Won et al. (2008)) the mean cost $\mathbb{E}[J]$ is fixed through an equality constraint and the cost variance $\mathbb{V}[J]$ (or alternatively the cost cumulant) is then minimized. However, no expression for the cost variance $\mathbb{V}[J]$ is given. So we will derive one ourselves.

2.2. PROBLEM FORMULATION

We consider continuous linear systems subject to stochastic process noise. Formally, we write these as

$$d\boldsymbol{x}(t) = A\boldsymbol{x}(t)\,dt + d\boldsymbol{w}(t), \tag{2.1}$$

where $\boldsymbol{w}(t)$ is a vector of Brownian motions. (Note that (2.1) is not an LQG system, because it is lacking input. The extension to LQG systems will be discussed in Section 2.4.) As a result, $d\boldsymbol{w}(t)$ is a Gaussian random process with zero-mean and an (assumed constant) covariance of $V\,dt$. Within the field of control (see for instance Skogestad and Postlethwaite (2005)) this system is generally rewritten according to

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + \boldsymbol{v}(t), \tag{2.2}$$

where $\boldsymbol{v}(t)$ is zero-mean Gaussian white noise with intensity V. That is, $\mathbb{E}[\boldsymbol{v}(t)\boldsymbol{v}^T(\tau)] = V\delta(t-\tau)$, with $\delta(.)$ the Kronecker delta function. From a formal mathematical perspective this simplification is incorrect, because $\boldsymbol{v}(t)$ is not measurable with nonzero probability. However, since this notation is common in the control literature, and since it prevents us from having to evaluate the corresponding Itô integrals, we will stick with it. (See Øksendal (1985) for how to properly deal with stochastic differential equations.)

We assume that the initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ has a Gaussian distribution satisfying

$$\boldsymbol{\mu}_0 \equiv \mathbb{E}[\boldsymbol{x}_0] \quad \text{and} \quad \Sigma_0 \equiv \mathbb{E}[\boldsymbol{x}_0\boldsymbol{x}_0^T]. \tag{2.3}$$

Note that the variance of $\boldsymbol{x}_0$ is *not* Σ_0, but actually equals $\Sigma_0 - \boldsymbol{\mu}_0\boldsymbol{\mu}_0^T$. We will consider two different cost functions: the infinite-time cost J and the finite-time cost J_T, respectively

defined as[1]

$$J \equiv \int_0^\infty e^{2\alpha t} \boldsymbol{x}^T(t) Q \boldsymbol{x}(t)\, dt, \tag{2.4}$$

$$J_T \equiv \int_0^T e^{2\alpha t} \boldsymbol{x}^T(t) Q \boldsymbol{x}(t)\, dt, \tag{2.5}$$

where Q is a user-defined symmetric weight matrix. The parameter α can be positive or negative. If it is positive, it is known as the prescribed degree of stability (see Anderson and Moore (1990) or Bosgra et al. (2008)), while if it is negative (like in Reinforcement Learning applications) it is known as the discount exponent. Integral (2.4) does not always converge to a finite value, but it does so for the conditions specified by Theorem 2.2.

2.3. MEAN AND VARIANCE OF THE LQG COST FUNCTION

In this section we derive expressions for $\mathbb{E}[J]$, $\mathbb{E}[J_T]$, $\mathbb{V}[J]$ and $\mathbb{V}[J_T]$. An overview of derived theorems, as well as the corresponding requirements, is shown in Table 2.1.

Table 2.1: The theorems with which the mean and variance of J and J_T can be found, as well as the requirements for these theorems.

	If $\alpha \neq 0$	If $\alpha = 0$	Requirements
$\mathbb{E}[J_T]$	Th. 2.1	Th. 2.3	A and A_α Sylvester
$\mathbb{E}[J]$	Th. 2.2		$\alpha < 0$ and A_α stable
$\mathbb{V}[J_T]$	Th. 2.4	Th. 2.6	$A_{-\alpha}$, A, A_α and $A_{2\alpha}$ Sylvester
$\mathbb{E}[J]$	Th. 2.5		$\alpha < 0$ and A_α stable

2.3.1. NOTATION AND TERMINOLOGY

Concerning the evolution of the state, we define $\boldsymbol{\mu}(t) \equiv \mathbb{E}[\boldsymbol{x}(t)]$, $\Sigma(t) \equiv \mathbb{E}[\boldsymbol{x}(t)\boldsymbol{x}^T(t)]$ and $\Sigma(t_1, t_2) \equiv \mathbb{E}[\boldsymbol{x}(t_1)\boldsymbol{x}^T(t_2)]$. These quantities can be found through the theorems of Appendix A.1.

We define the matrices $A_\alpha \equiv A + \alpha I$ and similarly $A_{k\alpha} \equiv A + k\alpha I$ for any number k. We also define $X^Q_{k\alpha}$ and $\bar{X}^Q_{k\alpha}$ to be the solutions of the Lyapunov equations

$$A_{k\alpha} X^Q_{k\alpha} + X^Q_{k\alpha} A^T_{k\alpha} + Q = 0, \tag{2.6}$$

$$A^T_{k\alpha} \bar{X}^Q_{k\alpha} + \bar{X}^Q_{k\alpha} A_{k\alpha} + Q = 0. \tag{2.7}$$

We often have $\alpha = 0$. In this case A_0 equals A, and we similarly shorten X^Q_0 to X^Q. The structure inherent in the Lyapunov equation induces interesting properties in its solutions $X^Q_{k\alpha}$, which are outlined in Appendix A.2.

[1] The T-notation is rather overloaded here. It is customary to write both a transpose $\boldsymbol{x}^T$ and a time period T using the letter T, and we did not want to deviate from this convention. Keep in mind that, if you see a vector/matrix with a T, then T denotes the transpose. Otherwise T represents the time period.

We define the time-dependent solution $X^Q_{k\alpha}(t_1,t_2)$ as

$$X^Q_{k\alpha}(t_1,t_2)=\int_{t_1}^{t_2} e^{A_{k\alpha}t}Qe^{A_{k\alpha}^T t}\,dt. \tag{2.8}$$

This integral can be calculated efficiently by solving a Lyapunov equation. (See Theorem A.7.) Often it happens that the lower limit t_1 of $X^Q_{k\alpha}(t_1,t_2)$ equals zero. To simplify notation, we then write $X^Q_{k\alpha}(t)\equiv X^Q_{k\alpha}(0,t)$. Another integral solution $\tilde{X}^Q_{k_1\alpha,k_2\alpha}(T)$ is defined as

$$\tilde{X}^Q_{k_1\alpha,k_2\alpha}(T)\equiv\int_0^T e^{A_{k_1\alpha}(T-t)}Qe^{A_{k_2\alpha}t}\,dt. \tag{2.9}$$

This quantity can be calculated (see van Loan (1978)) through

$$\tilde{X}^Q_{\alpha_1,\alpha_2}(T)=\begin{bmatrix}I & 0\end{bmatrix}\exp\left(\begin{bmatrix}A_{\alpha_1} & Q\\ 0 & A_{\alpha_2}\end{bmatrix}T\right)\begin{bmatrix}0\\ I\end{bmatrix}. \tag{2.10}$$

Considering terminology, we say that a matrix A is *stable* (Hurwitz) if and only if it has no eigenvalue λ_i with a real part equal to or larger than zero. Similarly, we say that a matrix A is *Sylvester* if and only if it has no two eigenvalues λ_i and λ_j (with possibly $i=j$) satisfying $\lambda_i=-\lambda_j$. This latter definition is new in literature, only defined before in Bijl et al. (2016).

2.3.2. THE EXPECTED COST

We now examine the expected costs $\mathbb{E}[J]$ and $\mathbb{E}[J_T]$. Expressions for these costs are already known for various special cases. (See for instance Bosgra et al. (2008), Åström (1970).) To provide a complete overview of the subject, we have included expressions which are as general as possible.

Theorem 2.1. *Consider system (2.2). Assume that $\alpha\neq 0$ and that A and A_α are both Sylvester. The expected value $\mathbb{E}[J_T]$ of the finite-time cost J_T (2.5) then equals*

$$\mathbb{E}[J_T]=tr\left(\left(\Sigma_0-e^{2\alpha T}\Sigma(T)+\left(1-e^{2\alpha T}\right)\left(\frac{-V}{2\alpha}\right)\right)\bar{X}^Q_\alpha\right). \tag{2.11}$$

Proof. From (2.5) follows directly that

$$\mathbb{E}[J_T]=\mathrm{tr}\left(\int_0^T e^{2\alpha t}\Sigma(t)\,dt\,Q\right)=\mathrm{tr}\left(Y(T)Q\right), \tag{2.12}$$

where $Y(T)$ is defined as the above integral. To find it, we multiply (A.8) by $e^{2\alpha t}$ and integrate it to get

$$\int_0^T e^{2\alpha t}\dot{\Sigma}(t)\,dt=AY(T)+Y(T)A^T+\int_0^T e^{2\alpha t}V\,dt. \tag{2.13}$$

The left part, through integration by parts, must equal

$$\int_0^T e^{2\alpha t}\dot{\Sigma}(t)\,dt=\left(e^{2\alpha T}\Sigma(T)-\Sigma_0\right)-2\alpha Y(T). \tag{2.14}$$

As a result, $Y(T)$ must satisfy the Lyapunov equation

$$A_\alpha Y(T) + Y(T)A_\alpha^T + \left(\frac{e^{2\alpha T}-1}{2\alpha}V + \Sigma_0 - e^{2\alpha T}\Sigma(T)\right) = 0. \tag{2.15}$$

Using Theorem A.8, we can now write $Y(T)$ as

$$Y(T) = \frac{e^{2\alpha T}-1}{2\alpha}X_\alpha^V + X_\alpha^{\Sigma_0} - e^{2\alpha T}X_\alpha^{\Sigma(T)}. \tag{2.16}$$

Combining this with (2.12) and applying Theorem A.9 (with $F = G = I$) completes the proof. □

Theorem 2.2. *Consider system (2.2). Assume that $\alpha < 0$ and that A_α is stable. The expected value $\mathbb{E}[J]$ of the infinite-time cost J (2.4) is then given by*

$$\mathbb{E}[J] = tr\left(\left(\Sigma_0 - \frac{V}{2\alpha}\right)\bar{X}_\alpha^Q\right). \tag{2.17}$$

Proof. If we examine (2.11) in the limit as $T \to \infty$, then this theorem directly follows. After all, Theorem A.1 implies that, for stable A_α, $e^{2\alpha T}\Sigma(T) \to 0$ as $T \to \infty$. □

Theorem 2.3. *Consider system (2.2). Assume that $\alpha = 0$ and that A is Sylvester. The expected value $\mathbb{E}[J_T]$ of the finite-time cost J_T (2.5) is then given by*

$$\mathbb{E}[J_T] = tr\left((\Sigma_0 - \Sigma(T) + TV)\bar{X}^Q\right). \tag{2.18}$$

Proof. If we consider (2.11) from Theorem 2.1 as $\alpha \to 0$, then this theorem directly follows. After all, we know from l'Hôpital's rule that $\lim_{\alpha\to 0}\frac{1-e^{2\alpha T}}{2\alpha} = -T$. □

2.3.3. THE COST VARIANCE

Next, we derive expressions for the variances $\mathbb{V}[J]$ and $\mathbb{V}[J_T]$, which is the main content of this chapter. If we define $\Delta = \Sigma_0 - X^V$, then $\mathbb{V}[J_T]$ and $\mathbb{V}[J]$ can be found through the following theorems.

Theorem 2.4. *Consider system (2.2). Assume that $\alpha \neq 0$ and that $A_{-\alpha}$, A, A_α and $A_{2\alpha}$ are Sylvester. The variance $\mathbb{V}[J_T]$ of the finite-time cost J_T (2.5) is then given by*

$$\begin{aligned}\mathbb{V}[J_T] = 2tr\left((\Delta\bar{X}_\alpha^Q(T))^2\right) - 2\left(\boldsymbol{\mu}_0^T\bar{X}_\alpha^Q(T)\boldsymbol{\mu}_0\right)^2 + 4tr\Big(X^V Q\Big(X^V\frac{e^{4\alpha T}\bar{X}_{-\alpha}^Q(T) - \bar{X}_\alpha^Q(T)}{4\alpha} \\ + 2X_{2\alpha}^\Delta\bar{X}_\alpha^Q(T) - 2\bar{X}_{3\alpha,\alpha}^{X_{2\alpha}^\Delta e^{A_\alpha^T T}Q}(T)\Big)\Big).\end{aligned} \tag{2.19}$$

Proof. We will start our proof by evaluating $\mathbb{E}[J^2]$. If we write $\boldsymbol{x}(t_1)$ as $\boldsymbol{x}_1$ and $\boldsymbol{x}(t_2)$ as $\boldsymbol{x}_2$, then we have

$$\mathbb{E}[J^2] = \mathbb{E}\left[\int_0^T\int_0^T e^{2\alpha(t_1+t_2)}\boldsymbol{x}_1^T Q\boldsymbol{x}_1\boldsymbol{x}_2^T Q\boldsymbol{x}_2\,dt_2\,dt_1\right]. \tag{2.20}$$

Taking the trace and applying Theorem A.12 gives us

$$\mathbb{E}[J^2] = \int_0^T \int_0^T \Big(\mathrm{tr}\big(e^{2\alpha t_1}\Sigma(t_1)Q\big)\,\mathrm{tr}\big(e^{2\alpha t_2}\Sigma(t_2)Q\big) + 2\mathrm{tr}\big(e^{2\alpha(t_1+t_2)}\Sigma(t_2,t_1)Q\Sigma(t_1,t_2)Q\big) - 2e^{2\alpha(t_1+t_2)}\boldsymbol{\mu}_1^T Q\boldsymbol{\mu}_1\boldsymbol{\mu}_2^T Q\boldsymbol{\mu}_2\Big)\,dt_2\,dt_1, \tag{2.21}$$

where $\boldsymbol{\mu}_1$ equals $\mathbb{E}[\boldsymbol{x}(t_1)] = e^{At_1}\boldsymbol{\mu}_0$ (see Theorem A.1) and similarly for $\boldsymbol{\mu}_2$. There are three terms in the above equation. We will denote them by T_1, T_2 and T_3, respectively. The first term T_1 directly equals $\mathbb{E}[J]^2$ (see Theorem 2.1). This is convenient, because $\mathbb{V}[J] = \mathbb{E}[J^2] - \mathbb{E}[J]^2$, which means that $\mathbb{V}[J]$ equals the remaining two terms $T_2 + T_3$.

The third term T_3 is, according to definition (2.8), equal to

$$T_3 = -2\left(\int_0^T e^{2\alpha t}\boldsymbol{\mu}_0^T e^{A^T t} Q e^{At}\boldsymbol{\mu}_0\,dt\right)^2 = -2\left(\boldsymbol{\mu}_0^T \bar{X}_\alpha^Q(T)\boldsymbol{\mu}_0\right)^2, \tag{2.22}$$

where $\bar{X}_\alpha^Q(T)$ can be evaluated through Theorem A.7. That leaves T_2. To find it, we first have to adjust the integrals. We note that T_2 is symmetric with respect to t_1 and t_2. That is, if we would interchange t_1 and t_2, the integrand would be the same. As a result, we do not have to integrate over all values of t_1 and t_2. We can also only consider all cases where $t_1 < t_2$, integrate over this area, and then multiply the final result by 2. This gives us

$$T_2 = 4\mathrm{tr}\left(\int_0^T \int_{t_1}^T e^{2\alpha(t_1+t_2)}\Sigma(t_2,t_1)Q\Sigma(t_1,t_2)Q\,dt_2\,dt_1\right). \tag{2.23}$$

Now, with $t_1 < t_2$, we can apply Theorem A.3 to substitute for $\Sigma(t_1,t_2)$. If we subsequently expand the brackets, and use the fact that X^V and hence also Δ is symmetric (see Theorem A.5), then the above term turns into

$$T_2 = 4\mathrm{tr}\Big(\int_0^T \int_{t_1}^T e^{2\alpha(t_1+t_2)}\Big(e^{At_2}\Delta e^{A^T t_1} Q e^{At_1}\Delta e^{A^T t_2} Q + e^{A(t_2-t_1)}X^V Q X^V e^{A^T(t_2-t_1)}Q + 2e^{A(t_2-t_1)}X^V Q e^{At_1}\Delta e^{A^T t_2}Q\Big)\,dt_2\,dt_1\Big). \tag{2.24}$$

This expression again has three terms. We call them $T_{2,1}$, $T_{2,2}$ and $T_{2,3}$, respectively. First we find $T_{2,1}$. We can again note that the integrand is symmetric with respect to t_1 and t_2, meaning we can apply the opposite trick of the one we applied at (2.23). This gives us

$$\begin{aligned} T_{2,1} &= 2\mathrm{tr}\left(\int_0^T \int_0^T e^{A_\alpha t_2}\Delta e^{A_\alpha^T t_1} Q e^{A_\alpha t_1}\Delta e^{A_\alpha^T t_2} Q\,dt_2\,dt_1\right) \\ &= 2\mathrm{tr}\left(\left(\int_0^T \Delta e^{A_\alpha^T t} Q e^{A_\alpha t}\,dt\right)^2\right) \\ &= 2\mathrm{tr}\left((\Delta\bar{X}_\alpha^Q(T))^2\right). \end{aligned} \tag{2.25}$$

The next term, $T_{2,2}$, is not symmetric in t_1 and t_2. To bring both integration bounds back to zero, we now substitute t_2 for $t_2 + t_1$. Subsequently interchanging the integrals results

in

$$T_{2,2} = 4\text{tr}\left(\int_0^T \int_0^{T-t_1} e^{2\alpha(2t_1+t_2)} e^{At_2} X^V Q X^V e^{A^T t_2} Q \, dt_2 \, dt_1\right)$$
$$= 4\text{tr}\left(\int_0^T \left(\int_0^{T-t_2} e^{4\alpha t_1} dt_1\right) e^{A_\alpha^T t_2} Q e^{A_\alpha t_2} X^V Q X^V dt_2\right)$$
$$= 4\text{tr}\left(\frac{e^{4\alpha T}\bar{X}_{-\alpha}^Q(T) - \bar{X}_\alpha^Q(T)}{4\alpha} X^V Q X^V\right). \tag{2.26}$$

That leaves $T_{2,3}$, which is the most involved term. We can apply the same substitution and interchanging of integrals to find that $T_{2,3}$ equals

$$T_{2,3} = 8\text{tr}\left(\int_0^T \int_0^{T-t_2} e^{2\alpha(2t_1+t_2)} e^{At_2} X^V Q e^{At_1} \Delta e^{A^T(t_2+t_1)} Q \, dt_1 \, dt_2\right)$$
$$= 8\text{tr}\left(\int_0^T X^V Q X_{2\alpha}^\Delta(T-t_2) e^{A_\alpha^T t_2} Q e^{A_\alpha t_2} dt_2\right). \tag{2.27}$$

Expanding $X_{2\alpha}^\Delta(T-t_2)$ using Theorem A.7 turns this into

$$T_{2,3} = 8\text{tr}\left(X^V Q\left(X_{2\alpha}^\Delta \int_0^T e^{A_\alpha^T t_2} Q e^{A_\alpha t_2} dt_2 - \int_0^T e^{A_{3\alpha}(T-t_2)} X_{2\alpha}^\Delta e^{A_\alpha^T T} Q e^{A_\alpha t_2} dt_2\right)\right)$$
$$= 8\text{tr}\left(X^V Q\left(X_{2\alpha}^\Delta \bar{X}_\alpha^Q(T) - \tilde{X}_{3\alpha,\alpha}^{X_{2\alpha}^\Delta e^{A_\alpha^T T} Q}(T)\right)\right), \tag{2.28}$$

where the final term $\tilde{X}_{3\alpha,\alpha}^{X_{2\alpha}^\Delta e^{A_\alpha^T T} Q}(T)$ can be found through (2.10). If we now merge all terms together, we find the result which we wanted to prove. □

Theorem 2.5. *Consider system (2.2). Assume that $\alpha < 0$ and that A_α is stable. The variance $\mathbb{V}[J]$ of the infinite-time cost J (2.4) is then given by*

$$\mathbb{V}[J] = 2tr\left((\Sigma_0 \bar{X}_\alpha^Q)^2\right) - 2\left(\boldsymbol{\mu}_0^T \bar{X}_\alpha^Q \boldsymbol{\mu}_0\right)^2 + 4tr\left(\left(X_{2\alpha}^{\Sigma_0} - \frac{X_{2\alpha}^V}{4\alpha}\right) \bar{X}_\alpha^Q V \bar{X}_\alpha^Q\right). \tag{2.29}$$

Proof. As $T \to \infty$, $e^{A_\alpha^T T}$ and $e^{4\alpha T}$ become zero, $\bar{X}_\alpha^Q(T)$ becomes $\bar{X}_\alpha^Q$ and hence (2.19) reduces to

$$\mathbb{V}[J] = 2\text{tr}\left((\Delta \bar{X}_\alpha^Q)^2\right) - 2\left(\boldsymbol{\mu}_0^T \bar{X}_\alpha^Q \boldsymbol{\mu}_0\right)^2 + 4\text{tr}\left(\bar{X}_\alpha^Q X^V Q\left(2X_{2\alpha}^\Delta - \frac{X^V}{4\alpha}\right)\right). \tag{2.30}$$

Through an excessive amount of elementary rewritings, using both $Q = -A_\alpha^T \bar{X}_\alpha^Q - \bar{X}_\alpha^Q A_\alpha$ and Theorem A.10, the above can be rewritten to (2.29), which is a slightly more elegant version of the above expression. □

Theorem 2.6. *Consider system (2.2). Assume that $\alpha = 0$ and that A is Sylvester. The variance $\mathbb{V}[J_T]$ of the finite-time cost J_T (2.5) is then given by*

$$\mathbb{V}[J_T] = 2tr\left((\Delta \bar{X}^Q(T))^2\right) - 2\left(\boldsymbol{\mu}_0^T \bar{X}^Q(T) \boldsymbol{\mu}_0\right)^2 + 4tr\Big(X^V Q\Big(X^V\left(T\bar{X}^Q - X^{X^Q}(T)\right)$$
$$+ 2X^\Delta \bar{X}^Q(T) - 2\tilde{X}^{X^\Delta e^{A^T T} Q}(T)\Big)\Big). \tag{2.31}$$

Proof. We can evaluate (2.19) from Theorem 2.4 as $\alpha \to 0$. While doing so, we may use the relation

$$\bar{X}_\alpha^{\bar{X}_{-\alpha}^Q}(T) = \frac{\bar{X}_\alpha^Q(T) - e^{4\alpha T}\bar{X}_{-\alpha}^Q(T)}{4\alpha} + \frac{e^{4\alpha T}-1}{4\alpha}\bar{X}_{-\alpha}^Q, \tag{2.32}$$

which follows from combining Theorems A.7 and A.10. From this, we find through application of l'Hôpital's rule that

$$\lim_{\alpha\to 0} \frac{e^{4\alpha T}\bar{X}_{-\alpha}^Q(T) - \bar{X}_\alpha^Q(T)}{4\alpha} = T\bar{X}^Q - X^{X^Q}(T). \tag{2.33}$$

By using the above relation, the theorem directly follows. □

2.3.4. FINDING $\mathbb{E}[J_T]$ AND $\mathbb{V}[J_T]$ USING MATRIX EXPONENTIALS

The method of using Lyapunov solutions to find $\mathbb{E}[J_T]$ and $\mathbb{V}[J_T]$ has a significant downside: if A or A_α is not Sylvester, the theorems do not hold. By solving integrals using matrix exponentials, according to the methods described in van Loan (1978), we can work around that problem.

Theorem 2.7. *If we define the matrix C as*

$$C = \begin{bmatrix} -A_{2\alpha}^T & Q & 0 & 0 & 0 \\ 0 & A & V & 0 & 0 \\ 0 & 0 & -A^T & Q & 0 \\ 0 & 0 & 0 & A_{2\alpha} & V \\ 0 & 0 & 0 & 0 & -A_{-2\alpha}^T \end{bmatrix}, \tag{2.34}$$

and write e^{CT} as

$$e^{CT} = \begin{bmatrix} C_{11}^e & \cdots & C_{15}^e \\ \vdots & \ddots & \vdots \\ C_{51}^e & \cdots & C_{55}^e \end{bmatrix}, \tag{2.35}$$

then we can find $\mathbb{E}[J_T]$ and $\mathbb{V}[J_T]$ through

$$\mathbb{E}[J_T] = tr\left((C_{44}^e)^T\left(C_{12}^e\Sigma_0 + C_{13}^e\right)\right), \tag{2.36}$$

$$\mathbb{V}[J_T] = 2tr\Big(\left((C_{44}^e)^T\left(C_{12}^e\Sigma_0 + C_{13}^e\right)\right)^2 - 2(C_{44}^e)^T(C_{14}^e\Sigma_0 + C_{15}^e)\Big) - 2\left(\boldsymbol{\mu}_0^T(C_{44}^e)^T C_{12}^e\boldsymbol{\mu}_0\right)^2. \tag{2.37}$$

Proof. We first prove the expression for $\mathbb{E}[J_T]$. If we insert (A.6) into (2.12), we find that

$$\mathbb{E}[J_T] = \mathrm{tr}\Big(\int_0^T e^{2\alpha t}e^{At}\Sigma_0 e^{A^T t}Q\,dt + \int_0^T\int_0^t e^{2\alpha t}e^{A(t-s)}Ve^{A^T(t-s)}Q\,ds\,dt\Big). \tag{2.38}$$

We know from van Loan (1978) that

$$C_{44}^e = e^{A_{2\alpha}T}, \tag{2.39}$$

$$C_{12}^e = \int_0^T e^{-A_{2\alpha}^T(T-t)}Qe^{At}\,dt, \tag{2.40}$$

$$C_{13}^e = \int_0^T\int_0^t e^{-A_{2\alpha}^T(T-t)}Qe^{A(t-s)}Ve^{-A^T s}\,ds\,dt. \tag{2.41}$$

From this (2.36) directly follows. Proving the expression for $\mathbb{V}[J_T]$ is done similarly, but with more bookkeeping. First of all, C^e_{14} equals (see van Loan (1978))

$$C^e_{14} = \int_0^T \int_0^t \int_0^s e^{-A^T_{2\alpha}(T-t)} Q e^{A(t-s)} V e^{-A^T(s-r)} Q e^{A_{2\alpha} r}\, dr\, ds\, dt, \tag{2.42}$$

with a similar expression for C^e_{15}. Next, we will find the terms T_3 (see (2.22)) and T_2 (see (2.23)), which together equal $\mathbb{V}[J_T]$. We can directly see from (2.22) that T_3 equals

$$T_3 = -2\left(\boldsymbol{\mu}_0^T (C^e_{44})^T C^e_{12} \boldsymbol{\mu}_0\right)^2. \tag{2.43}$$

Then we consider T_2 from (2.23). Instead of applying (A.6), we now use

$$\Sigma(t_1, t_2) = e^{At_1}\Sigma_0 e^{A^T t_2} + \int_0^{\min(t_1,t_2)} e^{A(t_1-s)} V e^{A^T(t_2-s)}\, ds, \tag{2.44}$$

which is derived in an identical way. For ease of notation, we write $\Sigma(t_1, t_2) = \Sigma_a + \Sigma_b$, with Σ_a and Σ_b the two parts in the above expression. Inserting $\Sigma(t_1, t_2)$ into (2.23) then gives

$$T_2 = 2\mathrm{tr}\Big(\int_0^T \int_0^T e^{2\alpha(t_1+t_2)}\left(\Sigma_a^T Q \Sigma_a Q + 2\Sigma_a^T Q \Sigma_b Q + \Sigma_b^T Q \Sigma_b Q\right) dt_2\, dt_1\Big). \tag{2.45}$$

The first term $T_{2,aa}$ here equals

$$\begin{aligned} T_{2,aa} &= 2\mathrm{tr}\left(\int_0^T \int_0^T e^{2\alpha(t_1+t_2)} e^{At_2}\Sigma_0 e^{A^T t_1} Q e^{At_1}\Sigma_0 e^{A^T t_2}\, dt_2\, dt_1\right) \\ &= 2\mathrm{tr}\left(\left(\int_0^T e^{2\alpha t} e^{A^T t} Q e^{At}\Sigma_0\, dt\right)^2\right) \\ &= 2\mathrm{tr}\left(\left((C^e_{44})^T C^e_{12}\Sigma_0\right)^2\right) = T_{2,aa}. \end{aligned} \tag{2.46}$$

The second term $T_{2,ab}$ is given by

$$T_{2,ab} = 4\mathrm{tr}\Big(\int_0^T \int_0^T \int_0^{\min(t_1,t_2)} e^{2\alpha(t_1+t_2)} e^{At_2}\Sigma_0 e^{A^T t_1} Q e^{A(t_1-s)} V e^{A^T(t_2-s)} Q\, ds\, dt_2\, dt_1\Big). \tag{2.47}$$

We want the integration order to be $dt_2\, ds\, dt_1$. If we note that the integration area is described by $0 \le s \le (t_1, t_2) \le T$, we can reorder the integrals. That is,

$$\begin{aligned} T_{2,ab} &= 4\mathrm{tr}\Big(\int_0^T \int_0^{t_1} \int_s^T \ldots dt_2\, ds\, dt_1\Big) \\ &= 4\mathrm{tr}\Big(\int_0^T \int_0^{t_1} \int_0^T \ldots dt_2\, ds\, dt_1 - \int_0^T \int_0^{t_1} \int_0^s \ldots dt_2\, ds\, dt_1\Big). \end{aligned} \tag{2.48}$$

We now have two integrals, but we can solve both. If we split up the first one and rewrite the second one, we get

$$\begin{aligned} T_{2,ab} &= 4\mathrm{tr}\Big(\Big(\int_0^T \int_0^{t_1} e^{2\alpha t_1} e^{A^T t_1} Q e^{A(t_1-s)} V e^{-A^T s}\, ds\, dt_1\Big)\Big(\int_0^T e^{2\alpha t_2} e^{A^T t_2} Q e^{At_2}\, dt_2\Big)\Sigma_0 \\ &\qquad - \int_0^T \int_0^{t_1} \int_0^s e^{2\alpha(t_1+t_2)} e^{A^T t_1} Q e^{A(t_1-s)} V e^{A^T(t_2-s)} Q e^{At_2}\Sigma_0\, dt_2\, ds\, dt_1\Big) \\ &= 4\mathrm{tr}\left((C^e_{44})^T C^e_{13}(C^e_{44})^T C^e_{12}\Sigma_0 - (C^e_{44})^T C^e_{14}\Sigma_0\right). \end{aligned} \tag{2.49}$$

Finally there is $T_{2,bb}$. We first concern ourselves with the integration order and limits. By rearranging integrals, and by using the symmetry between t_1 and t_2 as well as between s_1 and s_2, we can find that

$$\begin{aligned} T_{2,bb} &= 2\mathrm{tr}\Big(\int_0^T\int_0^T\int_0^{\min(t_1,t_2)}\int_0^{\min(t_1,t_2)} \dots ds_2\, ds_1\, dt_2\, dt_1\Big) \\ &= 2\mathrm{tr}\Big(\int_0^T\int_0^{t_2}\int_0^T\int_0^{t_1} \dots ds_1\, dt_2\, ds_2\, dt_1 - 2\int_0^T\int_0^{t_2}\int_0^{s_1}\int_0^{t_1} \dots ds_1\, dt_2\, ds_2\, dt_1\Big). \end{aligned} \tag{2.50}$$

After inserting the integrand, we can rewrite this to

$$\begin{aligned} T_{2,bb} &= 2\mathrm{tr}\Big(\Big(\int_0^T\int_0^t e^{2\alpha t} e^{A^T t} Q e^{A(t-s)} V e^{-A^T s}\, ds\, dt\Big)^2 \\ &\quad - 2\int_0^T\int_0^{t_2}\int_0^{s_1}\int_0^{t_1} e^{2\alpha(t_1+t_2)} e^{A^T(t_1-s_1)} Q e^{A(t_1-s_2)} V e^{A^T(t_2-s_2)} Q e^{A(t_2-s_1)} V\, ds_1\, dt_2\, ds_2\, dt_1\Big) \\ &= 2\mathrm{tr}\Big(\big((C_{44}^e)^T C_{13}^e\big)^2 - 2(C_{44}^e)^T C_{15}^e\Big). \end{aligned} \tag{2.51}$$

By combining all the results, we wind up with (2.37). □

So now we have two methods of finding $\mathbb{E}[J_T]$ and $\mathbb{V}[J_T]$. Which one is better? This mainly depends on the time T. Our experiments have shown that, for small times T, using matrix exponentials results in a better numerical accuracy than using Lyapunov solutions, but for large T the situation is exactly the opposite, and the numerical accuracy of the matrix exponential method quickly deteriorates. Similar results have been obtained by Wahlström et al. (2014), which examines the numerical accuracy of both algorithms when finding $X^Q(T)$.

2.4. APPLICATION TO AN LQG SYSTEM

So far we have only considered systems of the form (2.2), but in LQG systems there are also input and output signals. However, in that case we can always rewrite the system on the form (2.2). In this section we show how to do this. For more details we refer to Anderson and Moore (1990), Skogestad and Postlethwaite (2005), Bosgra et al. (2008), Åström (1970).

First, we consider a system $\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t) + \boldsymbol{v}(t)$ with input. Its corresponding cost function equals

$$J = \int_0^\infty e^{2\alpha t}(\boldsymbol{x}^T(t)Q\boldsymbol{x}(t) + \boldsymbol{u}^T(t)R\boldsymbol{u}(t))\, dt. \tag{2.52}$$

It is well-known in literature (see for instance Kalman (1960)) that the optimal control law minimizing $\mathbb{E}[J]$ is a linear control law $\boldsymbol{u}(t) = -F\boldsymbol{x}(t)$. If we assume that $Q = Q^T \geq 0$ and $R = R^T > 0$, then the optimal gain matrix F equals

$$F = R^{-1}B^T\hat{X}_\alpha, \tag{2.53}$$

with $\hat{X}_\alpha$ the solution to the algebraic Riccati equation

$$A_\alpha^T\hat{X}_\alpha + \hat{X}_\alpha A_\alpha + Q - \hat{X}_\alpha B R^{-1} B^T \hat{X}_\alpha = 0. \tag{2.54}$$

For this optimal gain matrix F (and for any other matrix F) the system and cost function can be written as

$$\dot{\boldsymbol{x}}(t) = (A - BF)\boldsymbol{x}(t) + \boldsymbol{v}(t) = \hat{A}\boldsymbol{x}(t) + \boldsymbol{v}(t), \tag{2.55}$$

$$J = \int_0^\infty e^{2\alpha t}\boldsymbol{x}^T(t)\left(Q + F^T R F\right)\boldsymbol{x}(t)\,dt = \int_0^\infty e^{2\alpha t}\boldsymbol{x}^T(t)\hat{Q}\boldsymbol{x}(t)\,dt. \tag{2.56}$$

This shows that the system is now in our original form (2.2).

A similar reduction can be performed when we are dealing with a noisy output equation $\boldsymbol{y}(t) = C\boldsymbol{x}(t) + \boldsymbol{w}(t)$, where $\boldsymbol{w}(t)$ is zero-mean Gaussian white noise with intensity W. To deal with this output equation, we take a state estimate $\hat{\boldsymbol{x}}(t)$ and update it through

$$\dot{\hat{\boldsymbol{x}}}(t) = A\hat{\boldsymbol{x}}(t) + B\boldsymbol{u}(t) + K(\boldsymbol{y}(t) - C\hat{\boldsymbol{x}}(t)). \tag{2.57}$$

To minimize the state estimation error $\boldsymbol{e}(t) = \hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)$, we need to choose the observer gain K equal to

$$K = EC^T W^{-1}, \tag{2.58}$$

where E is the solution to

$$AE + EA^T + V - EC^T W^{-1} CE = 0. \tag{2.59}$$

We need this state estimate in a new optimal control law $\boldsymbol{u} = -F\hat{\boldsymbol{x}}$. This reduces the system equations to

$$\begin{bmatrix}\dot{\boldsymbol{x}}\\ \dot{\hat{\boldsymbol{x}}}\end{bmatrix} = \begin{bmatrix}A - BF & -BF\\ KC & A - BF - KC\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ \hat{\boldsymbol{x}}\end{bmatrix} + \begin{bmatrix}\boldsymbol{v}\\ K\boldsymbol{w}\end{bmatrix}, \tag{2.60}$$

which is again of the form we have seen earlier, albeit with a somewhat larger state vector. Because of this, all the equations that were originally developed for system (2.2) are applicable to LQG systems as well.

2.5. NUMERICAL EVALUATION

In this section we look at an example of how to apply the derived equations. In literature, researchers almost always use the controller that minimizes the expected value of the cost. This is done irrespective of the variance of the cost. But if the goal is to keep the cost below a certain threshold, then this may not be the best approach.

Consider the two-state system

$$\dot{\boldsymbol{x}} = \begin{bmatrix}1 & 0\\ 1/20 & 1\end{bmatrix}\boldsymbol{x} + \begin{bmatrix}1\\ 0\end{bmatrix}\boldsymbol{u} + \boldsymbol{v}, \tag{2.61}$$

where we will apply $Q = I$, $R = I$ and $\alpha = -0.8$ in the cost function. As control law we use $\boldsymbol{u} = -F\boldsymbol{x}$. We assume that the state $\boldsymbol{x}$ is fully known, and hence only F needs to be chosen. In practice this is often not the case and only a noisy measurement $\boldsymbol{y}$ will be available. To solve this, we can apply the theory from Section 2.4 and subsequently choose the observer gain K along with F. However, this process is identical to choosing F. So for simplicity of presentation, we only consider selecting F.

The optimal control matrix follows from (2.53) as $F_{\text{opt}} = \begin{bmatrix} 1.6 & 9.9 \end{bmatrix}$. It minimizes $\mathbb{E}[J]$ at $\mathbb{E}[J(F_{\text{opt}})] = 154.4$. However, we can also minimize $\mathbb{V}[J]$ using a basic gradient descent method. This gives the minimum-variance control matrix $F_{\text{mv}} = \begin{bmatrix} 4.4 & 30.0 \end{bmatrix}$ with mean cost $\mathbb{E}[J(F_{\text{mv}})] = 187.5$. This mean cost is significantly larger than $\mathbb{E}[J]_{\text{opt}}$, making it seem as if this is a significantly worse control matrix.

However, now suppose that we do not care so much about the mean cost. All we want is to reduce the probability that the cost J is above a certain threshold $\bar{J}$. That is, we aim to minimize $p(J > \bar{J})$ where we use $\bar{J} = 1\,500$, which is roughly ten times the mean. Using 250 000 numerical simulations, with $T = 20$ s and $dt = 0.01$ s, we have found that

$$p(J(F_{\text{opt}}) > \bar{J}) \approx 0.091\%, \tag{2.62}$$

$$p(J(F_{\text{mv}}) > \bar{J}) \approx 0.059\%. \tag{2.63}$$

Hence the optimal controller has more than half as many threshold-violating cases as the minimum-variance control law, which is a significantly worse result.

3

Optimal control of discounted-cost LQG systems

Abstract — The linear quadratic Gaussian control paradigm is well-known in literature. The strategy of minimizing the cost function is available, both for the case where the state is fully known and where it is estimated through an observer. The situation is different when the cost function has an exponential discount factor, also known as a prescribed degree of stability. In this case, the optimal control strategy is only available when the state is fully known. This chapter builds onward from that result, deriving an optimal control strategy when working with an estimated state. Expressions for the resulting optimal expected cost are also given. The result is verified through an experimental validation.

The previous chapter considered autonomous linear systems, and how the cost function was distributed for these systems. In this chapter, based on Bijl and Schön (2017), we consider the non-autonomous linear system

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= A\boldsymbol{x}(t) + B\boldsymbol{u}(t) + \boldsymbol{v}(t), \\ \boldsymbol{y}(t) &= C\boldsymbol{x}(t) + D\boldsymbol{u}(t) + \boldsymbol{w}(t), \end{aligned} \tag{3.1}$$

with $\boldsymbol{x}$ the state, $\boldsymbol{u}$ the input, $\boldsymbol{y}$ the output, $\boldsymbol{v}$ and $\boldsymbol{w}$ Gaussian white noise with respective intensities V and W, and A, B, C and D the system matrices. We assume that the initial state $\boldsymbol{x_0}$ is unknown but distributed according to a Gaussian distribution with $\boldsymbol{\mu_0} = \mathbb{E}[\boldsymbol{x_0}]$ and $\Sigma_0 = \mathbb{E}\left[\boldsymbol{x_0}\boldsymbol{x_0}^T\right]$. Note that Σ_0 is *not* the variance of $\boldsymbol{x_0}$.

Our goal is to control system (3.1) such as to minimize the discounted (exponential) quadratic cost function

$$J(T) = \int_0^T e^{2\alpha t}\left(\boldsymbol{x}^T(t)Q\boldsymbol{x}(t) + \boldsymbol{u}^T(t)R\boldsymbol{u}(t)\right) dt, \tag{3.2}$$

with $J(T)$ the cost, α the discount exponent/prescribed degree of stability, and $Q \geq 0$ and $R > 0$ weight matrices. In particular, we will optimize the infinite-time expected cost $E[J]$,

with

$$J = \lim_{T\to\infty} J(T). \tag{3.3}$$

This integral does not always converge, but the theorems in this chapter will specify when it does.

Especially when the model may be inaccurate, as is the case for wind turbine applications, prescribing such a degree of stability can be valuable. In this chapter we derive the optimal controller and observer gains for the continuous-time linear system (3.1) such that the expected cost $\mathbb{E}[J]$ given in (3.3) is minimized.

We start this chapter off with a brief literature overview (Section 3.1) investigating the status of literature on this subject and whether this particular problem hasn't already been solved. We then examine a few already known Theorems in Section 3.2 and expand on this in Section 3.3. We verify the results with an experiment in Section 3.4 and state conclusions in Section 3.5.

3.1. LITERATURE OVERVIEW

Linear-Quadratic-Gaussian (LQG) systems—linear systems with a quadratic cost function subject to Gaussian noise—have been thoroughly investigated in the past. This was especially true near the 1960s, with for instance the publication of the Kalman filter Kalman (1960), Kalman and Bucy (1961).

The discoveries from the decades afterwards have been summarized in various textbooks. Examples include the books by (Åström, 1970, Chapter 7), (Kwakernaak and Sivan, 1972, Chapter 5), (Grimble and Johnson, 1988, Chapter 1), (Anderson and Moore, 1990, Chapters 3, 8), (Stengel, 1994, Chapter 6), (Trentelman et al., 2001, Chapter 10), (Skogestad and Postlethwaite, 2005, Chapter 9) and (Bosgra et al., 2008, Chapter 4). All these books examine the non-discounted cost function (with $\alpha = 0$). Only Anderson and Moore (1990) (Section 3.5) also looks at the discounted cost function, presenting results from an earlier paper Anderson and Moore (1969). Here it was shown that discounting the cost function is equivalent to prescribing a degree of stability.

The prescribed degree of stability is actually a relevant problem in that it is a generalization of the regular LQG paradigm with the non-discounted cost function. There is also a variety of applications of this idea, such as fault tolerant flight control Ciubotaru et al. (2013), spacecraft guidance Meng et al. (2016) and robot manipulators Takahashi and Sato (2016). However, to the best of the authors knowledge there are still fundamental properties remaining to be established and our main contribution in this paper is to provide one of those. The work Anderson and Moore (1969) only examined the situation where the state is assumed to be known. If the state can only be observed through noisy output measurements—a familiar problem for the non-discounted cost function—then no work is available on jointly optimizing the controller and state estimator. The closest is the work by Rusnak (2016), who strived to set up a state estimator with minimal mean squared error, given a prescribed convergence rate. However, this work ignores uncertainty in the initial state and does not examine the problem of jointly optimizing the controller and observer gains. In fact, no mention is made on whether the separation principle still holds when using the discounted cost function. Hence, the problem of jointly optimizing the controller and observer gains, subject to a discounted cost function, appears to be an

open problem.

3.2. A SUMMARY OF KNOWN THEOREMS

Before we start with the actual problem, we examine the cases for which the optimal control law is already known. When doing so, we also aim to give the corresponding expected cost, using expressions from Chapter 2. We begin with the situation where the state is fully known, first examining the non-discounted cost ($\alpha = 0$) and moving on to the discounted cost ($\alpha \neq 0$). Then we add an observer and do the same.

3.2.1. THE NON-DISCOUNTED CASE WITH FULLY KNOWN STATE

We examine the non-discounted case ($\alpha = 0$) where the state $\boldsymbol{x}(t)$ is fully known (for instance when $C = I$ and $W = 0$). The solution of this problem is well-known.

Theorem 3.1. *Consider system (3.1), where the state is assumed known. If (A, B) is stabilizable, then the optimal control law minimizing the expected non-discounted cost $\mathbb{E}[J]$ (i.e., with $\alpha = 0$) is a linear control law $\boldsymbol{u}(t) = -F\boldsymbol{x}(t)$, where*

$$F = R^{-1}B^T X, \tag{3.4}$$

and X is the solution to the Riccati equation

$$A^T X + XA + Q - XBR^{-1}B^T X = 0. \tag{3.5}$$

When $V = 0$, the corresponding expected cost equals

$$\mathbb{E}[J] = \mathbb{E}\left[\boldsymbol{x_0}^T X \boldsymbol{x_0}\right] = tr(X\Sigma_0). \tag{3.6}$$

When $V \neq 0$, then $\mathbb{E}[J(T)] \to \infty$, but the steady-state cost rate equals

$$\lim_{T\to\infty} \frac{d\mathbb{E}[J(T)]}{dT} = tr(XV). \tag{3.7}$$

Proof. See any of the aforementioned books; for example Kwakernaak and Sivan (1972), Theorem 3.9. □

It is important to note that the noise V does not affect the optimal control law whatsoever. The control strategy can be summarized as 'Bring the state back to zero as efficiently as possible, from any disturbance that may occur.' Which exact disturbance occurs is irrelevant here.

3.2.2. THE DISCOUNTED CASE WITH FULLY KNOWN STATE

Next, we add the discount exponent α. Note that α can be both positive (a prescribed degree of stability) and negative (a discount exponent), but for ease of writing we always call it a discount exponent. We also define $A_\alpha = A + \alpha I$.

Theorem 3.2. *Consider system (3.1), where the state is assumed known. If (A_α, B) is stabilizable, then the optimal control law minimizing the expected discounted cost $\mathbb{E}[J]$ is a linear control law $\boldsymbol{u}(t) = -F_\alpha \boldsymbol{x}(t)$, where*

$$F_\alpha = R^{-1}B^T X_\alpha, \tag{3.8}$$

and X_α is the solution to the Riccati equation

$$A_\alpha^T X_\alpha + X_\alpha A_\alpha + Q - X_\alpha B R^{-1} B^T X_\alpha = 0. \tag{3.9}$$

The corresponding expected cost (for both zero and nonzero V) when $\alpha < 0$ equals

$$\mathbb{E}[J] = tr\left(\left(\frac{-X_\alpha}{2\alpha}\right)(V - 2\alpha\Sigma_0)\right). \tag{3.10}$$

When $\alpha \geq 0$, then $\mathbb{E}[J(T)] \to \infty$ as $T \to \infty$.

Proof. The key insight (see Anderson and Moore (1990), Section 3.5) is to define

$$\begin{aligned} \tilde{\boldsymbol{x}}(t) &= e^{\alpha t}\boldsymbol{x}(t), \\ \tilde{\boldsymbol{u}}(t) &= e^{\alpha t}\boldsymbol{u}(t), \\ \tilde{\boldsymbol{v}}(t) &= e^{\alpha t}\boldsymbol{v}(t). \end{aligned} \tag{3.11}$$

Note that $\tilde{\boldsymbol{v}}(t)$ is Gaussian white noise with a time-varying intensity $\tilde{V}(t) = e^{2\alpha t}V$. The time-derivative of $\tilde{\boldsymbol{x}}(t)$ now follows as

$$\dot{\tilde{\boldsymbol{x}}}(t) = A_\alpha \tilde{\boldsymbol{x}}(t) + B\tilde{\boldsymbol{u}}(t) + \tilde{\boldsymbol{v}}(t), \tag{3.12}$$

where $\tilde{\boldsymbol{x}}(0) = \boldsymbol{x}(0)$. At the same time, expression (3.2) for the cost $J(T)$ turns into

$$J(T) = \int_0^T \left(\tilde{\boldsymbol{x}}^T(t) Q \tilde{\boldsymbol{x}}(t) + \tilde{\boldsymbol{u}}^T(t) R \tilde{\boldsymbol{u}}(t)\right) dt. \tag{3.13}$$

We can directly apply Theorem 3.1 to this situation, proving the Theorem except for (3.10). This expression follows directly from Theorem 2.2. □

It is important to note what 'optimality' means here. After all, we have process noise continuously exciting the system. If $\alpha > 0$, then the cost J will be infinite, regardless of the control law used. A control law now is optimal when the corresponding expected cost $\mathbb{E}[J(T)]$ is smaller than or equal to the expected cost $\mathbb{E}[J'(T)]$ corresponding to any other control law, in the limit $T \to \infty$.

An interesting property of the above theorem is that the control law ensures that all the closed-loop system eigenvalues are smaller than $-\alpha$. (See Anderson and Moore (1990), Section 3.5.) This is why α (when positive) is known as the prescribed degree of stability. We do not directly place the eigenvalues like Kučera and Cigler (2010), but we do prescribe them to satisfy $\lambda_i < -\alpha$.

Generally, positive values of α result in more aggressive controllers (higher gains) while negative values of α result in more lazy controllers (lower gains). Roughly put, when α is negative, the controller does not bother putting in a large effort now to 'fix' future states, because these future states will hardly matter anyway.

3.2.3. THE NON-DISCOUNTED CASE WITH UNKNOWN STATE

We now consider the complete system (3.1), where the state is not anymore assumed known. We do for now assume that $\alpha = 0$. The common solution in literature to deal with the unknown state is to set up an observer with state estimate $\hat{\boldsymbol{x}}$, which is updated according to

$$\dot{\hat{\boldsymbol{x}}}(t) = A\hat{\boldsymbol{x}}(t) + B\boldsymbol{u}(t) + K\big(\boldsymbol{y}(t) - C\hat{\boldsymbol{x}}(t) - D\boldsymbol{u}(t)\big), \tag{3.14}$$

subject to some initial state estimate $\hat{\boldsymbol{x}}(0)$. We also define the state estimation error as $\boldsymbol{e}(t) = \boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)$. To minimize this error (i.e., its variance) we can use the following Theorem.

Theorem 3.3. *Consider system (3.1). If (A, C) is detectable, then the optimal observer gain minimizing the steady-state error covariance is*

$$K = EC^T W^{-1}, \tag{3.15}$$

where E is the optimal steady-state error covariance, found through

$$AE + EA^T + V - EC^T W^{-1} CE = 0. \tag{3.16}$$

Proof. This is the famous Kalman-Bucy filter from Kalman and Bucy (1961). A proof can also be found in Kwakernaak and Sivan (1972), Theorem 4.5. □

Theorem 3.4. *Consider system (3.1). If (A, B) is stabilizable and (A, C) is detectable, then the optimal control law minimizing the expected non-discounted cost (i.e., with $\alpha = 0$) is a linear control law $\boldsymbol{u}(t) = -F\hat{\boldsymbol{x}}(t)$, with F given by (3.4), $\hat{\boldsymbol{x}}(t)$ following from (3.14) and the observer gain K taken as (3.15). The resulting expected steady-state cost rate is given by*

$$\lim_{T\to\infty} \frac{d\mathbb{E}[J(T)]}{dT} = tr\big(XKWK^T\big) + tr(EQ) = tr(XV) + tr\big(EF^T RF\big), \tag{3.17}$$

with X the solution of (3.5) and E the solution of (3.16).

Proof. The optimal controller and observer gains follow from the separation principle. See for instance Kwakernaak and Sivan (1972), Theorem 5.4. This leaves us only with the proof for the expected steady-state cost rate. Though a proof is also given by Kwakernaak and Sivan (1972), we derive one for ourselves here, so that we can use a similar proof when dealing with the discount exponent α.

To find it, we first note that our system, with the control law and the observer, can be written in an adjusted notation as

$$\begin{bmatrix} \dot{\hat{\boldsymbol{x}}}(t) \\ \dot{\boldsymbol{e}}(t) \end{bmatrix} = \begin{bmatrix} A - BF & -KC \\ 0 & A - KC \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}}(t) \\ \boldsymbol{e}(t) \end{bmatrix} + \begin{bmatrix} K\boldsymbol{w}(t) \\ K\boldsymbol{w}(t) + \boldsymbol{v}(t) \end{bmatrix}. \tag{3.18}$$

If we (re-)define

$$\tilde{\boldsymbol{x}}(t) = \begin{bmatrix} \hat{\boldsymbol{x}}(t) \\ \boldsymbol{e}(t) \end{bmatrix}, \qquad \tilde{A} = \begin{bmatrix} A - BF & -KC \\ 0 & A - KC \end{bmatrix},$$
$$\tilde{V} = \begin{bmatrix} KWK^T & KWK^T \\ KWK^T & KWK^T + V \end{bmatrix}, \qquad \tilde{Q} = \begin{bmatrix} Q + F^T RF & -Q \\ -Q & Q \end{bmatrix}, \tag{3.19}$$

then we can apply Theorem 2.3 to find the expected steady-state cost rate. This becomes (also through Theorem A.9)

$$\lim_{T\to\infty}\frac{d\mathbb{E}[J(T)]}{dT}=\mathrm{tr}\left(X_{\tilde{A}}^{\tilde{V}}\tilde{Q}\right)=\mathrm{tr}\left(\tilde{V}\bar{X}_{\tilde{A}}^{\tilde{Q}}\right). \tag{3.20}$$

where we use the notation from Section 2.3.1 for $X_{\tilde{A}}^{\tilde{V}}$ and $\bar{X}_{\tilde{A}}^{\tilde{Q}}$. So these parameters per definition are the solutions of the Lyapunov equations

$$\tilde{A}X_{\tilde{A}}^{\tilde{V}}+X_{\tilde{A}}^{\tilde{V}}\tilde{A}^T+\tilde{V}=0, \tag{3.21}$$

$$\tilde{A}^T\bar{X}_{\tilde{A}}^{\tilde{Q}}+\bar{X}_{\tilde{A}}^{\tilde{Q}}\tilde{A}+\tilde{Q}=0. \tag{3.22}$$

We will initially use the first of these two expression, expanding (3.21) into

$$(A-BF)X_{\tilde{A},11}^{\tilde{V}}-KCX_{\tilde{A},12}^{\tilde{V}}+X_{\tilde{A},11}^{\tilde{V}}(A-BF)^T-X_{\tilde{A},12}^{\tilde{V}}C^TK^T+KWK^T=0, \tag{3.23}$$

$$(A-BF)X_{\tilde{A},12}^{\tilde{V}}-KCX_{\tilde{A},22}^{\tilde{V}}+X_{\tilde{A},12}^{\tilde{V}}(A-KC)^T+KWK^T=0, \tag{3.24}$$

$$(A-KC)X_{\tilde{A},22}^{\tilde{V}}+X_{\tilde{A},22}^{\tilde{V}}(A-KC)^T+KWK^T+V=0. \tag{3.25}$$

For the given values of F and K, we can directly find[1] that $X_{\tilde{A},12}^{\tilde{V}}=0$. Subsequently, the solution for (3.23) equals $X_{\tilde{A},11}^{\tilde{V}}=X$ (since (3.23) then equals (3.5)) and the solution for (3.25) equals $X_{\tilde{A},22}^{\tilde{V}}=E$ (since (3.25) then equals (3.16)). It immediately follows that

$$\lim_{t\to\infty}\frac{d\mathbb{E}[J]}{dt}=\mathrm{tr}\left(X_{\tilde{A}}^{\tilde{V}}\tilde{Q}\right)=\mathrm{tr}\left(X\left(Q+F^TRF\right)+EQ\right). \tag{3.26}$$

Through applying Theorem A.9, we directly find the first half of expression (3.17). To find the second half, we need to apply the above methodology in an identical way to (3.22) instead of (3.21). When doing so, we complete our proof. □

3.3. OPTIMIZING THE DISCOUNTED COST FUNCTION

In this section we derive the main result: the optimal controller/observer gains for the complete system. The following Theorem shows that the separation principle still holds, if in an adjusted form, when using the discounted cost function.

Theorem 3.5. *Consider system (3.1). If (A_α,B) is stabilizable and (A_α,C) is detectable, then the optimal control law minimizing the expected discounted cost $\mathbb{E}[J]$ is a linear control law $\boldsymbol{u}(t)=-F_\alpha\hat{\boldsymbol{x}}(t)$, with F_α given by (3.8) and X_α by (3.9). Identically to (3.14), $\hat{\boldsymbol{x}}(t)$ is given by the observer*

$$\dot{\hat{\boldsymbol{x}}}(t)=A\hat{\boldsymbol{x}}(t)+B\boldsymbol{u}(t)+K_\alpha\left(\boldsymbol{y}(t)-C\hat{\boldsymbol{x}}(t)-D\boldsymbol{u}(t)\right), \tag{3.27}$$

[1] This actually also follows from the separation principle. We can note that $X_{\tilde{A},12}^{\tilde{V}}=\lim_{t\to\infty}\mathbb{E}\left[\hat{\boldsymbol{x}}(t)\boldsymbol{e}^T(t)\right]$, and the separation principle implies that this quantity becomes zero; see Anderson and Moore (1990), Section 8.2.

where $\hat{\boldsymbol{x}}_{\mathbf{0}}$ is set to $\boldsymbol{\mu}_{\mathbf{0}}$, where the observer gain K_α is given by

$$K_\alpha = E_\alpha C^T W^{-1} \tag{3.28}$$

and where E_α is the solution to the Riccati equation

$$A_\alpha E_\alpha + E_\alpha A_\alpha^T + \left(V - 2\alpha\left(\Sigma_0 - \boldsymbol{\mu}_{\mathbf{0}}\boldsymbol{\mu}_{\mathbf{0}}{}^T\right)\right) - E_\alpha C^T W^{-1} C E_\alpha = 0. \tag{3.29}$$

The corresponding expected cost for $\alpha < 0$ equals

$$\begin{aligned}\mathbb{E}[J] &= tr\left(\left(-\frac{X_\alpha}{2\alpha}\right)\left(K_\alpha W K_\alpha^T - 2\alpha\boldsymbol{\mu}_{\mathbf{0}}\boldsymbol{\mu}_{\mathbf{0}}{}^T\right)\right) + tr\left(\left(-\frac{E_\alpha}{2\alpha}\right)Q\right)\\ &= tr\left(\left(-\frac{X_\alpha}{2\alpha}\right)(V - 2\alpha\Sigma_0)\right) + tr\left(\left(-\frac{E_\alpha}{2\alpha}\right)F_\alpha^T R F_\alpha\right).\end{aligned} \tag{3.30}$$

When $\alpha \geq 0$, then $\mathbb{E}[J(T)] \to \infty$ as $T \to \infty$.

Proof. This Theorem follows from Theorem 3.4. To see how, we need to carefully note the differences between the two problems.

Consider the adjusted system (3.18). First of all, since $\hat{\boldsymbol{x}}_{\mathbf{0}} = \boldsymbol{\mu}_{\mathbf{0}}$, the initial state satisfies

$$\begin{aligned}\tilde{\boldsymbol{\mu}}_{\mathbf{0}} &= \mathbb{E}[\tilde{\boldsymbol{x}}_{\mathbf{0}}] = \mathbb{E}\begin{bmatrix}\hat{\boldsymbol{x}}_{\mathbf{0}}\\ \boldsymbol{e}_{\mathbf{0}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\mu}_{\mathbf{0}}\\ \mathbf{0}\end{bmatrix},\\ \tilde{\Sigma}_0 &= \mathbb{E}\left[\tilde{\boldsymbol{x}}_{\mathbf{0}}\tilde{\boldsymbol{x}}_{\mathbf{0}}{}^T\right] = \mathbb{E}\begin{bmatrix}\hat{\boldsymbol{x}}_{\mathbf{0}}\hat{\boldsymbol{x}}_{\mathbf{0}}{}^T & \hat{\boldsymbol{x}}_{\mathbf{0}}\hat{\boldsymbol{e}}_{\mathbf{0}}{}^T\\ \hat{\boldsymbol{e}}_{\mathbf{0}}\hat{\boldsymbol{x}}_{\mathbf{0}}{}^T & \hat{\boldsymbol{e}}_{\mathbf{0}}\hat{\boldsymbol{e}}_{\mathbf{0}}{}^T\end{bmatrix} = \begin{bmatrix}\boldsymbol{\mu}_{\mathbf{0}}\boldsymbol{\mu}_{\mathbf{0}}{}^T & 0\\ 0 & \Sigma_0 - \boldsymbol{\mu}_{\mathbf{0}}\boldsymbol{\mu}_{\mathbf{0}}{}^T\end{bmatrix}.\end{aligned} \tag{3.31}$$

According to Theorem 3.2, the expected cost equals

$$\mathbb{E}[J] = \mathrm{tr}\left(\left(\frac{-\tilde{X}_\alpha}{2\alpha}\right)\left(\tilde{V} - 2\alpha\tilde{\Sigma}_0\right)\right). \tag{3.32}$$

This is the quantity that we need to optimize, contrary to Theorem 3.4 where we needed to optimize $\mathrm{tr}\left(\tilde{X}\tilde{V}\right)$. To turn one situation into the other, we hence need to substitute $\tilde{A}$ by $\tilde{A}_\alpha$, which is equivalent to replacing A by A_α. Similarly, we must substitute $\tilde{V}$ for $\left(\tilde{\Sigma}_0 - \frac{\tilde{V}}{2\alpha}\right)$, which is equivalent to replacing W by $\frac{-W}{2\alpha}$ and V by $\left(\Sigma_0 - \boldsymbol{\mu}_{\mathbf{0}}\boldsymbol{\mu}_{\mathbf{0}}{}^T - \frac{V}{2\alpha}\right)$. In addition, due to the top left term of $\tilde{\Sigma}_0$, expression (3.23) also gets an extra term $\boldsymbol{\mu}_{\mathbf{0}}\boldsymbol{\mu}_{\mathbf{0}}{}^T$. After these replacements and adjustments, the problems are identical. Elementary algebra subsequently turns Theorem 3.4 into the desired result. □

It is interesting to note that, for the discounted cost, the optimal controller gains stay the same as compared to Theorem 3.2, and Theorem 3.2 was just a minor adjustment with respect to Theorem 3.1. The controller still trades off aggressively controlling the state (for high Q) with being frugal at applying input (for high R).

The situation is different for the observer. In the non-discounted case (Theorem 3.4) it trades off aggressive adjustments of $\hat{\boldsymbol{x}}$ to compensate for process noise (for high V) with cautious adjustments due to the presence of measurement noise (for high W). In the discounted case, also the variance $\Sigma_0 - \boldsymbol{\mu}_{\mathbf{0}}\boldsymbol{\mu}_{\mathbf{0}}{}^T$ of the initial state matters. If the initial state is highly uncertain (large variance) and if the present matters more than the future

(highly negative α) then we get a more aggressively adjusting state estimator. Roughly put, the state estimator wants to 'fix' the uncertainty in the state while things still matter. This is contrary to the default effect of α: just like a highly negative value of α results in a lazy controller, it generally also gives a lazy observer.

3.4. EXPERIMENTAL VERIFICATION

To check the derived equations, we can set up an experiment. Ideally, this experiment is as basic as possible. Applying the derived equations to an industrial experiment is hardly different from applying the known Theorem 3.4 (regular LQG control) to the same experiment, so doing that will not provide any new insights.

We set up an experiment with the following parameters,

$$\begin{aligned} \boldsymbol{\mu_0} &= \begin{bmatrix} 10 \\ -8 \end{bmatrix}, & \Sigma_0 &= \begin{bmatrix} 1 & 0.5 \\ 0.5 & 2 \end{bmatrix} + \boldsymbol{\mu_0}\boldsymbol{\mu_0}^T, \\ A &= \begin{bmatrix} 2 & -1 \\ 0.5 & -3 \end{bmatrix}, & B &= \begin{bmatrix} 0.5 \\ -1 \end{bmatrix}, \\ C &= \begin{bmatrix} 0 & 1 \end{bmatrix}, & D &= \begin{bmatrix} 0 \end{bmatrix}. \end{aligned} \tag{3.33}$$

In addition, we use $V = 0.3I$, $W = 0.01$, $Q = I$, $R = 0.2$ and $\alpha = -0.1$. The system matrices are mostly chosen randomly, albeit taken such that the system is both stabilizable and detectable. The other parameters are set up such that all factors (the mean initial state, the initial state variance, the process noise and the measurement noise) contribute roughly equally to the final cost.

For the given set-up, we can calculate the optimal controller/observer gains. These turn out to be

$$F_\alpha = \begin{bmatrix} 6.92 & -1.41 \end{bmatrix}, \qquad K_\alpha = \begin{bmatrix} 80.2 \\ 9.54 \end{bmatrix}. \tag{3.34}$$

The resulting expected cost, according to (3.30), becomes $\mathbb{E}[J] = 1\,093$. We can verify this through numerical simulations. After running 10^5 simulations, each time initializing the system in a random initial state and applying the appropriate process/measurement noise for $T = 40$ seconds, while keeping track of the cost J, the mean cost of all simulations was 1 094. The tiny difference from the predicted mean can partly be explained by statistical deviations, and partly by minor inaccuracies in the discretization of the simulation. However, the result is still very close to the prediction, verifying (3.30).

We should also check whether the given controller settings indeed minimize the discounted cost. To do so, we can make minor adjustments to the gains F_α and K_α and investigate the resulting change in expected cost. The results of this are shown in Table 3.1. Here we see that small adjustments to either of the gains, both positive and negative, will only increase the expected cost $\mathbb{E}[J]$. This confirms that the given controller parameters are indeed optimal.

3.5. CONCLUSIONS

It is now possible to find the optimal controller and observer gains of an LQG system with discounted cost using the expressions provided in Theorem 3.5. The explicit expressions

Table 3.1: The effects of minor adjustments of elements of the optimal controller/observer gains F_α and K_α on the expected cost $\mathbb{E}[J]$. Note that the change in cost is always positive: the cost always increases. Numbers were derived using (3.30).

Adjustment of F_α	$\Delta F_{\alpha,1} = -0.1$	$\Delta F_{\alpha,1} = 0.1$	$\Delta F_{\alpha,2} = -0.1$	$\Delta F_{\alpha,2} = 0.1$
Effect on $\mathbb{E}[J]$	$\Delta\mathbb{E}[J] = 0.23$	$\Delta\mathbb{E}[J] = 0.22$	$\Delta\mathbb{E}[J] = 0.41$	$\Delta\mathbb{E}[J] = 0.41$
Adjustment of K_α	$\Delta K_{\alpha,1} = -0.5$	$\Delta K_{\alpha,1} = 0.5$	$\Delta K_{\alpha,2} = -0.1$	$\Delta K_{\alpha,2} = 0.1$
Effect on $\mathbb{E}[J]$	$\Delta\mathbb{E}[J] = 0.059$	$\Delta\mathbb{E}[J] = 0.058$	$\Delta\mathbb{E}[J] = 0.087$	$\Delta\mathbb{E}[J] = 0.088$

for the expected cost are also provided. As a result, this chapter also serves as an overview when it comes to this part of control engineering.

Future work on this subject can look into replacing the discount exponent α by a discount matrix. This seems to be a mostly straightforward problem with fascinating additional tuning possibilities. A more complicated matter would be to figure out how a finite time window T affects the optimal controller/observer parameters for various α. Yet another avenue for further research would be to look into time-varying systems, similarly to Rusnak (2016), and verify whether the same results are still applicable then.

4

Online System Identification through Gaussian Process Regression

Abstract — There has been a growing interest in using non-parametric regression methods like Gaussian Process (GP) regression for system identification. GP regression does traditionally have three important downsides: (1) it is computationally intensive, (2) it cannot efficiently implement newly obtained measurements online, and (3) it cannot deal with stochastic (noisy) input points. In this chapter we present an algorithm tackling all these three issues simultaneously. The resulting Sparse Online Noisy Input GP (SONIG) regression algorithm can incorporate new noisy measurements in constant runtime. A comparison has shown that it is more accurate than similar existing regression algorithms. When applied to non-linear black-box system modeling, its performance is competitive with existing non-linear ARX models.

The Gaussian Process (GP) (Rasmussen and Williams, 2006) has established itself as a standard model for nonlinear functions. It offers a representation that is non-parametric and probabilistic. The *non-parametric* nature of the GP means that it does not rely on any particular parametric functional form to be postulated. The fact that the GP is a probabilistic model means that it is able to take uncertainty into account in every aspect of the model. This makes it a promising method to use for the identification of nonlinear systems with strong uncertainties like wind turbines. However, this is not as straightforward as it may seem. In this chapter we look at why that is the case and how these problems can be overcome.

We start this chapter by describing the system identification set-up we aim to tackle and why GP regression can be used for this in Section 4.1. Section 4.2 then examines three important existing problems within GP regression, giving a quick summary of the solutions discussed in literature. In Section 4.3 we expand on these methods, enabling

GP regression to be applied in an online manner using noisy input points. This results in the basic version of the new algorithm. Section 4.4 subsequently outlines various ways of extending this algorithm, allowing it to be applied to system identification problems. Experimental results, first for the basic algorithm (Algorithm 1) and then for its system identification set-up (Algorithm 2) are shown in Section 4.5. Section 4.6 finally gives conclusions and recommendations for future work along this direction.

4.1. SYSTEM IDENTIFICATION PROBLEM SET-UP

The nonlinear system identification problem amounts to learning a mathematical model based on data that is observed from a dynamical phenomenon under study. Recently there has been a growing interest of using the GP to this end and it has in fact allowed researchers to successfully revisit the linear system identification problem and establish new and significantly improved results on the impulse estimation problem (Pillonetto and De Nicolao, 2010, Pillonetto et al., 2011, Chen et al., 2012). There are also older results on nonlinear ARX type models (Kocijan et al., 2005) and new results on the nonlinear state space models based on the GP (Svensson and Schön, 2017, Frigola et al., 2013, 2014). We also mention the nice overview by Kocijan (2016).

When the basic GP model is used for regression, it results in a computational complexity that is too high to be practically useful, stochastic inputs cannot be used and it cannot be used in an online fashion. These three fundamental problems of basic GP regression have been addressed in many different ways, which we return to in Section 4.2. The nonlinear system identification problem typically requires us to solve these three problems simultaneously. This brings us to our two main contributions of this chapter. (1) We derive an algorithm allowing us to—in an online fashion—include stochastic training points to one of the classic sparse GP models, the so-called FITC algorithm. (2) We adapt the new algorithm to the nonlinear system identification problem, resulting in a novel online algorithm for nonlinear system identification. The experimental results show that the our new algorithm is indeed competitive compared to existing solutions.

The system identification formulation takes inspiration from the nonlinear autoregressive model with exogenous (ARX) inputs of the form

$$\boldsymbol{y_k} = \boldsymbol{\phi}(\boldsymbol{y_{k-1}}, \ldots, \boldsymbol{y_{k-n_y}}, \boldsymbol{u_{k-1}}, \ldots, \boldsymbol{u_{k-n_u}}), \tag{4.1}$$

where $\boldsymbol{\phi}(\cdot)$ denotes some nonlinear function of past inputs $\boldsymbol{u_{k-1}}, \ldots, \boldsymbol{u_{k-n_u}}$ and past outputs $\boldsymbol{y_{k-1}}, \ldots, \boldsymbol{y_{k-n_y}}$ to the system. To this end we develop a non-parametric and probabilistic GP model which takes the vector

$$\boldsymbol{x_k} = (\boldsymbol{y_{k-1}}, \ldots, \boldsymbol{y_{k-n_y}}, \boldsymbol{u_{k-1}}, \ldots, \boldsymbol{u_{k-n_u}}), \tag{4.2}$$

as its input vector. The crucial part behind our solution from a system identification point of view is that we continuously keep track of the covariances between respective output estimates $\boldsymbol{y_k}, \ldots, \boldsymbol{y_{k-n_y}}$ and inputs $\boldsymbol{u_k}, \ldots, \boldsymbol{u_{k-n_u}}$. Every time we incorporate new training data, the respective means and covariances of these parameters are further refined.

4.2. LIMITATIONS OF GAUSSIAN PROCESS REGRESSION

Gaussian process regression is a powerful regression method but, like any method, it has its limitations. In this section we look at its background, what exact limitations it has and what methods are available in literature to tackle these. Also the assumptions made and the notation used is introduced.

4.2.1. REGULAR GAUSSIAN PROCESS REGRESSION

GP regression (Rasmussen and Williams, 2006) is about probabilistically modeling a function $f(\boldsymbol{x})$ given a number of n training points (measurements) $(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \ldots, (\boldsymbol{x_n}, y_n)$. Here, $\boldsymbol{x}$ denotes the training input point and y the measured function output. (For now we assume scalar outputs. Section 4.4.1 looks at the multi-output case.) We assume that the training outputs are corrupted by noise, such that $y_i = f(\boldsymbol{x_i}) + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ being Gaussian white noise with zero mean and variance σ_n^2.

As shorthand notation, we merge all the training points $\boldsymbol{x_i}$ into a training set X and all corresponding output values y_i into an output vector $\boldsymbol{y}$. We now write the noiseless function output $f(X)$ as $\boldsymbol{f}$, such that $\boldsymbol{y} = \boldsymbol{f} + \boldsymbol{\varepsilon}$, with $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_n)$ and $\Sigma_n = \sigma_n^2 I$.

Once we have the training data, we want to predict the function value $f(\boldsymbol{x}_*)$ at a specific test point $\boldsymbol{x}_*$. Equivalently, we can also predict the function values $\boldsymbol{f}_* = f(X_*)$ at a whole set of test points X_*. Given that we model the function $f(\boldsymbol{x})$ as a Gaussian process, it follows that $\boldsymbol{f}_*$ and $\boldsymbol{f}$ have a prior joint Gaussian distribution given by

$$\begin{bmatrix} \boldsymbol{f^0} \\ \boldsymbol{f_*^0} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} k(X,X) & k(X,X_*) \\ k(X_*,X) & k(X_*,X_*) \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{m}_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right), \tag{4.3}$$

where in the second part of the equation we have introduced another shorthand notation. Note here that $m(\boldsymbol{x})$ is the prior mean function for the Gaussian process and $k(\boldsymbol{x}, \boldsymbol{x}')$ is the prior covariance function. The superscript 0 in $\boldsymbol{f^0}$ and $\boldsymbol{f_*^0}$ also denotes that we are referring to the prior distribution: no training points have been taken into account yet. In this chapter we make no assumptions on the prior mean/kernel functions, but our examples will apply a zero mean function $m(\boldsymbol{x}) = 0$ and a squared exponential covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \alpha^2 \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}')^T \Lambda^{-1} (\boldsymbol{x} - \boldsymbol{x}') \right), \tag{4.4}$$

with α a characteristic output length scale and Λ a diagonal matrix of characteristic input squared length scales. For now we assume that these hyperparameters are known, but in Section 4.4.3 we look at ways to tune them.

From (4.3) we can find the posterior distribution of both $\boldsymbol{f}$ and $\boldsymbol{f}_*$ given $\boldsymbol{y}$ as

$$\begin{aligned} \begin{bmatrix} \boldsymbol{f^n} \\ \boldsymbol{f_*^n} \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix}, \begin{bmatrix} \Sigma^n & \Sigma_*^n \\ (\Sigma_*^n)^T & \Sigma_{**}^n \end{bmatrix} \right), \\ \begin{bmatrix} \boldsymbol{\mu^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix} &= \begin{bmatrix} \left(K^{-1} + \Sigma_n^{-1}\right)^{-1} \left(K^{-1}\boldsymbol{m} + \Sigma_n^{-1}\boldsymbol{y}\right) \\ \boldsymbol{m}_* + K_*^T (K + \Sigma_n)^{-1} (\boldsymbol{y} - \boldsymbol{m}) \end{bmatrix}, \\ \begin{bmatrix} \Sigma^n & \Sigma_*^n \\ (\Sigma_*^n)^T & \Sigma_{**}^n \end{bmatrix} &= \begin{bmatrix} \left(K^{-1} + \Sigma_n^{-1}\right)^{-1} & \Sigma_n (K + \Sigma_n)^{-1} K_* \\ K_*^T (K + \Sigma_n)^{-1} \Sigma_n & K_{**} - K_*^T (K + \Sigma_n)^{-1} K_* \end{bmatrix}. \end{aligned} \tag{4.5}$$

Note here that, while we use $\boldsymbol{m}$ and K to denote properties of prior distributions, we use $\boldsymbol{\mu}$ and Σ for posterior distributions. The superscript n indicates these are posteriors taking n training points into account, and while a star $*$ subscript denotes a parameter of the test set, an omitted subscript denotes a training parameter.

4.2.2. SPARSE GAUSSIAN PROCESS REGRESSION

An important limitation of Gaussian process regression is its computational complexity of $\mathcal{O}(n^3)$, with n the number of training points. This can be tackled through parallel computing (Gal et al., 2014, Deisenroth and Ng, 2015) but a more common solution is to use the so-called sparse methods. An overview of these is given by Candela and Rasmussen (2005), summarizing various contributions (Smola and Bartlett, 2001, Csató and Opper, 2002, Seeger et al., 2003, Snelson and Ghahramani, 2006a) into a comprehensive framework. With these methods, and particularly with the FITC method that we will apply in this chapter, the runtime can be reduced to being linear with respect to n, with only a limited reduction in how well the available data is being used.

All these sparse methods make use of so-called inducing input points X_u to reduce the computational complexity. Such inducing input points are also used in the more recent work on variational inference (Titsias, 2009, Titsias and Lawrence, 2010). However, as pointed out by McHutchon (2014), these points are now not used for the sake of computational speed but merely as 'information store'. McHutchon (2014) also noted that the corresponding methods have a large number of parameters to optimize, making the computation of the derivatives rather slow. Furthermore, even though variations have been developed which do allow the application of variational inference to larger data sets (Hensman et al., 2013, Gal et al., 2014, Damianou et al., 2016), online methods generally require simpler and faster methods, like NIGP, which is what we will focus on.

To apply sparse GP regression, we first find the posterior distribution of the inducing outputs $\boldsymbol{f_u}$ at the corresponding inducing input points X_u. This can be done in $\mathcal{O}(n^3)$ time through (4.5) (replacing $\boldsymbol{f_*}$ by $\boldsymbol{f_u}$) or in $\mathcal{O}(nn_u^2)$ time through the FITC regression equation

$$
\begin{aligned}
\begin{bmatrix}\boldsymbol{f^n}\\ \boldsymbol{f_u^n}\end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu^n}\\ \boldsymbol{\mu_u^n}\end{bmatrix}, \begin{bmatrix}\Sigma^n & \Sigma_u^n\\ \left(\Sigma_u^n\right)^T & \Sigma_{uu}^n\end{bmatrix}\right),\\
\begin{bmatrix}\boldsymbol{\mu^n}\\ \boldsymbol{\mu_u^n}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{m} + \Sigma^n\Sigma_n^{-1}\left(\boldsymbol{y}-\boldsymbol{m}\right)\\ \boldsymbol{m_u} + \left(\Sigma_u^n\right)^T\left(\Lambda_n^{-1}+\Sigma_n^{-1}\right)\left(\boldsymbol{y}-\boldsymbol{m}\right)\end{bmatrix},\\
\Sigma^n &= \left(\Lambda_n^{-1}+\Sigma_n^{-1}\right)^{-1} + \Sigma_n\left(\Lambda_n+\Sigma_n\right)^{-1}K_u\Delta^{-1}K_u^T\left(\Lambda_n+\Sigma_n\right)^{-1}\Sigma_n,\\
\Sigma_u^n &= \Sigma_n\left(\Lambda_n+\Sigma_n\right)^{-1}K_u\Delta^{-1}K_{uu},\\
\Sigma_{uu}^n &= K_{uu}\Delta^{-1}K_{uu}.
\end{aligned}
\tag{4.6}
$$

Here we have used the shorthand notation $K_u = k(X, X_u)$, $K_{uu} = k(X_u, X_u)$ and $\Delta = K_{uu} + K_u^T\left(\Lambda_n+\Sigma_n\right)^{-1}K_u$, in which also $\Lambda_n = \mathrm{diag}\left(K - K_u^T K_{uu}^{-1} K_u\right)$, with diag being the function that sets all non-diagonal elements of the given matrix to zero. The FITC regression equation is an approximation, based on the assumption that all function values $f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_n})$ are (a priori) independent given $\boldsymbol{f_u}$.

Once we know the distribution of $\boldsymbol{f_u}$, we calculate the posterior distribution of $\boldsymbol{f_*}$.

Mathematically, this method is equivalent to assuming that $\boldsymbol{f}$ and $\boldsymbol{f}_*$ are conditionally independent, given $\boldsymbol{f_u}$. It follows that

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{f_u^n} \\ \boldsymbol{f_*^n} \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix}, \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u*}^n \\ \Sigma_{*u}^n & \Sigma_{**}^n \end{bmatrix} \right), \\
\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu_u^n} \\ \boldsymbol{m_*} + K_{*u} K_{uu}^{-1} \left(\boldsymbol{\mu_u^n} - \boldsymbol{m_u} \right) \end{bmatrix}, \\
\begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u*}^n \\ \Sigma_{*u}^n & \Sigma_{**}^n \end{bmatrix} &= \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu}^n K_{uu}^{-1} K_{u*} \\ K_{*u} K_{uu}^{-1} \Sigma_{uu}^n & K_{**} - K_{*u} K_{uu}^{-1} \left(K_{uu} - \Sigma_{uu}^n \right) K_{uu}^{-1} K_{u*} \end{bmatrix}.
\end{aligned} \tag{4.7}$$

4.2.3. ONLINE GAUSSIAN PROCESS REGRESSION

A second limitation of GP regression is the difficulty with which it can incorporate new training points. For regular GP regression, a new measurement $n+1$ can be added to the existing set of n training points through a matrix update, resulting in an $\mathcal{O}(n^2)$ runtime. For sparse methods using inducing input (basis) points this can generally be done more efficiently (Csató and Opper, 2002, Candela and Rasmussen, 2005, Ranganathan et al., 2011, Kou et al., 2013, Hensman et al., 2013). The main downside is that most methods set requirements on these inducing input points. However, the FITC and the PITC methods can be set up in an online way without such constraints (Huber, 2013, 2014, Bijl et al., 2015). We briefly summarize the resulting algorithm.

Suppose that we know the distribution of $\boldsymbol{f_u}$, given the first n training points. This is written as $\boldsymbol{f_u^n}$. Next, consider a new measurement $(\boldsymbol{x_{n+1}}, y_{n+1})$, whose notation we will shorten to $(\boldsymbol{x}_+, y_+)$. To incorporate it, we first predict the posterior distribution of $f_+ = f(\boldsymbol{x}_+)$ based on only the first n training points. Identically to (4.7), this results in

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{f_u^n} \\ f_+^n \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \mu_+^n \end{bmatrix}, \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u+}^n \\ \Sigma_{+u}^n & \Sigma_{++}^n \end{bmatrix} \right), \\
\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \mu_+^n \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu_u^n} \\ m_+ + K_{+u} K_{uu}^{-1} \left(\boldsymbol{\mu_u^n} - \boldsymbol{m_u} \right) \end{bmatrix}, \\
\begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u+}^n \\ \Sigma_{+u}^n & \Sigma_{++}^n \end{bmatrix} &= \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu}^n K_{uu}^{-1} K_{u+} \\ K_{+u} K_{uu}^{-1} \Sigma_{uu}^n & K_{++} - K_{+u} K_{uu}^{-1} \left(K_{uu} - \Sigma_{uu}^n \right) K_{uu}^{-1} K_{u+} \end{bmatrix}.
\end{aligned} \tag{4.8}$$

If we subsequently incorporate the new measurement, identically to (4.5), then we get

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{f_u^{n+1}} \\ f_+^{n+1} \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u^{n+1}} \\ \mu_+^{n+1} \end{bmatrix}, \begin{bmatrix} \Sigma_{uu}^{n+1} & \Sigma_{u+}^{n+1} \\ \Sigma_{+u}^{n+1} & \Sigma_{++}^{n+1} \end{bmatrix} \right), \\
\begin{bmatrix} \boldsymbol{\mu_u^{n+1}} \\ \mu_+^{n+1} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu_u^n} + \Sigma_{u+}^n \left(\Sigma_{++}^n + \sigma_n^2 \right)^{-1} \left(y_+ - \mu_+^n \right) \\ \sigma_n^2 \left(\Sigma_{++}^n + \sigma_n^2 \right)^{-1} \mu_+^n + \Sigma_{++}^n \left(\Sigma_{++}^n + \sigma_n^2 \right)^{-1} y_+ \end{bmatrix}, \\
\begin{bmatrix} \Sigma_{uu}^{n+1} & \Sigma_{u+}^{n+1} \\ \Sigma_{+u}^{n+1} & \Sigma_{++}^{n+1} \end{bmatrix} &= \begin{bmatrix} \Sigma_{uu}^n - \Sigma_{u+}^n (\Sigma_{++}^n + \sigma_n^2)^{-1} \Sigma_{+u}^n & \Sigma_{u+}^n (\Sigma_{++}^n + \sigma_n^2)^{-1} \sigma_n^2 \\ \sigma_n^2 (\Sigma_{++}^n + \sigma_n^2)^{-1} \Sigma_{+u}^n & \sigma_n^2 (\Sigma_{++}^n + \sigma_n^2)^{-1} \Sigma_{++}^n \end{bmatrix}.
\end{aligned} \tag{4.9}$$

This expression (or at least the part relating to $\boldsymbol{f_u}$) is the update law for the FITC algorithm. It tells us exactly how the distribution of $\boldsymbol{f_u^{n+1}}$ (both $\boldsymbol{\mu_u^{n+1}}$ and Σ_{uu}^{n+1}) depends on $\boldsymbol{x}_+$ and y_+. With this distribution, we can subsequently always find the distribution of new test point outputs $\boldsymbol{f}_*$ in an efficient way through (4.7).

4.2.4. USING STOCHASTIC INPUT POINTS

The third limitation is that the GP regression algorithm assumes that the input points are deterministic. This assumption concerns both the training (measurement) points $\boldsymbol{x}$ and the test points $\boldsymbol{x}_*$. For noisy (stochastic) test points $\boldsymbol{x}_*$, we can work around the problem by applying moment matching (Deisenroth, 2010). This technique can subsequently also be expanded for noisy training points (Dallaire et al., 2009), but the effectiveness is limited because the method integrates over all possible *a priori* functions, and not over all possible *a posteriori* functions. There are methods that include posterior distributions (Girard and Murray-Smith, 2003) but these often only work for noisy test points and not for noisy training points. Only the NIGP algorithm (McHutchon and Rasmussen, 2011) both includes posterior distributions and works with noisy training points. This is the method we will be expanding upon in this chapter.

4

It should be noted that the variational methods mentioned earlier can also deal with stochastic input points, up to a certain degree. However, as also mentioned before, they cannot do so as computationally efficient as the NIGP algorithm or the algorithm that we will develop, so their applicability to online system identification remains limited. Additionally, it is also possible to take into account the effects of noisy training points by assuming that the noise variance varies over the input space; a feature called heteroscedasticity. This has been investigated quite in-depth (Goldberg et al., 1997, Le et al., 2005, Wang and Neal, 2012, Snelson and Ghahramani, 2006b) but it would give more degrees of freedom to the learning algorithm than would be required, and as a result these methods have a reduced performance for the problems we consider. We will not consider these methods further in this thesis.

4.3. EXPANDING THE ALGORITHM FOR STOCHASTIC TRAINING POINTS

This section contains our main contribution: enabling the FITC algorithm to handle stochastic training points in an online way. From a computational point of view, the novel update laws given here are very simple and efficient, relative to other methods.

4.3.1. THE ONLINE STOCHASTIC MEASUREMENT PROBLEM

Consider the case where we know the distribution $\boldsymbol{f_u^n} \sim \mathcal{N}\left(\boldsymbol{\mu_u^n}, \boldsymbol{\Sigma_{uu}^n}\right)$ (initially we have $\boldsymbol{f_u^0} \sim \mathcal{N}\left(\boldsymbol{\mu_u^0}, \Sigma_{uu}^0\right) = \mathcal{N}\left(\boldsymbol{m_u}, K_{uu}\right)$) and we obtain a new measurement at some unknown input point $\boldsymbol{x}_+$. However, we do have a measurement $\hat{\boldsymbol{x}}_+$ of this point, and we assume that the discrepancy $\boldsymbol{\varepsilon}$ has a normal distribution. That is,

$$\hat{\boldsymbol{x}}_+ = \boldsymbol{x}_+ + \boldsymbol{\varepsilon}, \tag{4.10}$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_{+_x})$. As before, the true function output $f_+ = f(\boldsymbol{x}_+)$ is also unknown. Our measurement only gives us values $\hat{\boldsymbol{x}}_+$ and $\hat{f}_+$ approximating these and hence tells us that $\boldsymbol{x}_+ \sim \mathcal{N}(\hat{\boldsymbol{x}}_+, \Sigma_{+_x})$ and[1] $f_+ \sim \mathcal{N}(\hat{f}_+, \Sigma_{+_f})$. We assume that the noise on the input and the output is independent, and hence that $\boldsymbol{x}_+$ and f_+ are a priori not correlated.

[1] Note that $\hat{f}_+$ and y_+ are identical, and so are Σ_{+_f} and σ_n^2. For the sake of uniform notation, we have renamed them.

Our main goal is to find the posterior distribution $\boldsymbol{f_u^{n+1}}$, given this stochastic training point. This can be done through

$$p(\boldsymbol{f_u^{n+1}}|\hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n}) = \int_X p(\boldsymbol{f_u^{n+1}}|\boldsymbol{x}_+, \hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n}) p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n})\, d\boldsymbol{x}_+. \tag{4.11}$$

In the integral, the first probability $p(\boldsymbol{f_u^{n+1}}|\boldsymbol{x}_+, \hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n})$ is the update law for $\boldsymbol{f_u^{n+1}}$ if $\boldsymbol{x}_+$ was known exactly. It directly follows from (4.9). The second probability $p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n})$ is the posterior distribution of $\boldsymbol{x}_+$, given both $\boldsymbol{f_u^n}$ and the new measurement. Since this latter term is more difficult to deal with, we examine it first.

4.3.2. THE POSTERIOR DISTRIBUTION OF THE TRAINING POINT

The posterior distribution of $\boldsymbol{x}_+$ can be found through Bayes' theorem,

$$p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) = \frac{p(\hat{f}_+|\boldsymbol{x}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})}{p(\hat{f}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})}. \tag{4.12}$$

Here $p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) = p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+) = \mathcal{N}(\hat{\boldsymbol{x}}_+, \Sigma_{+_x})$ and $p(\hat{f}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})$ equals an unknown constant (i.e., not depending on $\boldsymbol{x}_+$). Additionally,

$$p(\hat{f}_+|\boldsymbol{x}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) = p(\hat{f}_+|\boldsymbol{x}_+, \boldsymbol{f_u^n}) = \mathcal{N}\left(\mu_+^n, \Sigma_{++}^n + \Sigma_{+_f}\right), \tag{4.13}$$

where μ_+^n and Σ_{++}^n follow from (4.8). Note that both these quantities depend on $\boldsymbol{x}_+$ in a non-linear way. Because of this, the resulting probability $p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})$ will be non-Gaussian. To work around this problem, we have to make some simplifying assumptions. Similarly to Girard and Murray-Smith (2003), we linearize the Gaussian process $\hat{f}_+$ (which depends on $\boldsymbol{x}_+$) around a point $\bar{\boldsymbol{x}}_+$. That is, we assume that

$$p(\hat{f}_+|\boldsymbol{x}_+, \boldsymbol{f_u^n}) = \mathcal{N}\left(\mu_+^n(\bar{\boldsymbol{x}}_+) + \frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}(\boldsymbol{x}_+ - \bar{\boldsymbol{x}}_+), \Sigma_{++}^n(\bar{\boldsymbol{x}}_+) + \Sigma_{+_f}\right). \tag{4.14}$$

In other words, we assume that the mean varies linearly with $\boldsymbol{x}_+$, while the covariance is constant everywhere. The solution for $p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})$ is now Gaussian and follows as

$$\begin{aligned}
p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) &= \mathcal{N}\left(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}, \Sigma_{+_x}^{n+1}\right), \\
\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}} &= \hat{\boldsymbol{x}}_+ + \Sigma_{+_x}^{n+1}\left(\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}\right)^T\left(\Sigma_{++}^n(\bar{\boldsymbol{x}}_+) + \Sigma_{+_f}\right)^{-1}\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}(\bar{\boldsymbol{x}}_+ - \hat{\boldsymbol{x}}_+) + \left(\hat{f}_+ - \mu_+^n(\bar{\boldsymbol{x}}_+)\right)\right)\right), \\
\Sigma_+^{n+1} &= \left(\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}\right)^T\left(\Sigma_{++}^n(\bar{\boldsymbol{x}}_+) + \Sigma_{+_f}\right)^{-1}\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}\right) + \Sigma_{+_x}^{-1}\right)^{-1}.
\end{aligned} \tag{4.15}$$

We are left to choose a linearization point $\bar{\boldsymbol{x}}_+$. The above equation is easiest to apply when we choose $\bar{\boldsymbol{x}}_+ = \hat{\boldsymbol{x}}_+$, but when $\left(\hat{f}_+ - \mu_+^n(\bar{\boldsymbol{x}}_+)\right)$ is large, this may result in inaccuracies due to the linearization. It is generally more accurate to choose $\bar{\boldsymbol{x}}_+$ as $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$. However, $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ is initially unknown, so it may be necessary to apply the above equation multiple times, each time resetting $\bar{\boldsymbol{x}}_+$ to the latest value of $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ that was found, to find the most accurate posterior distribution of $\boldsymbol{x}_+$.

4.3.3. UPDATING THE INDUCING INPUT POINT FUNCTION VALUES

Using the above, we can solve (4.11). This is done by approximating the GP $\boldsymbol{f_u}(\boldsymbol{x}_+)$ by its Taylor expansion around $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$. Element-wise we write this as

$$f_{u_i}^{n+1}(\boldsymbol{x}_+) = f_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \frac{df_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\left(\boldsymbol{x}_+ - \hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}\right) + \frac{1}{2}\left(\boldsymbol{x}_+ - \hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}\right)^T \frac{d^2 f_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\left(\boldsymbol{x}_+ - \hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}\right) + \ldots. \tag{4.16}$$

Within this Taylor expansion, Girard and Murray-Smith (2003) made the assumption that higher order derivatives like $\frac{d^2 \boldsymbol{f_u^n}}{d\boldsymbol{x}_+^2}$ are negligible, remaining with just a linearization of the GP. We do not make this assumption, but instead assume that $\Sigma_{+_x}^2$ and higher powers of Σ_{+_x} are negligible. (If the uncertainties in $\boldsymbol{x}_+$ are so large that this assumption does not hold, then any form of Gaussian process regression is likely to fail.) This assumption is not only more loose—resulting in an extra term in (4.16)—but also easier to verify.

An additional assumption we need to make is that $\boldsymbol{x}_+$ is independent of $\boldsymbol{f_u}$. This is reasonable, as $\boldsymbol{x}_+$ is only contaminated by Gaussian white noise. Applying this, we can solve (4.11) through both (4.15) and (4.16). The result equals

$$\begin{aligned}
\boldsymbol{f_u^{n+1}} &\sim \mathcal{N}\left(\boldsymbol{\mu_u^{n+1}}, \Sigma_{uu}^{n+1}\right), \\
\mu_{u_i}^{n+1} &= \mu_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \frac{1}{2}\mathrm{tr}\left(\frac{d^2\mu_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\Sigma_{+_x}^{n+1}\right), \\
\Sigma_{u_i u_j}^{n+1} &= \Sigma_{u_i u_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \left(\frac{d\mu_{u_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)\Sigma_{+_x}^{n+1}\left(\frac{d\mu_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)^T + \frac{1}{2}\mathrm{tr}\left(\left(\frac{d^2\Sigma_{u_i u_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\right)\Sigma_{+_x}^{n+1}\right).
\end{aligned} \tag{4.17}$$

Here, the functions $\mu_{u_i}^{n+1}(\boldsymbol{x}_+)$ and $\Sigma_{u_i u_j}^{n+1}(\boldsymbol{x}_+)$ (for a given point $\boldsymbol{x}_+$) are given by (4.9), combined with (4.8). Finding all the derivatives of these parameters can be a daunting task, especially for non-scalar inputs $\boldsymbol{x}_+$, but the mathematics are relatively straightforward, so for this we refer to the Appendix.

It is interesting to compare expression (4.17) with what was used by McHutchon and Rasmussen (2011) in their NIGP algorithm. They did not include the term involving $d^2\mu_{u_i}^{n+1}/d\boldsymbol{x}_+^2$. Later on, in Section 4.5.1, we will find that exactly this term causes the new algorithm to perform better than the NIGP algorithm. As such, the above update law (4.17) also serves as an improvement with respect to the NIGP algorithm.

4.3.4. THE SONIG ALGORITHM

Applying the equations developed so far is done through the Sparse Online Noisy Input GP (SONIG) algorithm, outlined in Algorithm 1. This algorithm is computationally efficient, in the sense that a single updating step (incorporating one training point) can be done in constant runtime with respect to the number of training points already processed. The runtime does depend on the number of inducing input points through $\mathcal{O}(n_u^3)$, just like it does for all sparse GP regression algorithms.

Input:

A possibly expanding set of training points $(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)$ in which both $\boldsymbol{x}$ and y are distorted by Gaussian white noise.

Preparation:

Either choose the hyperparameters based on expert knowledge, or apply the NIGP hypertuning methods of McHutchon and Rasmussen (2011) on a subset of the data (a few hundred points) to find the hyperparameters.
Optionally, apply the NIGP regression methods on this subset of data to obtain an initial distribution of $\boldsymbol{f_u}$. Otherwise, initialize $\boldsymbol{f_u}$ as $\mathcal{N}(\boldsymbol{m_u}, K_{uu})$.

Updating:

while *there are unprocessed training points* $(\boldsymbol{x_{n+1}}, y_{n+1})$ **do**

1. Apply (4.15) to find the posterior distribution of the training point $\boldsymbol{x_{n+1}}$ (written as $\boldsymbol{x}_+$).
2. Use (4.17) to update the distribution of $\boldsymbol{f_u}$.
3. Optionally, use (4.18) and (4.19) to calculate the posterior distribution of the function value $\boldsymbol{f}(\boldsymbol{x}_+)$ (written as $\boldsymbol{f}_+$).

end

Prediction:

Apply (4.7) to find the distribution $\boldsymbol{f}_*$ for any set of deterministic test points. For stochastic test points, use the expansion from Section 4.4.5.

Algorithm 1: The Sparse Online Noisy Input GP (SONIG) algorithm: an online version of the FITC algorithm capable of dealing with stochastic (noisy) training points.

4.4. EXTENSIONS OF THE SONIG ALGORITHM

In the previous section we have presented the basic idea behind the SONIG algorithm. There are various further extensions that can be derived and implemented in the algorithm. For instance, the algorithm can deal with multi-output functions $\boldsymbol{f}(\boldsymbol{x})$ (Section 4.4.1), it can give us the posterior distribution of the output $\boldsymbol{f}_+$ as well as its correlation with the input $\boldsymbol{x}_+$ (Section 4.4.2), we can implement hyperparameter tuning (Section 4.4.3), we can add inducing input points online (Section 4.4.4) and we can make predictions $\boldsymbol{f}_*$ using stochastic test points $\boldsymbol{x}_*$ (Section 4.4.5). Many of these extensions are necessary to apply the SONIG algorithm for system identification. The exact SI algorithm is finally summarized in Algorithm 2.

4.4.1. MULTIPLE OUTPUTS

So far we have approximated functions $f(\boldsymbol{x})$ with only one output. It is also possible to approximate functions $\boldsymbol{f}(\boldsymbol{x})$ with $d_y > 1$ outputs. A common way in which this is done in GP regression algorithms (Deisenroth and Rasmussen, 2011, Álvarez et al., 2012) is by assuming that, given a deterministic input $\boldsymbol{x}$, all outputs $f_1(\boldsymbol{x}), \ldots, f_{d_y}(\boldsymbol{x})$ are independent. With this assumption, it is possible to keep a separate inducing input point distribution $\boldsymbol{f_u^i} \sim \mathcal{N}\left(\boldsymbol{\mu_u^i}, \Sigma_u^i\right)$ for each output $f_i(\boldsymbol{x})$. Hence, each output is basically treated separately.

When using stochastic input points (again, see Deisenroth and Rasmussen (2011)) the outputs do become correlated. We now have two options. If we take this correlation into account, we have to keep track of the joint distribution of the vectors $\boldsymbol{f_u^1}, \ldots, \boldsymbol{f_u^{d_y}}$, effectively merging them into one big vector. This results in a vector of size $n_u d_y$, giving our algorithm a computational complexity of $\mathcal{O}(n_u^3 d_y^3)$. Alternatively, we could also neglect the correlation between the inducing input point distributions $\boldsymbol{f_u^i}$ caused by stochastic training points $\boldsymbol{x}_+ \sim \mathcal{N}(\hat{\boldsymbol{x}}_+, \Sigma_{+_x})$. If we do, we can continue to treat each function output separately, giving our algorithm a runtime of $\mathcal{O}(n_u^3 d_y)$. Because one of the focuses of this chapter is to reduce the runtime of GP regression algorithms, we will apply the second option.

When each output is treated separately, each output also has its own hyperparameters. Naturally, the prior output covariance α_i^2 and the output noise $\sigma_{n_i}^2$ can differ per output f_i, but also the input length scales Λ_i may be chosen differently for each output. In fact, it is even possible to specify a fully separate covariance function $k_i(\boldsymbol{x}, \boldsymbol{x}')$ per output f_i, though in this thesis we stick with the squared exponential covariance function.

Naturally, there are a few equations which we should adjust slightly in the case of multivariate outputs. In particular, in equation (4.15), the parameter $\Sigma_{++}^n(\bar{\boldsymbol{x}}_+)$ would not be a scalar anymore, but become a matrix. And because of our assumption that the outputs are independent, it would be a diagonal matrix. Similarly, the derivative $d\mu_+^n / d\boldsymbol{x}_+$ would not be a row vector anymore. Instead, it would turn into the matrix $d\boldsymbol{\mu}_+^{\boldsymbol{n}} / d\boldsymbol{x}_+$. With these adjustments, (4.15) still holds and all other equations can be applied as usual.

4.4.2. THE POSTERIOR DISTRIBUTION OF THE MEASURED OUTPUT

In Section 4.3.3 we found the posterior distribution for $\boldsymbol{f_u}$. For some applications (like the system identification set-up presented in Section 4.5.2) we also need the posterior distribution of the measured function value $\boldsymbol{f}_+$, even though we do not exactly know to

which input it corresponds. We can find this element-wise, using the same methods, through

$$\mathbb{E}[\boldsymbol{f}_+]_i = \mu_{+_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \frac{1}{2}\mathrm{tr}\left(\frac{d^2\mu_{+_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\Sigma_{+_x}^{n+1}\right), \tag{4.18}$$

$$\mathbb{V}[\boldsymbol{f}_+, \boldsymbol{f}_+]_{i,j} = \Sigma_{+_i+_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \left(\frac{d\mu_{+_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)\Sigma_{+_x}^{n+1}\left(\frac{d\mu_{+_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)^T + \frac{1}{2}\mathrm{tr}\left(\left(\frac{d^2\Sigma_{+_i+_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\right)\Sigma_{+_x}^{n+1}\right).$$

Note here that $\Sigma_{++}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})$ is (by assumption) a diagonal matrix, simplifying the above equation for non-diagonal terms. As such, the covariance between two different function outputs f_{+_i} and f_{+_j} only depends on the second term in the above expression.

It may occur that we also need to know the posterior covariance between the function value $\boldsymbol{f}_+$ and the function input $\boldsymbol{x}_+$. Using the same method, we can find that

$$\mathbb{V}[\boldsymbol{f}_+, \boldsymbol{x}_+] = \left(\frac{d\boldsymbol{\mu}_+^{\boldsymbol{n+1}}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)\Sigma_{+_x}^{n+1}. \tag{4.19}$$

This allows us to find the joint posterior distribution of $\boldsymbol{x}_+$ and $\boldsymbol{f}_+$.

4.4.3. APPLYING HYPERPARAMETER TUNING TO THE ALGORITHM

So far we have assumed that the hyperparameters of the Gaussian process were known a priori. When this is not the case, they need to be tuned first. Naturally, this could be done using expert knowledge of the system, it can also be done automatically.

McHutchon and Rasmussen (2011), with their NIGP method, offer an effective method of tuning the hyperparameters, which also tells us the input noise variance Σ_{+_x}. However, this method has a computational complexity of $\mathcal{O}(n^3)$, with n still the number of training points. As such, it can only be used for a small number of measurements and it cannot be used online. NIGP seems not to be applicable to our problem.

However, Chalupka et al. (2013) compare various GP regression algorithms, including methods to tune hyperparameters. One of their main conclusions is that the subset-of-data (SoD) method—using a limited number of training points to tune hyperparameters—provides a good trade-off between computational complexity and prediction accuracy. As such, applying NIGP for hyperparameter tuning on a subset of data—say, the first few hundred data points—seems to be a decent choice.

Chalupka et al. (2013) also conclude that, for the regression problem with known hyperparameters, the FITC algorithm provides a very good trade-off between complexity and accuracy. So after the SoD method has given us an estimate of the hyperparameters, it will be an effective choice to use the online FITC algorithm with stochastic training points (that is, the SONIG algorithm) as our regression method.

4.4.4. ADJUSTING THE SET OF INDUCING INPUT POINTS ONLINE

When using an online GP regression algorithm, it is often not known in advance what kind of measured input points $\boldsymbol{x}$ the system will get. As such, choosing the inducing input points X_u in advance is not always possible, nor wise. Instead, we can adjust the inducing input points while the algorithm is running. There are ways to fully tune the set

of inducing input points, like using the latent variable methods by Titsias (2009), Titsias and Lawrence (2010), but those methods require the optimization of many parameters, resulting in a computationally complex procedure. To keep the required computations limited, we have to opt for a simpler method and add/remove inducing input points based on areas of the input space we are interested in or have data at.

Suppose that we denote the current set of inducing input points by X_u, and that we want to add an extra set of inducing input points X_{u_+}. In this case, given all the data that we have, the distributions of $\boldsymbol{f_u}$ and $\boldsymbol{f_{u_+}}$ satisfy, identically to (4.7),

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{f_u} \\ \boldsymbol{f_{u_+}} \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u} \\ \boldsymbol{\mu_{u_+}} \end{bmatrix}, \begin{bmatrix} \Sigma_{uu} & \Sigma_{uu_+} \\ \Sigma_{u_+u} & \Sigma_{u_+u_+} \end{bmatrix} \right), \\
\begin{bmatrix} \boldsymbol{\mu_u} \\ \boldsymbol{\mu_{u_+}} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu_u} \\ \boldsymbol{m_{u_+}} + K_{u_+u} K_{uu}^{-1} \left(\boldsymbol{\mu_u} - \boldsymbol{m_u} \right) \end{bmatrix}, \\
\begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu_+}^n \\ \Sigma_{u_+u}^n & \Sigma_{u_+u_+}^n \end{bmatrix} &= \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu}^n K_{uu}^{-1} K_{uu_+} \\ K_{u_+u} K_{uu}^{-1} \Sigma_{uu}^n & K_{u_+u_+} - K_{u_+u} K_{uu}^{-1} \left(K_{uu} - \Sigma_{uu}^n \right) K_{uu}^{-1} K_{uu_+} \end{bmatrix}.
\end{aligned} \tag{4.20}$$

With this combined set of old and new inducing input points, we can then continue incorporating new training points without losing any data.

Additionally, though not always wise, it is also possible to remove unimportant inducing input points. In this case, its entry can simply be removed from $\boldsymbol{f_u}$. Since it is possible to both add and remove inducing input points, it is naturally also possible to shift them around (first add new points, then remove old points) whenever deemed necessary.

The way in which we add inducing input points in the SONIG algorithm is as follows. Whenever we incorporate a new training point with posterior input distribution $\boldsymbol{x_+} \sim \mathcal{N}\left(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}, \Sigma_{+_x}^{n+1}\right)$, we check if $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ is already close to any existing inducing input point. To be more precise, we examine the normalized squared distance

$$\left(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}} - \boldsymbol{x_{u_i}}\right)^T \Lambda^{-1} \left(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}} - \boldsymbol{x_{u_i}}\right) \tag{4.21}$$

for each inducing input point $\boldsymbol{x_{u_i}}$. If there is no inducing input point whose normalized squared distance is below a given threshold (often chosen to be roughly 1, but tuned to get a satisfactory number of points), then it means that there is no inducing input point $\boldsymbol{x_{u_i}}$ close to our new training point $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$. As a result, we add $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ to our set of inducing input points. This guarantees that each training point is close to at least one inducing input point, which always allows the data from the measurement to be taken into account.

4.4.5. PREDICTIONS FOR STOCHASTIC TEST POINTS

For deterministic test points $\boldsymbol{x_*}$ we can simply make use of (4.7) to compute predictions. However, for stochastic test point $\boldsymbol{x_*} \sim \mathcal{N}\left(\hat{\boldsymbol{x}}_*, \Sigma_{*_x}\right)$ it is more challening, since we have to calculate the distribution of $\boldsymbol{f_*} = \boldsymbol{f}(\boldsymbol{x_*})$, requiring us to solve an integration problem. This will not result in a Gaussian distribution, so once more we will apply moment matching. Previously, we had to make additional assumptions, to make sure that the mean vector and the covariance matrix could be solved for analytically. This time we do not have to. Deisenroth (2010) showed that, for the squared exponential covariance function (and also various other functions) the mean vector and the covariance matrix can be calculated analytically. We can apply the same ideas in our present setting.

For our results, we will first define some helpful quantities. When doing so, we should note that in theory every output $f_k(\boldsymbol{x})$ can have its own covariance function $k_k(\ldots)$, and as such its own hyperparameters α_k and Λ_k. (See Section 4.4.1.) Keeping this in mind, we now define the vectors $\boldsymbol{q^k}$ and matrices Q^{kl} element-wise as

$$\begin{aligned} q_i^k &= \int_X k_k(\boldsymbol{x_{u_i}}, \boldsymbol{x}_*) p(\boldsymbol{x}_*)\, d\boldsymbol{x}_* \\ &= \frac{\alpha_k^2}{\sqrt{|\Sigma_{*_x}||\Sigma_{*_x}^{-1} + \Lambda_k^{-1}|}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x_{u_i}} - \hat{\boldsymbol{x}}_*\right)^T \left(\Lambda_k + \Sigma_{*_x}\right)^{-1} \left(\boldsymbol{x_{u_i}} - \hat{\boldsymbol{x}}_*\right)\right), \end{aligned} \tag{4.22}$$

$$\begin{aligned} Q_{ij}^{kl} &= \int_X k_k(\boldsymbol{x_{u_i}}, \boldsymbol{x}_*) k_l(\boldsymbol{x}_*, \boldsymbol{x_{u_j}}) p(\boldsymbol{x}_*)\, d\boldsymbol{x}_* \\ &= \frac{\alpha_k^2 \alpha_l^2}{\sqrt{|\Sigma_{*_x}||\Sigma_{*_x}^{-1} + \Lambda_k^{-1} + \Lambda_l^{-1}|}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x_{u_i}} - \boldsymbol{x_{u_j}}\right)^T (\Lambda_k + \Lambda_l)^{-1} \left(\boldsymbol{x_{u_i}} - \boldsymbol{x_{u_j}}\right)\right) \\ &\qquad \exp\left(-\frac{1}{2}\left(\bar{\boldsymbol{x}}_{\boldsymbol{u_{ij}}}^{\boldsymbol{kl}} - \hat{\boldsymbol{x}}_*\right)^T \left(\left(\Lambda_k^{-1} + \Lambda_l^{-1}\right)^{-1} + \Sigma_{*_x}\right)^{-1} \left(\bar{\boldsymbol{x}}_{\boldsymbol{u_{ij}}}^{\boldsymbol{kl}} - \hat{\boldsymbol{x}}_*\right)\right), \end{aligned} \tag{4.23}$$

where we have defined

$$\bar{\boldsymbol{x}}_{\boldsymbol{u_{ij}}}^{\boldsymbol{kl}} = \left(\Lambda_k^{-1} + \Lambda_l^{-1}\right)^{-1} \left(\Lambda_k^{-1} \boldsymbol{x_{u_i}} + \Lambda_l^{-1} \boldsymbol{x_{u_j}}\right). \tag{4.24}$$

With these quantities, we can find that

$$\begin{aligned} \boldsymbol{f}_* &\sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*), \\ [\boldsymbol{\mu}_*]_k &= \left(\boldsymbol{q^k}\right)^T \left(K_{uu}^k\right)^{-1} \boldsymbol{\mu_u^k}, \\ [\Sigma_*]_{k,k} &= \alpha_k^2 - \operatorname{tr}\left(\left(K_{uu}^k\right)^{-1} \left(K_{uu}^k - \Sigma_u^k\right) \left(K_{uu}^k\right)^{-1} Q^{kk}\right) + \left(\boldsymbol{\mu_u^k}\right)^T \left(K_{uu}^k\right)^{-1} Q^{kk} \left(K_{uu}^k\right)^{-1} \boldsymbol{\mu_u^k} - [\boldsymbol{\mu}_*]_k^2, \\ [\Sigma_*]_{k,l} &= \left(\boldsymbol{\mu_u^k}\right)^T \left(K_{uu}^k\right)^{-1} Q^{kl} \left(K_{uu}^l\right)^{-1} \boldsymbol{\mu_u^l} - [\boldsymbol{\mu}_*]_k [\boldsymbol{\mu}_*]_l, \end{aligned} \tag{4.25}$$

where the latter expression is for the non-diagonal terms of Σ_* (with $k \neq l$). Note that the first line of the above expression is in fact an approximation. In reality the distribution $\boldsymbol{f}_*$ is not Gaussian. The other two lines, however, are the analytic mean vector and covariance matrix. With these quantities, we can accurately predict the distribution of the output $\boldsymbol{f}_*$ for stochastic test points $\boldsymbol{x}_*$.

4.5. EXPERIMENTAL RESULTS

In this section we apply the developed algorithm to test problems and compare its performance to existing state of the art solutions. First we apply the basic SONIG algorithm (Algorithm 1) to approximate a sample function, allowing us to compare its performance to other regression algorithms. The results of this are discussed in Section 4.5.1. Then we apply the SONIG algorithm with all the extensions from Section 4.4 (Algorithm 2) to identify a magneto-rheological fluid damper, the outcome of which is reported in Section 4.5.2. All code for these examples, as well as for using the SONIG algorithm in general, is available on GitHub, see Bijl (2016).

Input:

A set of inputs $\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots$ and outputs $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots$ of a system that is to be identified. Both the input and the output can be disturbed by noise.

Preparation:

Define hyperparameters, either through the NIGP algorithm or by using expert knowledge about the system. Optionally, also define an initial set of inducing input points X_u.

Updating:

while *there are unprocessed measurements* $\boldsymbol{y_{k+1}}$ **do**

1. Set up $\boldsymbol{x_{k+1}}$ (shortened to $\boldsymbol{x_+}$) using its definition in (4.1). Find its prior distribution using known covariances between system outputs $\boldsymbol{y_k}, \ldots, \boldsymbol{y_{k-(n_y-1)}}$ and (if necessary) system inputs $\boldsymbol{u_k}, \ldots, \boldsymbol{u_{k-(n_u-1)}}$. Also find the prior distribution of the function output $\boldsymbol{y_{k+1}}$ (denoted as $\boldsymbol{f_{k+1}}$ or shortened as $\boldsymbol{f_+}$).
2. Apply (4.15) to find the posterior distribution $\mathcal{N}\left(\hat{\boldsymbol{x}}_+^{\boldsymbol{k+1}}, \Sigma_{+_x}^{k+1}\right)$ of $\boldsymbol{x_+}$. Optionally, use this to update the posterior distribution of the system outputs $\boldsymbol{y_k}, \ldots, \boldsymbol{y_{k-(n_y-1)}}$ and system inputs $\boldsymbol{u_k}, \ldots, \boldsymbol{u_{k-(n_u-1)}}$.
3. Optionally, if $\hat{\boldsymbol{x}}_+^{\boldsymbol{k+1}}$ is far removed from any inducing input point, add it to the set of inducing inputs X_u using (4.20). (Or rearrange/tune the inducing input points in any desired way.)
4. Calculate the posterior distribution of the inducing input vector $\boldsymbol{f_u}$ for each of the outputs of $\boldsymbol{\phi}$ using (4.17).
5. Calculate the posterior distribution of $\boldsymbol{y_{k+1}}$ using (4.18). Additionally, calculate the covariances between $\boldsymbol{y_{k+1}}$ and each of the previous system outputs $\boldsymbol{y_k}, \ldots, \boldsymbol{y_{k-(n_y-1)}}$ and inputs $\boldsymbol{u_k}, \ldots, \boldsymbol{u_{k-(n_u-1)}}$ through (4.19).

end

Prediction:

For any deterministic set of previous outputs $\boldsymbol{y_k}, \ldots, \boldsymbol{y_{k-(n_y-1)}}$ and inputs $\boldsymbol{u_k}, \ldots, \boldsymbol{u_{k-(n_u-1)}}$, apply (4.7) to predict the next output $\boldsymbol{y_{k+1}}$. For stochastic parameters, use the expansion from Section 4.4.5.

Algorithm 2: The steps required to identify nonlinear systems with measurement noise in an online way using the SONIG method.

4.5.1. TESTING THE SONIG ALGORITHM THROUGH A SAMPLE FUNCTION

To compare the SONIG algorithm with other algorithms, we have set up a basic single-input single-output GP experiment. First, we randomly generate a sample function from a Gaussian process with a squared exponential covariance function (see (4.4)). This is done on the range $x \in [-5,5]$, subject to the hyperparameters $\alpha = 1$ and $\Lambda = 1$. Subsequently, we take n training points at random places in the input range and distort both the input x and the output y with zero-mean Gaussian white noise with standard deviation $\sigma_x = 0.4$ and $\sigma_n = 0.1$, respectively. We use $n = 200$ unless mentioned otherwise. To this data set, we then apply the following algorithms.

(1) GP regression without any input noise and with the exact hyperparameters, given above. This serves as a reference case: all other algorithms get noisy input points and tuned hyperparameters.

(2) GP regression with input noise and with hyperparameters tuned through the maximum-likelihood method.

(3) The NIGP algorithm of McHutchon and Rasmussen (2011). This algorithm has its own method of tuning hyperparameters, including σ_x.

(4) The SONIG algorithm, starting with $\boldsymbol{\mu}_{\boldsymbol{u}}^{\mathbf{0}} = \boldsymbol{m}_{\boldsymbol{u}}$ and $\Sigma_{uu}^0 = K_{uu}$, using the hyperparameters given by (3). We use $n_u = 21$ evenly distributed inducing input points.

(5) The same as (4), but now with more training points (800 instead of 200). Because the SONIG algorithm is computationally more efficient than the NIGP algorithm, the runtime of this is similar to that of (3), being roughly 2-3 seconds when using Matlab, although this of course does depend on the exact implementation of the algorithms.

(6) NIGP applied on a subset of data (100 training points) to predict the distribution of the inducing input points, followed by the SONIG algorithm applied to the remainder (700) of the training set, further updating the inducing input points. The runtime of this approach is again similar to that of (3), being 2-3 seconds.

(7) The FITC algorithm, using the hyperparameters of (2). This serves as a reference case.

For all these algorithms, we examine both the Mean Squared Error (MSE) of the resulting prediction and the mean variance given by the regression algorithm. The latter is basically the estimate by the regression algorithm of the MSE. By comparing it with the real MSE, we learn about the integrity of the algorithm. As such, the ratio between these two is an indication of the algorithm integrity. We do this whole process 400 times, each time for a different randomly generated sample function from the Gaussian process. The average of the results is subsequently shown in Table 4.1.

There are many things that can be noticed from Table 4.1. First of all, it is that for the given type of functions, and for an equal number of training points, the SONIG algorithm performs better than the NIGP algorithm. This is surprising, because the SONIG algorithm can be seen as a computationally efficient approximation of the NIGP algorithm. Further experiments have shown that this is mostly because the SONIG term takes into account the second derivative of the mean in its approximation; see $\mu_{u_i}^{n+1}$ from (4.17). The NIGP algorithm does not, and if SONIG also does not (detailed experiment results not included here for sake of brevity) the performance of the two algorithms is comparable.

Table 4.1: Comparison of various GP regression algorithms, applied to noisy measurements of 400 randomly generated sample functions. For details, see the main text.

	n	MSE	Mean variance	Ratio
(1) GPR with exact hyperparameters and no input noise	200	$0.87 \cdot 10^{-3}$	$0.85 \cdot 10^{-3}$	1.02
(2) GPR with tuned hyperparameters	200	$28.0 \cdot 10^{-3}$	$8.3 \cdot 10^{-3}$	3.4
(3) NIGP with its own hyperparameter tuning	200	$26.2 \cdot 10^{-3}$	$5.6 \cdot 10^{-3}$	4.7
(4) SONIG using the hyperparameters of (3)	200	$21.5 \cdot 10^{-3}$	$8.1 \cdot 10^{-3}$	2.7
(5) SONIG using the hyperparameters of (3)	800	$12.5 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$	5.6
(6) NIGP on a subset, followed by SONIG on the rest	100/700	$16.5 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	7.1
(7) FITC, using the hyperparameters of (2)	800	$19.5 \cdot 10^{-3}$	$2.7 \cdot 10^{-3}$	7.1

A second thing that can be noticed is that more training points provide a higher accuracy. In particular, even the FITC algorithm (which does not take input noise into account) with 800 training points performs better than the NIGP or SONIG algorithms with 200 training points. It should be noted here that part of the reason is the type of function used: for functions with a steeper slope, it is expected that the NIGP and SONIG algorithms still perform better than FITC.

Finally, it is interesting to note that all algorithms, with the exception of regular GP regression with the exact hyperparameters, are much more optimistic about their predictions than is reasonable. That is, the ratio between the MSE and the mean variance is way larger than the value of 1 which it should have. Ideally, the predicted variance of all algorithms would be significantly higher.

Next, we will look at some plots. To be precise, we will examine algorithms (3) and (4) closer, but subject to only $n = 30$ training points and $n_u = 11$ inducing input points. The predictions of the two algorithms, for a single random sample function, are shown in Figure 4.1.

The most important thing that can be noticed here is that (for both methods) the posterior standard deviation varies with the slope of the to-be-approximated function. When the input is near -2, and the function is nearly flat, the standard deviation is small (well below 0.1). However, when the input is near -3 or $-1/2$, the standard deviation is larger (near 0.2). This is what can be expected, because measurements in these steep regions are much more affected/distorted by the noise, and hence provide less information.

A second thing to be noticed is the difference between the two methods. Especially for $x > 2$, where there are relatively few training points, the SONIG algorithm gives much higher variances. There are two reasons for this. The first is inherent to sparse algorithms. (The FITC algorithm would show a similar trend.) The second reason is inherent to the SONIG algorithm. Whereas regular GP regression (and similarly the NIGP algorithm) uses all training points together, the SONIG algorithm only uses data from previous training points while incorporating a new training point. As a result, when there are relatively few measurements in a certain region, and many of these measurements appear early in the updating process, the accuracy in that region can be expected to be slightly lower. However, as more training points are incorporated, which can be done very efficiently, the problem will quickly disappear.

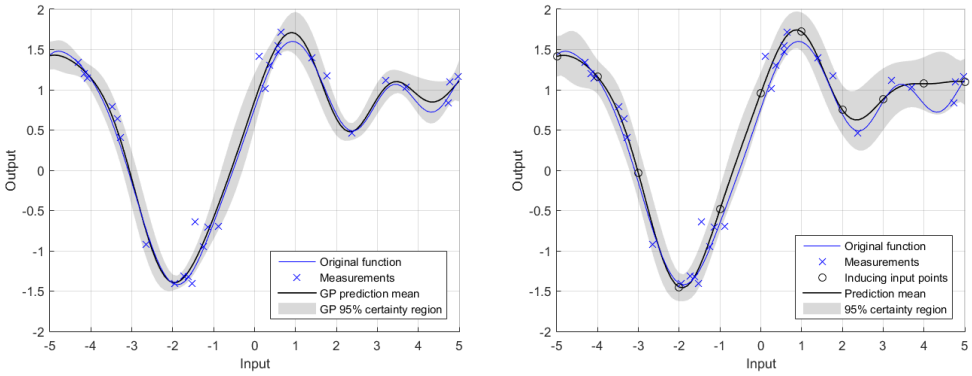

Figure 4.1: Predictions of the NIGP algorithm (left) and the SONIG algorithm (right) after $n = 30$ training points have been incorporated. Exact conditions are described in the main text.

4.5.2. IDENTIFYING THE DYNAMICS OF A FLUID DAMPER

In the next experiment we will apply the developed system identification algorithm (Algorithm 2) to a practical problem. In particular, we model the dynamical behavior of a magneto-rheological fluid damper. The measured data for this example was provided by Wang et al. (2009) and supplied through The MathWorks Inc. (2015), which also discusses various system identification examples using the techniques from Ljung (1999). This example is a common benchmark in system identification applications. It has for instance been used more recently in the context of Gaussian Process State Space Models (GP-SSM) by Svensson and Schön (2017) in their Reduced Rank GP-SSM (RR GP-SSM) algorithm.

This example has 3499 measurements provided, sampled every $\Delta t = 0.05$ seconds. We will use the first 2000 measurements (10 seconds) for training and the next 1499 measurements (7.5 seconds) for evaluation. The MathWorks Inc. (2015) recommended to use one past output and three past inputs to predict subsequent outputs. Based on this, we learn a black-box model of the following functional form

$$y_{k+1} = \phi(y_k, u_k, u_{k-1}, u_{k-2}). \tag{4.26}$$

Hyperparameters were tuned by passing a subset of the data to the NIGP algorithm. Rounded off for simplicity (which did not affect performance) they equaled

$$\begin{aligned} \Lambda &= \operatorname{diag}\left(70^2, 20^2, 10^2, 10^2\right), & \alpha^2 &= 70^2, \\ \Sigma_{+_x} &= \operatorname{diag}\left(2^2, 0.1^2, 0.1^2, 0.1^2\right), & \Sigma_{+_f} &= 2^2. \end{aligned} \tag{4.27}$$

After processing a measurement y_{k+1}, the SONIG algorithm provided us with a posterior distribution of y_{k+1}, y_k, u_k, u_{k-1} and u_{k-2}. The marginal posterior distribution of y_{k+1}, u_k and u_{k-1} was then used as prior distribution while incorporating the next measurement. Inducing input points were added online, as specified in Section 4.4.4, which eventually gave us 32 inducing input points. This is a low number, and as a result, the whole training was done in only a few (roughly 10) seconds.

After all training measurements had been used, the SONIG algorithm was given the input data for the remaining 1499 measurements, but not the output data. It had to predict this output data by itself, using each prediction y_k to predict the subsequent y_{k+1}. While doing so, the algorithm also calculated the variance of each prediction y_k, taking this into account while predicting the next output using the techniques from Section 4.4.5. The resulting predictions can be seen in Figure 4.2.

A comparison of the algorithm with various other methods is shown in Table 4.2. We also added in regular GP regression and NIGP, applied to the ARX model (4.1), as comparison. This table shows that the SONIG algorithm, when applied in its system identification set-up, can clearly outperform other black-box modeling approaches. It is better than regular GP regression at taking into account uncertainties and better than NIGP mainly due to the reasons explained before. It should be noted here, however, that this is all subject to the proper tuning of hyperparameters and the proper choice of inducing input points. With different hyperparameters or inducing input point selection strategies, the performance of the SONIG algorithm will degrade slightly, though it is still likely to outperform other identification algorithms.

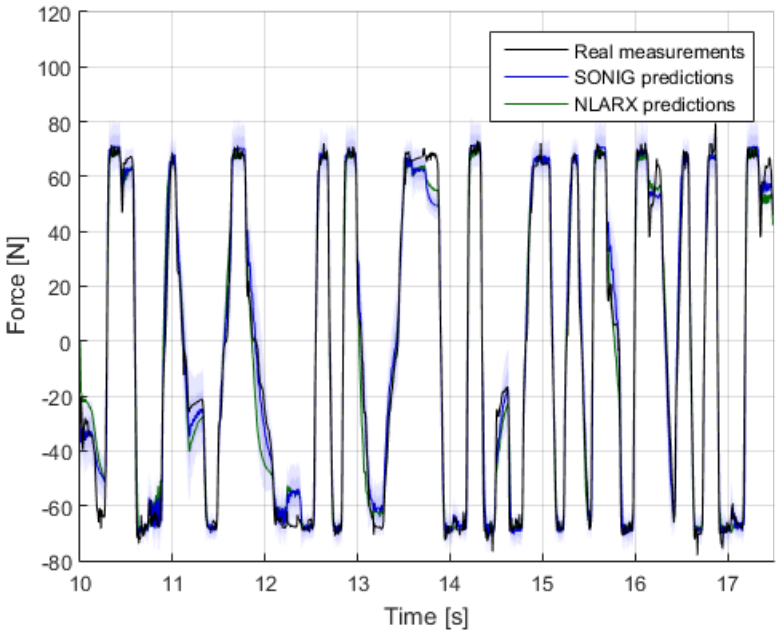

Figure 4.2: Prediction of the output of the magneto-rheological fluid damper by the SONIG algorithm, compared to the real output. The grey area represents the 95% uncertainty region as given by the algorithm. It shows that in the transition regions (like near $t = 2$ s) which the algorithm is less well trained on, the uncertainty is larger. As comparison, also the best non-linear ARX model predictions from The MathWorks Inc. (2015) are plotted. It is interesting to note that this model makes very similar errors as the SONIG algorithm, indicating the errors are mostly caused by distortions in the training/evaluation data.

Table 4.2: Comparison of various system identification models and algorithms when applied to data from the magneto-rheological fluid damper. All algorithms were given 2000 measurements for training and 1499 measurements for evaluation.

Algorithm	**RMSE**	**Source**
Linear OE model (4th order)	27.1	The MathWorks Inc. (2015)
Hammerstein-Wiener (4th order)	27.0	The MathWorks Inc. (2015)
NLARX (3rd order, wavelet network)	24.5	The MathWorks Inc. (2015)
NLARX (3rd order, tree partition)	19.3	The MathWorks Inc. (2015)
NIGP	10.2	This thesis
GP regression (GP-ARX)	9.87	This thesis
NLARX (3rd order, sigmoid network)	8.24	The MathWorks Inc. (2015)
RR GP-SSM	8.17	Svensson et al. (2016)
SONIG	**7.12**	This thesis

4.5.3. NOISY STATE MEASUREMENTS OF THE PITCH-PLUNGE SYSTEM

After this benchmark test, we can apply the SONIG algorithm to a wind turbine application. We will consider the pitch-plunge system shown in Figure 4.3. This system has been modeled by O'Neil and Strganac (1996), O'Neil et al. (1996). The specific purpose of creating this model was to research nonlinear aeroelastic behavior of wings, which is exactly what also takes place in wind turbines. Subsequently, a basic analysis of the stability characteristics and control possibilities has been performed in Ko et al. (1997), O'Neil and Strganac (1998), Ko et al. (1998). More advanced applications of this model can be found in Lind and Baldelli (2005), van Wingerden (2008).

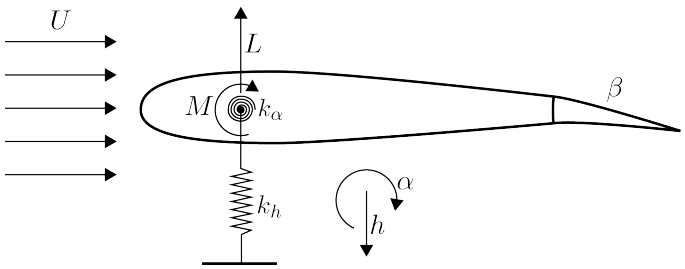

Figure 4.3: An airfoil with a trailing-edge flap. The exact system set-up is explained in the main text.

The system consists of an airfoil which is placed in an airflow with velocity U. The airfoil can plunge as well as pitch. The plunge is indicated by the height h, where downwards is positive, and the pitch is described by the angle of attack with respect to the free flow α. The blade is constrained by two springs: a vertical spring with spring constant k_h and a rotational spring with varying spring 'constant'

$$k_\alpha(\alpha) = k_\alpha\left(1 + k_{\alpha_1}\alpha^1 + k_{\alpha_2}\alpha^2 + k_{\alpha_3}\alpha^3 + k_{\alpha_4}\alpha^4\right). \tag{4.28}$$

This causes the system to be nonlinear. If we define $\boldsymbol{x} = \begin{bmatrix} h & \alpha \end{bmatrix}^T$, then the equations of

motion are given by

$$M\ddot{\boldsymbol{x}} + (C + UE)\,\dot{\boldsymbol{x}} + (K(\boldsymbol{x}) + U^2 D)\boldsymbol{x} = U^2 F\beta. \tag{4.29}$$

The matrices M, $K(\boldsymbol{x})$, C, D, E and F equal

$$M = \begin{bmatrix} m & mx_\alpha b \\ mx_\alpha b & I_\alpha \end{bmatrix}, \qquad D = \begin{bmatrix} 0 & \rho b c_{L_\alpha} \\ 0 & -\rho b^2 c_{m_\alpha} \end{bmatrix}, \tag{4.30}$$

$$K(\boldsymbol{x}) = \begin{bmatrix} k_h & 0 \\ 0 & k_\alpha(\alpha) \end{bmatrix}, \qquad E = \begin{bmatrix} \rho b c_{L_\alpha} & \rho b^2 c_{L_\alpha}\left(\frac{1}{2} - a\right) \\ -\rho b^2 c_{m_\alpha} & -\rho b^3 c_{m_\alpha}\left(\frac{1}{2} - a\right) \end{bmatrix},$$

$$C = \begin{bmatrix} c_h & 0 \\ 0 & c_\alpha \end{bmatrix}, \qquad F = \begin{bmatrix} -\rho U^2 b c_{L_\beta} \\ \rho U^2 b^2 c_{m_\beta} \end{bmatrix}.$$

The values of all parameters can be found in Table 4.3.

Table 4.3: Numerical values of the parameters of the pitch-plunge system.

Lengths	Inertia	Aerodynamics	Springs (linear)	Springs (nonlinear)
$c = 0.270\,\text{m}$	$m = 12.387\,\text{kg}$	$c_{L_\alpha} = 6.28$	$k_h = 2844.4\,\text{N/m}$	$k_{\alpha_1} = -22.1\,\text{rad}^{-1}$
$b = 0.135\,\text{m}$	$I_{CG} = 0.051\,\text{kgm}^2$	$c_{L_\beta} = 3.358$	$k_\alpha = 2.82\,\text{Nm/rad}$	$k_{\alpha_2} = 1315.5\,\text{rad}^{-2}$
$a = -0.6$	$I_\alpha = 0.065\,\text{kgm}^2$	$c_{m_\alpha} = -0.628$	$c_h = 27.43\,\text{Ns/m}$	$k_{\alpha_3} = -8580\,\text{rad}^{-3}$
$x_\alpha = 0.2466$	$\rho = 1.225\,\text{kg/m}^3$	$c_{m_\beta} = -0.635$	$c_\alpha = 0.180\,\text{Nms/rad}$	$k_{\alpha_4} = 17289.7\,\text{rad}^{-4}$

To this model, we will apply our system identification algorithm. Note that our system has four states: h, α, $\dot{h}$ and $\dot{\alpha}$. Or alternatively, the state consists of $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$. There is also one input, being β. To identify the system, we will discretize it with time step $\Delta t = 0.1\,\text{s}$. We can now approximate the system as

$$\boldsymbol{x_{k+1}} \approx f(\boldsymbol{x_k}, \boldsymbol{x_{k-1}}, \beta_k). \tag{4.31}$$

This is the function that we will strive to approximate using the SONIG algorithm.

To get any data about the system, we need to excite it. For this we use a sinusoidal input signal $\beta(t) = A\sin(2\pi f t)$, with amplitude $A = 0.5\,\text{rad}$ and frequency $f = 0.4\,\text{Hz}$. To make it a bit more challenging, we also add a disturbance to the input signal randomly taken from the uniform interval $[-0.06, 0.06]$. The resulting input signal for the first ten seconds is shown in Figure 4.4.

You may argue here that only identifying the dynamical behavior of the system at one input frequency is a bit too easy. In fact, to properly identify a system, you generally need to excite it at a sufficient number of different frequencies. Only then can you identify all the different dynamics that may be present in the system. That is also the case here. We mainly pick one frequency for reasons of simplicity: it is easier to understand what is going on during the learning process, and the reduced training time makes it easier to play around with the algorithm. Naturally it is also possible to introduce more input frequencies, but this will increase the training time required for the SONIG algorithm to figure out sufficiently well what is going on.

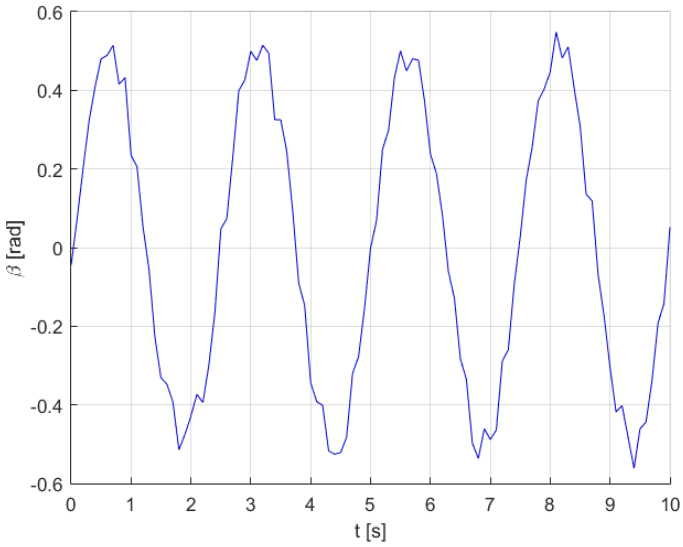

Figure 4.4: The first ten seconds of input $\beta(t)$ provided to the pitch-plunge system during its identification through the SONIG algorithm. It is a sinusoid with distortions added for extra excitations.

For the given input signal we run a simulation. This results in the state development shown in Figure 4.5. This development already looks rather shaky. To make things harder for our prediction algorithm, we will also add noise to these state values. Specifically, we use Gaussian white noise with standard deviations of $\sigma_h = 10^{-4}$m and $\sigma_\alpha = 6 \cdot 10^{-4}$rad.

After the SONIG algorithm has been trained on the first fifty seconds of measurements, it has only 27 inducing input points. With these inducing input points, we can make the predictions shown in Figure 4.6. For these predictions, only the state $\boldsymbol{x}$ at time $t = 50$s was given as well as the input β at any subsequent time.

In Figure 4.6 it is worthwhile to note that the variances of the estimates increase as time passes. This makes sense. The further we go into the future, the more uncertain our predictions become. What is more interesting is that, at some point, the uncertainties just explode. At this point the algorithm basically tells us it does not have a clue anymore what the state is likely to be.

The position of this big increase in variance depends on many factors. If the SONIG algorithm gets more training data, then it becomes more certain about its predictions, so this 'uncertainty explosion' takes place later. But if the input noise would become larger, then the uncertainty increases and the jump happens sooner. Predicting in advance when this variance increase takes place seems to be nearly impossible though. It depends on too many factors.

After the big increase in the prediction variance, also the mean of the estimates is not always sensible anymore, especially when predicting α. But this does raise the question, 'What would have happened if that jump in variance had not occurred? Would the mean of the estimates be more sensible then?' We can figure this out if we manually toggle down the variance. For example, we can manually tell the SONIG algorithm to

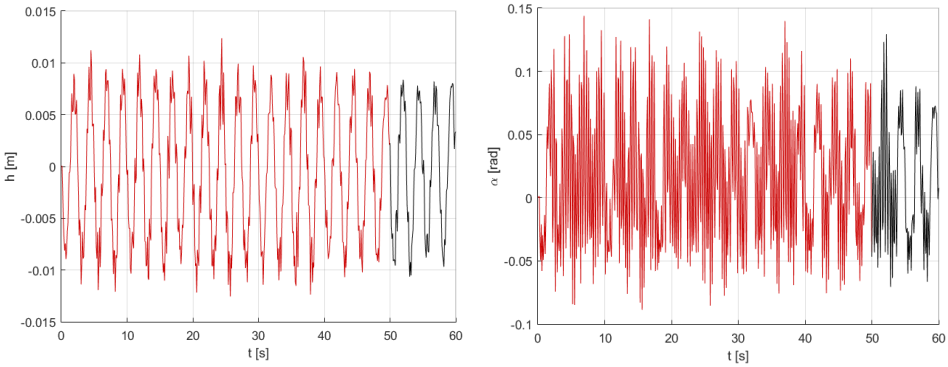

Figure 4.5: The state development of the pitch-plunge system. The first fifty seconds are used for training and the remaining ten seconds for validation. Measurements were taken at a wind speed of $U = 10\,\mathrm{m/s}$.

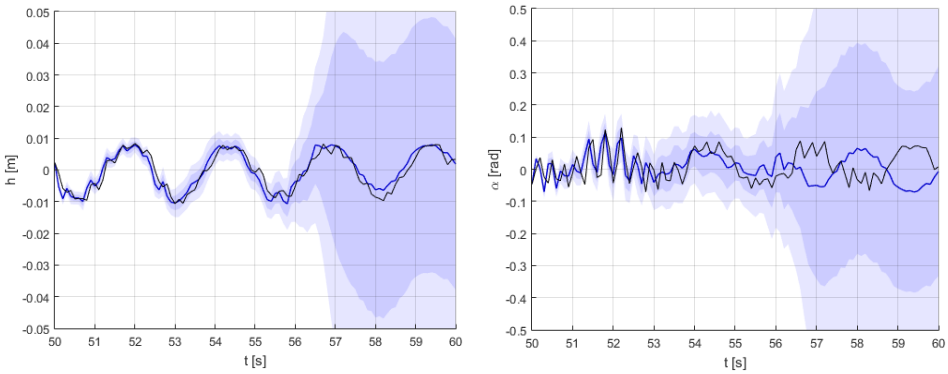

Figure 4.6: The prediction of the state development of the pitch-plunge system for the final ten seconds. The blue (lighter) line is the mean of the prediction, with the dark area indicating once the standard deviation and the light area denoting twice the standard deviation. The black (darker) line is the true value. The uncertainty in the predictions grows as we predict states further into the future. Eventually the algorithm gives a very high variance, which is its way of saying it is unsure about what will happen. Its estimates may be correct (like in the left plot) or incorrect (like in the right plot).

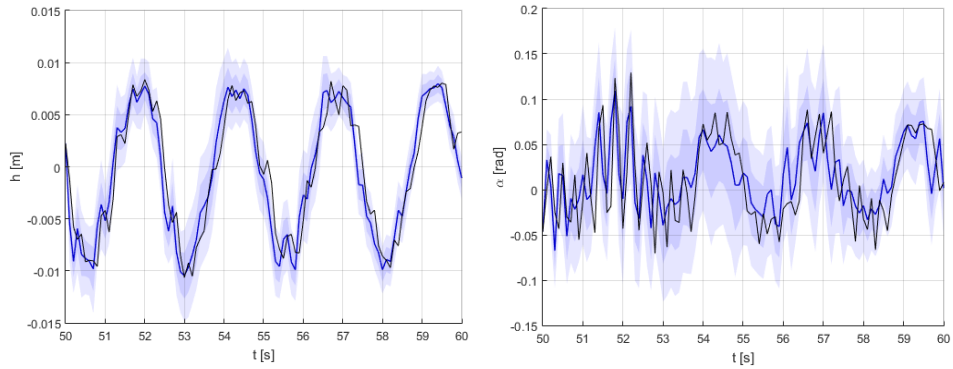

Figure 4.7: The prediction of the state development of the pitch-plunge system for the final ten seconds, when the SONIG algorithm is told to reduce the variance of its estimates by 5% after every time step.

reduce the variance of the state predictions by 5% after every time step. This seems like a rather arbitrary thing to do, but it does provide us with more accurate results, shown in Figure 4.7. So apparently it can sometimes be worthwhile to manually adjust the variance of predictions made by the SONIG algorithm.

From this experiment we can conclude that the SONIG algorithm is capable of identifying nonlinear systems with multiple states that are subject to measurement noise. Further research can look into what exactly the limits of the SONIG algorithm are. Can the SONIG algorithm also identify more complicated systems?

This is actually a very subjective question. I believe that, if plenty of measurements and computational resources are available, and if sufficient time is spent on properly tuning the hyperparameters, then pretty much any system can be identified through the SONIG algorithm. So the main question is not what the SONIG algorithm is capable of doing, but how easy it is to apply, especially when time, measurements and computational resources are limited. And that strongly depends on the background of the person applying the SONIG algorithm.

4.6. CONCLUSIONS AND RECOMMENDATIONS

We can conclude that the presented SONIG algorithm works as intended. Just like the FITC algorithm that it expands on, it is mainly effective when there are more measurements than the NIGP algorithm (or regular GP regression) can handle. The SONIG algorithm can then include the additional measurements very efficiently—incorporating each training point in constant runtime—resulting in a higher accuracy than what the NIGP algorithm could have achieved. However, even when this is not the case, the SONIG algorithm has on average a better performance than the NIGP algorithm, though it still needs the NIGP algorithm for hyperparameter tuning.

Though the SONIG algorithm can be used for any type of regression problem, it has been successfully applied, in its system identification set-up, to a non-linear black-box system identification problem. With the proper choice of hyperparameters and inducing input points, it outperformed existing state-of-the-art non-linear system identification

algorithms.

Nevertheless, there are still many improvements that can be made to the SONIG algorithm. For instance, to improve the accuracy of the algorithm, we can look at reducing some of the approximating assumptions, like the linearization assumption (4.14) or the assumption that higher order terms of Σ_{+_x} are negligible. Since the second derivative of $\mu_{u_i}^{n+1}$ from (4.17) already plays an important role, it is interesting to find what the effect would be of incorporating $\Sigma_{+_x}^2$.

Another way to improve the accuracy of the algorithm is to increase the number of inducing input points, but this will slow the algorithm down. To compensate, we could look into updating only the few nearest inducing input points (with the highest covariance) when incorporating a new training point. Experience has shown that updates hardly affect inducing inputs far away from the training point (with a low covariance) so this could lead to more efficient updates.

A final possible improvement would concern the addition of a smoothing step in the algorithm. Currently, early measurements are used to provide more accuracy for later measurements, but not vice versa. If we also walk back through the measurements, like in a smoothing algorithm, a higher accuracy might again be obtained.

5

Fixed-Structure Controller Synthesis through Reinforcement Learning

Abstract — In industrial applications like wind turbines, fixed-structure controllers are often desired. But for systems with large uncertainties, or for systems with mostly unknown system dynamics, it is often unclear as to how to choose the controller parameters. In this chapter we set up an algorithm that chooses the parameters of such a controller using only a limited amount of system interaction data. The algorithm applies Gaussian process tools to a reinforcement learning problem set-up to derive an approximation of the value function. This approximation is expressed in the system state and in the controller parameters. Then, by assuming a distribution of the initial state of the system, the value function approximation is expressed only as a function of the controller parameters. By subsequently optimizing this value function approximation, the optimal controller parameters with respect to the value function approximation can be found. The effectiveness of the proposed methodology has been shown in a simulation study.

A wind turbine is fundamentally a nonlinear system with large uncertainties. We consider such a system

$$\dot{\boldsymbol{z}}(t) = f(\boldsymbol{z}(t), \boldsymbol{u}(t)), \tag{5.1}$$

where $\boldsymbol{z}$ is the state vector and $\boldsymbol{u}$ is the input vector. (We use $\boldsymbol{z}$ instead of the more customary $\boldsymbol{x}$ because later on we will use $\boldsymbol{x}$ as input for Gaussian process regression.) In addition, we assume that we have a fixed-structure control law

$$\boldsymbol{u}(t) = C(\boldsymbol{z}(t), \boldsymbol{\theta}), \tag{5.2}$$

with $\boldsymbol{\theta}$ a constant vector (i.e., not dependent on the time or the state) of to-be-determined controller parameters.

In this chapter we assume that it is possible to exactly measure the state $\boldsymbol{z}$, while the system f is unknown. That is, we may have some model $\bar{f}$ such that $\dot{\boldsymbol{z}} \approx \bar{f}(\boldsymbol{z}, \boldsymbol{u})$, but this model differs from the true state transition function due to inaccuracies in modeling, manufacturing inaccuracies, external influences like temperature, aging effects like wear, and other reasons. The main question is how we can directly tune the controller parameters $\boldsymbol{\theta}$, optimizing a given value function.

The contents of this chapter have been published in Bijl et al. (2014). The chapter starts with a literature study (Section 5.1) broadly looking into methods of how to tackle the problem. Section 5.2 discusses the theory behind the proposed algorithm. Once the theory is in place, section 5.3 gives an overview of the algorithm. In section 5.4 an application of the algorithm is described. Section 5.5 closes off the chapter with conclusions and a discussion of the strengths and weaknesses of the presented algorithm.

5.1. LITERATURE OVERVIEW

We first study general methods of dealing with uncertainties (Section 5.1.1) and then focus on literature combining Gaussian process regression with reinforcement learning (Section 5.1.2).

5.1.1. METHODS TO DEAL WITH UNCERTAINTIES

A conventional way of dealing with uncertainties is through robust control, as described by Skogestad and Postlethwaite (2005). In this case hard bounds on the uncertainties are assumed to be known. Subsequently a controller is designed that is guaranteed to have a certain performance level for any possible system f within these bounds. This works well if the uncertainties are relatively small. However, if the possible differences between the model $\bar{f}$ and the system f become big, creating a controller that has sufficient performance for all possible systems $\bar{f}$ is not always possible.

A different way to tackle this problem is by using system measurements. Subsequently, it is possible to apply System Identification (SI) methods, as described by Verhaegen and Verdult (2007), to develop a system model. This model is then used to develop a controller. For linear systems, the theory on system identification is very well developed. For nonlinear systems, this is less so. In this case it generally requires a lot of system data before a model with sufficient accuracy is obtained. For industrial applications, obtaining a sufficient amount of system data is not always possible, mainly due to financial reasons. And if there is an insufficient amount of data, the identified system model will have relatively large uncertainties. Next to this, applying SI with a small data set is likely to result in model bias, as argued by Deisenroth and Rasmussen (2011). So in this case it may be beneficial to skip the SI step altogether.

Not using a system model at all is something that is often done in Reinforcement Learning (RL) applications. (See the work by Sutton and Barto (1998) for background theory on RL.) For instance, in Q-learning by Watkins (1989) the RL algorithm directly learns the Q-value of applying a certain input $\boldsymbol{u}$ (or action a) in a certain state $\boldsymbol{z}$ (or s). From this, the optimal input $\boldsymbol{u}$ can be obtained. The problem with the Q-learning algorithm and most other reinforcement learning algorithms, is that they are traditionally designed for discrete-time systems with discrete state and action spaces.

There are many RL algorithms that expand on this. For instance, Bertsekas and Tsitsiklis (1996) describe algorithms that can be applied to systems with continuous state and action spaces. They use various types of function approximators to deal with the continuous states and actions in the (discrete-time) systems. When the value function is approximated by a linear combination of basis functions, then the algorithm generally works well. Most such algorithms have proven convergence and result in a reasonable approximation of the value function, despite the limitations of the linear function approximator. Vrabie et al. (2013) describe an algorithm that expands these methods to the continuous time domain, but this algorithm requires the system dynamics to be affine in the input.

Another challenge is that RL often has the drawback of being data-inefficient. That is, RL algorithms require a large number of trials to learn a particular task. For many industrial problems in which system data is expensive to obtain, RL algorithms are therefore not suitable. An exception to this rule occurs when the tools used to model Gaussian Processes (GP) are implemented. These GP tools can be applied as function approximator, and because they take into account the uncertainties of their approximations and cleverly incorporate prior knowledge, they are generally known for their efficient use of input data.

5.1.2. COMBINATIONS OF RL AND GP IN LITERATURE

Reinforcement learning and Gaussian processes have not been combined often in literature. Dearden et al. (1998) did approximate a Q-function by using probability distributions for the Q-values. However, their algorithm worked for discrete state and action spaces, so no Gaussian process tools had to be used.

The first significant instance of applying GP to RL problems was by Rasmussen and Kuss (2004). Though promising, the proposed algorithm applies system identification through GP, in which every state parameter is independently modeled by a separate GP. This results in a highly computationally intensive algorithm. Next to that, the algorithm requires a significant number of maximizations of a Gaussian process during each policy iteration, which is also computationally problematic. This strongly limits the practical applicability of the algorithm.

Another combination of GP and RL was by Engel et al. (2005). In their article the authors come up with a method of determining the value function that is very similar to the way that we will propose here. However, they have not looked at an efficient method of selecting an action. Their suggestion is maximizing the Gaussian process value function with respect to the input $\boldsymbol{u}$ at each time step. This comes down to solving a nonlinear optimization problem at each time step, which is likely to be infeasible in practice.

Probably the most promising combination of GP and RL is by Deisenroth and Rasmussen (2011). Their PILCO algorithm (Probabilistic Inference for Learning Control) uses Gaussian processes to learn a system model, instead of a value function. Because of this, PILCO is very suitable for higher-dimensional problems, more so than the algorithm presented in this chapter. However, PILCO only works for specific types of fixed-structure controllers, which is a limitation. The algorithm presented in this chapter is able to tune controller parameters for any fixed-structure state controller $C(\boldsymbol{z},\boldsymbol{\theta})$.

5.2. UNDERLYING THEORY

In this section we examine how to apply Gaussian processes to reinforcement learning problems. In subsection 5.2.1 we present the problem from a reinforcement learning perspective. We briefly introduce Gaussian process tools in subsection 5.2.2 and apply them to our problem set-up in subsection 5.2.3. Finally, in subsection 5.2.4, we examine how we can use our data to select controller parameters.

5.2.1. REINFORCEMENT LEARNING SET-UP

To be able to judge the effectiveness of certain controller parameters $\boldsymbol{\theta}$, we need an optimality criterion. In reinforcement learning applications, a common choice is the value function V, describing the value of a certain state $\boldsymbol{z}(t)$ with respect to a given control law. It does this according to

$$\begin{aligned} V(\boldsymbol{z}(t),\boldsymbol{\theta}) &= \frac{\int_t^\infty \gamma^{\tau-t} r(\boldsymbol{z}(\tau), C(\boldsymbol{z}(\tau),\boldsymbol{\theta}))\, d\tau}{\int_t^\infty \gamma^{\tau-t}\, d\tau} \\ &= (-\log(\gamma)) \int_t^\infty \gamma^{\tau-t} r(\boldsymbol{z}(\tau), C(\boldsymbol{z}(\tau),\boldsymbol{\theta}))\, d\tau, \end{aligned} \tag{5.3}$$

where $\gamma < 1$ is a constant discount factor and r is a reward function chosen to encourage certain types of behavior. In the above relation, we have decided to divide by the (constant) sum of the discount factors. This has (among others) the benefit that it will shorten some of the subsequent equations.

Suppose now that a trial time step is performed. This experiment starts in state $\boldsymbol{z}(t)$, lasts T seconds (with T usually small) and puts the system in state $\boldsymbol{z}(t+T)$. In this case the expression for the value function can be rewritten to

$$V(\boldsymbol{z}(t),\theta) = (-\log(\gamma)) \int_t^{t+T} \gamma^{\tau-t} r(\boldsymbol{z}(\tau), C(\boldsymbol{z}(\tau),\boldsymbol{\theta}))\, d\tau + \gamma^T V(\boldsymbol{z}(t+T),\theta). \tag{5.4}$$

Note that T in the term γ^T is not a transpose but an exponent. After all, γ is a scalar.

The above integral is usually not analytically solvable. In such a case it is often easier to approximate the (varying) reward by a constant (mean) reward $\bar{r}$ throughout the trial time step T. As long as we take T to be small, such an approximation can be justified. (The definition of 'small' naturally depends on the time constants of the system involved.) The above equation then reduces to

$$V(\boldsymbol{z}(t),\boldsymbol{\theta}) = (1-\gamma^T)\bar{r}(\boldsymbol{z}(t), C(\boldsymbol{z}(t),\boldsymbol{\theta})) + \gamma^T V(\boldsymbol{z}(t+T),\boldsymbol{\theta}). \tag{5.5}$$

This recursive relation, akin to the Bellman equation, will prove to be useful when Gaussian process tools will be implemented.

5.2.2. THE NOTATION USED FOR GAUSSIAN PROCESS REGRESSION

In literature on reinforcement learning, it is customary to use a function approximator to approximate the value function. We will do something similar here. We will assume that the value function V is a Gaussian process, and consequentially approximate it using the tools developed to describe Gaussian processes. An important advantage of using

Gaussian processes is that we do not only get an estimate of the value function V, but also of the uncertainty (i.e. the variance) of our approximation.

Let us briefly look at the notation that we use for Gaussian process regression. It is the same notation as used by Rasmussen and Williams (2006). When we work with Gaussian processes, we usually do measurements y of a function $h(\boldsymbol{x})$, with $\boldsymbol{x}$ the (vector) input and y the scalar output. (We use a function $h(\boldsymbol{x})$ instead of the customary $f(\boldsymbol{x})$, because we already use f for the state transition function.) We assume that these measurements are corrupted by white noise with intensity σ_n^2. We can assemble all n sets of measurement data $(\boldsymbol{x}, y)$ into a matrix X and a vector $\boldsymbol{y}$ like

$$X = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} y_1 & \cdots & y_n \end{bmatrix}^T. \tag{5.6}$$

In the above definition, the indices indicate the measurement number. Subsequently, we define a prior mean function $m(\boldsymbol{x})$ and a prior covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$ for the output values of the function h, where $\boldsymbol{x}$ and $\boldsymbol{x}'$ can be any inputs to h. Customary functions are the zero mean function $m(\boldsymbol{x}) = 0$ and the squared exponential covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \lambda_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \Lambda^{-1} (\boldsymbol{x} - \boldsymbol{x}')\right), \tag{5.7}$$

with Λ a diagonal matrix of (squared) length scales.

Now examine a point $\boldsymbol{x}_*$ for which we want to estimate $h(\boldsymbol{x}_*)$, or alternatively the corresponding (noise-corrupted) measurement y_*. From our assumptions it follows that

$$\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(\boldsymbol{x}_*) \end{bmatrix}, \begin{bmatrix} k(X,X) & k(X,\boldsymbol{x}_*) \\ k(\boldsymbol{x}_*, X) & k(\boldsymbol{x}_*, \boldsymbol{x}_*). \end{bmatrix} + \sigma_n^2 I\right), \tag{5.8}$$

with $\mathcal{N}(.,.)$ denoting a multivariate Gaussian distribution. If we use the fact that the measurements $\boldsymbol{y}$ are known, then we can derive a distribution for y_*. Denoting $k(X,X) = K$, $k(X,\boldsymbol{x}_*) = K_*$ and $k(\boldsymbol{x}_*, \boldsymbol{x}_*) = K_{**}$, we get

$$y_* | \boldsymbol{y} \sim \mathcal{N}\left(m(\boldsymbol{z}_*) + K_*^T (K + \sigma_n^2 I)^{-1} (\boldsymbol{y} - m(X)), K_{**} - K_*^T (K + \sigma_n^2 I)^{-1} K_* + \sigma_n^2 I\right). \tag{5.9}$$

5.2.3. APPLYING GAUSSIAN PROCESSES TO REINFORCEMENT LEARNING

Equation (5.9) is a familiar equation in the field of Gaussian process regression. However, it cannot be directly applied to reinforcement learning problems. This is because in RL we do not get (noisy) measurements of the value function $V(\boldsymbol{x}, \boldsymbol{\theta})$. Instead, at every trial time step we find a relation between points in the value function. We can rewrite such a relation, equation (5.5), to

$$\begin{bmatrix} 1 & -\gamma^T \end{bmatrix} \begin{bmatrix} V(\boldsymbol{z}^{\boldsymbol{i}}, \theta) \\ V(\boldsymbol{z}^{\boldsymbol{f}}, \theta) \end{bmatrix} = \begin{bmatrix} (1 - \gamma^T)\bar{r} \end{bmatrix}, \tag{5.10}$$

where, for the sake of compactness of notation, we have shortened the initial state $\boldsymbol{z}(t)$ to $\boldsymbol{z}^{\boldsymbol{i}}$, the final state $\boldsymbol{z}(t+T)$ to $\boldsymbol{x}^{\boldsymbol{f}}$, and we have dropped the brackets after $\bar{r}$. If we do an additional trial time step, then the above turns into

$$\begin{bmatrix} 1 & -\gamma^{T_1} & 0 & 0 \\ 0 & 0 & 1 & -\gamma^{T_2} \end{bmatrix} \begin{bmatrix} V(\boldsymbol{z}_1^{\boldsymbol{i}}, \boldsymbol{\theta_1}) \\ V(\boldsymbol{z}_1^{\boldsymbol{f}}, \boldsymbol{\theta_1}) \\ V(\boldsymbol{z}_2^{\boldsymbol{i}}, \boldsymbol{\theta_2}) \\ V(\boldsymbol{z}_2^{\boldsymbol{f}}, \boldsymbol{\theta_2}) \end{bmatrix} = \begin{bmatrix} (1 - \gamma^{T_1})\bar{r}_1 \\ (1 - \gamma^{T_2})\bar{r}_2 \end{bmatrix}, \tag{5.11}$$

5

where the index denotes the trial number. In this way, we can expand the above equation for any number of trial time steps, even when these trials have different durations T.

Next, we can write the above relation as $M\boldsymbol{V} = \bar{\boldsymbol{r}}$, with M, $\boldsymbol{V}$ and $\bar{\boldsymbol{r}}$ defined accordingly. Subsequently, given that $M\boldsymbol{V} = \bar{\boldsymbol{r}}$, we can use Theorem 5.1 to find the posterior distribution of $\boldsymbol{V}$.

Theorem 5.1. *Consider a random variable $\boldsymbol{V} \sim \mathcal{N}(\boldsymbol{0}, K)$. If it is known that $M\boldsymbol{V} = \hat{\boldsymbol{r}}$, with M of full row rank, and if we have obtained a noisy measurement $\bar{\boldsymbol{r}}$ of $\hat{\boldsymbol{r}}$, distorted by zero-mean Gaussian white noise with covariance Σ_n, then*

$$\boldsymbol{V}|\bar{\boldsymbol{r}} \sim \mathcal{N}(KM^T(MKM^T+\Sigma_n)^{-1}\bar{\boldsymbol{r}}, K - KM^T(MKM^T+\Sigma_n)^{-1}MK). \tag{5.12}$$

Proof. This can be proven through (5.9). The proof is based on a coordinate transformation of $\boldsymbol{V}$. We define

$$\begin{bmatrix}\boldsymbol{V}'_{\boldsymbol{a}}\\ \boldsymbol{V}'_{\boldsymbol{b}}\end{bmatrix} = \boldsymbol{V}' = P\boldsymbol{V} = \begin{bmatrix}M\\ N\end{bmatrix}\boldsymbol{V}, \tag{5.13}$$

with the square matrix P defined as shown above. Here we choose N such that its rows span the null space of the rows of M. This ensures that $MN^T = 0$ and P is invertible. In addition, the rows of N should all have unit length and be orthogonal, giving us $NN^T = I$.

By applying $\boldsymbol{V} = P^{-1}\boldsymbol{V}'$, it follows that the relation $M\boldsymbol{V} = \hat{\boldsymbol{r}}$ reduces to $\boldsymbol{V}'_{\boldsymbol{a}} = \hat{\boldsymbol{r}}$. That is, $\boldsymbol{V}'_{\boldsymbol{a}}$ is measured, just like in equation (5.9) we had a measurement $\boldsymbol{y}$. Identically, $\boldsymbol{V}'_{\boldsymbol{b}}$ is not measured, just like y_* was not measured. Noting that $\boldsymbol{V}' \sim \mathcal{N}(\boldsymbol{0}, PKP^T)$, we find

$$\begin{aligned}\begin{bmatrix}\boldsymbol{V}'_{\boldsymbol{a}}\\ \boldsymbol{V}'_{\boldsymbol{b}}\end{bmatrix}|\bar{\boldsymbol{r}} \sim \mathcal{N}\Bigg(&\begin{bmatrix}MKM^T\left(MKM^T+\Sigma_n\right)^{-1}\bar{\boldsymbol{r}}\\ NKM^T\left(MKM^T+\Sigma_n\right)^{-1}\bar{\boldsymbol{r}}\end{bmatrix}, \begin{bmatrix}MKM^T - MKM^T\left(MKM^T+\Sigma_n\right)^{-1}MKM^T & \\ NKM^T - NKM^T\left(MKM^T+\Sigma_n\right)^{-1}MKM^T & \\ MKN^T - MKM^T\left(MKM^T+\Sigma_n\right)^{-1}MKN^T & \\ NKN^T - NKM^T\left(MKM^T+\Sigma_n\right)^{-1}MKN^T & \end{bmatrix}\Bigg)\\ = \mathcal{N}\Big(&PKM^T\left(MKM^T+\Sigma_n\right)^{-1}\bar{\boldsymbol{r}}, PKP^T - PKM^T\left(MKM^T+\Sigma_n\right)^{-1}MKP^T\Big).\end{aligned} \tag{5.14}$$

Next, by transforming the result back, equation (5.12) is directly obtained.

Note that we have assumed that the mean function $m(\boldsymbol{x}) = 0$, for the sake of keeping the notation brief. If this is not the case, the mean function $m(\boldsymbol{x})$ can directly be incorporated. Also note that (5.12) is actually a generalization of (5.9). (That is, apart from the assumption that $m(\boldsymbol{x}) = 0$.) If $M = \begin{bmatrix}I & 0\end{bmatrix}$, then (5.12) directly reduces back to equation (5.9). □

The above Theorem is a novel way of applying GP regression to linear constraints of function values for different input points, published about in Bijl et al. (2014). Something very similar was already done by Engel et al. (2003, 2005), although the above generalization of GP regression was not noted in that work. Similar things have been set up by Salzmann and Urtasun (2010), who constrain the different output elements of a multi-output function for the same input point, and Jidling et al. (2017), where constraints on derivatives can also be incorporated and built into the covariance function.

According to the Theorem 5.1, Σ_n equals the covariance matrix of $\bar{\boldsymbol{r}}$. In this chapter we will use $\Sigma_n = (1-\gamma^T)^2\sigma_n^2 I$, where σ_n^2 is the variance of the noise present when measuring the mean reward $\bar{r}$. For our application, we can find the matrix K through a covariance function $k(\boldsymbol{z},\boldsymbol{\theta},\boldsymbol{z}',\boldsymbol{\theta}')$. In the case where we only have one trial time step, K equals

$$K = \begin{bmatrix} k(\boldsymbol{z_1^i},\boldsymbol{\theta_1},\boldsymbol{z_1^i},\boldsymbol{\theta_1}) & k(\boldsymbol{z_1^i},\boldsymbol{\theta_1},\boldsymbol{z_1^f},\boldsymbol{\theta_1}) \\ k(\boldsymbol{z_1^f},\boldsymbol{\theta_1},\boldsymbol{z_1^i},\boldsymbol{\theta_1}) & k(\boldsymbol{z_1^f},\boldsymbol{\theta_1},\boldsymbol{z_1^f},\boldsymbol{\theta_1}) \end{bmatrix}. \tag{5.15}$$

If we have performed n trial time steps, then K expands accordingly to a $2n\times 2n$ covariance matrix.

It is customary to decouple the covariance function $k(\boldsymbol{z},\boldsymbol{\theta},\boldsymbol{z}',\boldsymbol{\theta}')$ into $k_z(\boldsymbol{z},\boldsymbol{z}')k_\theta(\boldsymbol{\theta},\boldsymbol{\theta}')$. Subsequently, we can use a squared exponential function for both k_z and k_θ. This would result in

$$k(\boldsymbol{z},\boldsymbol{\theta},\boldsymbol{z}',\boldsymbol{\theta}') = \lambda_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{z}')^T\Lambda^{-1}(\boldsymbol{z}-\boldsymbol{z}') - \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}')^T\Lambda_\theta^{-1}(\boldsymbol{\theta}-\boldsymbol{\theta}')\right). \tag{5.16}$$

Now the framework is in place to give us an estimate of the value $V(\boldsymbol{x}_*,\boldsymbol{\theta}_*)$ for some state $\boldsymbol{z}_*$ and set of controller parameters $\boldsymbol{\theta}_*$. To get this estimate, we can add $V(\boldsymbol{z}_*,\boldsymbol{\theta}_*)$ to the value vector $\boldsymbol{V}$, add a corresponding column of zeros to the constraint matrix M and add a column and row to K. Next, we can apply relation (5.12) to derive the posterior distribution of $\boldsymbol{V}$ and hence of $V(\boldsymbol{z}_*,\boldsymbol{\theta}_*)$. If we do this for multiple points, we can plot $E[V(\boldsymbol{z},\boldsymbol{\theta})]$ with respect to $\boldsymbol{z}$ and/or $\boldsymbol{\theta}$.

It is interesting to note that the computational complexity of this method does not depend on the size of Z. Instead, it depends on the size of $\bar{\boldsymbol{r}}$. After all, the bottleneck in equation (5.12) is the inversion of the matrix $(MKM^T+\sigma_n^2 I)$. This matrix has size $n\times n$, with n the number of time steps examined. So the runtime of this algorithm will be $\mathcal{O}(n^3)$.

5.2.4. DETERMINING THE EXPECTED VALUE OF CONTROLLER PARAMETERS

With the method described previously, it is possible to approximate the value $V(\boldsymbol{z},\boldsymbol{\theta})$, as a function of the state $\boldsymbol{z}$ and a set of controller parameters $\boldsymbol{\theta}$. By maximizing $V(\boldsymbol{z},\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, for a constant $\boldsymbol{z}$, it is possible to find the optimal set of controller parameters for each state. That is, as much as is allowed by the accuracy of the approximation of $V(\boldsymbol{z},\boldsymbol{\theta})$.

However, for our problem we cannot let the controller parameters/gains vary with each state. Instead, a controller with constant gains is required. Choosing the optimal $\boldsymbol{\theta}$ now takes place differently.

Suppose that, from experience, it is known that the prior state $\boldsymbol{z}$ is distributed according to $p(\boldsymbol{z})$. Then the set of optimal controller parameters $\boldsymbol{\theta_{opt}}$, that (per definition) maximizes the expected value, is found through marginalization as

$$\boldsymbol{\theta_{opt}} \equiv \arg\max_{\boldsymbol{\theta}} \int_Z V(\boldsymbol{z},\boldsymbol{\theta})\,p(\boldsymbol{z})\,d\boldsymbol{z}. \tag{5.17}$$

Finding an analytic solution for this relation is generally not possible. However, if we use a squared exponential covariance function like in equation (5.16), and if $p(\boldsymbol{z})$ also has a Gaussian distribution – that is, $\boldsymbol{z}\sim\mathcal{N}(\boldsymbol{\mu_z},S)$ – then the integral in equation (5.17) can be

solved analytically. The mathematics behind this are explained by Girard et al. (2003) and Candela et al. (2003). Application results in

$$\bar{V}(\boldsymbol{\theta}) \equiv E\left[V(\boldsymbol{z},\boldsymbol{\theta})|\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu_z}, S)\right] = \boldsymbol{l}^T M^T (MKM^T + \sigma_n^2 I)^{-1} \bar{\boldsymbol{r}}, \tag{5.18}$$

where the elements l_i of the column vector $\boldsymbol{l}$ depend both on the current controller parameters $\boldsymbol{\theta}$ and the respective experiment input data $(\boldsymbol{z_i}, \boldsymbol{\theta_i})$. To be precise, they are given by

$$\begin{aligned} l_i &= k_S(\boldsymbol{z_i}, \boldsymbol{\theta_i}, \boldsymbol{\mu_z}, \boldsymbol{\theta}) \\ &\equiv \lambda_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{z_i} - \boldsymbol{\mu_z})^T (\Lambda + S)^{-1} (\boldsymbol{z_i} - \boldsymbol{\mu_z}) - \frac{1}{2}(\boldsymbol{\theta_i} - \boldsymbol{\theta}) \Lambda_\theta^{-1} (\boldsymbol{\theta_i} - \boldsymbol{\theta})\right) \frac{\sqrt{|\Lambda|}}{\sqrt{|\Lambda + S|}}. \end{aligned} \tag{5.19}$$

Note that the subscript i now is an index, indicating a relation to the i^{th} element of $\boldsymbol{V}$ and its corresponding state $\boldsymbol{z_i}$ and controller parameters $\boldsymbol{\theta_i}$. Also note that, for reasons of notation, the conditioning on $\bar{\boldsymbol{r}}$ has been omitted.

5

The function $\bar{V}(\boldsymbol{\theta})$ is called the approximated controller quality function. This quality function can subsequently be plotted with respect to the controller parameters $\boldsymbol{\theta}$. Such a plot can then be used to gain more understanding of how the value of the system varies with respect to the controller parameters. Additionally, it is possible to use an optimization method to maximize this function, automatically tuning the controller parameters.

5.3. THE GP CONTROLLER TUNING ALGORITHM

The ideas and theories of the previous section can be assembled into a Gaussian process controller tuning algorithm. To automatically tune the controller parameters, the following steps need to be taken.

- Generate n measurements. Record the initial state $\boldsymbol{x^i}$, the final state $\boldsymbol{x^f}$, the weighted average reward $\bar{r}$, the applied controller parameters $\boldsymbol{\theta}$ and the experiment duration T.
- Assemble M and $\bar{\boldsymbol{r}}$ as shown in equation (5.11) and the matrix K as shown in equation (5.15).
- Optionally, tuning of hyperparameters like λ_f, Λ and Λ_θ can be done through evidence maximization (also known as relevance determination) as described by Rasmussen and Williams (2006) (chapter 5).
- Derive the approximated controller quality function $\bar{V}(\boldsymbol{\theta})$ through (5.18).
- Optimize $\bar{V}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ to find the controller parameters.

The resulting algorithm has a relatively high computational complexity. This is because it requires inversion of the matrix $(MKM^T + \sigma_n^2 I)$, which is of size $n \times n$. The run-time of the algorithm hence is $\mathcal{O}(n^3)$. This makes the algorithm unsuitable for systems in which a lot of data is available. In such cases conventional system identification algorithms can be applied, or alternative algorithms like the one presented in Chapter 4. The algorithm presented here is mainly suitable for nonlinear systems for which a limited data set is available.

5.4. APPLICATION IN AN EXPERIMENT

To test the performance of the algorithm, we apply it to a well-known problem: stabilizing an inverted pendulum.

5.4.1. THE PROBLEM SET-UP

The equation of motion of an inverted pendulum is given by

$$mr^2\ddot{\alpha} + c\dot{\alpha} - mg\sin(\alpha) = u, \tag{5.20}$$

where m is the mass (1 kg), r is the radius (1 m), α is the angle (in radians) with respect to the vertical, c is the friction coefficient (0.5 kg m/s), g is the gravitational constant (10 m/s^2) and u is the input to the system (in Nm).

The problem to be solved is the maximization of the value function of equation (5.3). The parameter γ is set to 1/2, and as reward function we will use

$$r(\boldsymbol{z},\boldsymbol{u}) = -\frac{1}{2}\left(\left(\frac{\alpha}{\bar{\alpha}}\right)^2 + \left(\frac{\dot{\alpha}}{\bar{\dot{\alpha}}}\right)^2 + \frac{1}{10}\left(\frac{u}{\bar{u}}\right)^2\right). \tag{5.21}$$

In this equation, $\bar{\alpha}$, $\bar{\dot{\alpha}}$ and $\bar{u}$ are normalizing constants. We define $\bar{\alpha}$ as 45 degrees. We then define $\bar{\dot{\alpha}} = 140$ deg/s, which is (approximately) the angular velocity that the pendulum gets at $\alpha = 45°$ when released from a stationary position at $\alpha \approx 0°$. Finally, we define $\bar{u} = 5\sqrt{2}$ Nm, which is the input torque required to keep the pendulum stationary at $\alpha = 45°$. In the reward function above, the first term is present to make the system move the pendulum to the upright position, making α zero. The second term is to prevent the system from making excessively violent movements, which are likely to result in an overshoot. The third term is present to prevent the system from applying excessively large inputs. Since it is not the priority to strongly limit the input, this term has a lower weight.

For this problem, we assume that (due to external reasons) we are restricted to a PD controller architecture $u = -K_p\alpha - K_d\dot{\alpha}$. If the proportional gain K_p is high enough to overcome the gravitational force, and if the derivative gain K_d is positive (adding some damping), the inverted pendulum is already stabilized. The question remains which controller gains maximize the value function. Very aggressive controllers give high velocities $\dot{\alpha}$ and inputs u, while very cautious controllers are slow at reducing α.

5.4.2. APPLICATION OF THE CONTROLLER TUNING ALGORITHM

To solve the problem, the algorithm of section 5.3 will be applied. First of all data is generated. For the given system, a total of $n = 200$ trial time steps have been run. For every trial time step, the system was given a random initialization

$$\begin{bmatrix}\alpha_0\\ \dot{\alpha}_0\end{bmatrix} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{z}}, S) = \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}(\bar{\alpha}/2)^2 & 0\\ 0 & \left(\bar{\dot{\alpha}}/2\right)^2\end{bmatrix}\right). \tag{5.22}$$

Next to this, the design choice was made to let $\log(K_p)$ and $\log(K_d)$ be uniformly distributed, with $\log(K_p) \in [5/2, 9/2]$ and $\log(K_d) \in [3/2, 7/2]$. Both K_p and K_d were kept constant during a trial time step.

The duration of each trial time step was 0.1 seconds. For each time step, the initial state $\boldsymbol{z}^{\boldsymbol{i}}$ and the final state $\boldsymbol{z}^{\boldsymbol{f}}$ was recorded, as well as an estimate $\bar{r}$ of the average reward

received during the time step. Also the controller parameters $\boldsymbol{\theta} = \begin{bmatrix} K_p & K_d \end{bmatrix}^T$ were noted. Based on this, the matrices M and K and the vector $\bar{\boldsymbol{r}}$ were set up.

The covariance function used was given in equation (5.16). For this covariance function, evidence maximization to tune the hyperparameters has been applied, using a gradient descent method. Of course the exact results varied per algorithm execution, because random inputs have been used. However, the general trends were the same per test run. What we will look at here is a single test run that can be seen as an 'average' run. The resulting matrices Λ and Λ_θ were

$$\Lambda = \begin{bmatrix} 0.68 & 0 \\ 0 & 35.9 \end{bmatrix} \quad \text{and} \quad \Lambda_\theta = \begin{bmatrix} 3.8 & 0 \\ 0 & 5.0 \end{bmatrix}. \tag{5.23}$$

After tuning, λ_f^2 became 0.22 and σ_n^2 became $2.0 \cdot 10^{-6}$. Furthermore, it has been assumed that the prior mean $m(\boldsymbol{z})$ of the value function was a constant value μ. This constant μ has also been tuned and was set to -0.86.

It is interesting to note that, should either K_p or K_d have no influence on the value function, then the corresponding length parameter in Λ_θ would increase to infinity during evidence maximization. Because of this, it is possible to determine which controller parameters are useful and which ones have less effect. In this case all elements of Λ_θ converged to a finite value, so all controller parameters have a significant effect on the value.

Next, the approximated controller quality function $\bar{V}(\theta)$ was derived through relation (5.18). When doing this, we assumed that our prior distribution of $\boldsymbol{z}$ was $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{z}}, S)$, with $\boldsymbol{\mu}_{\boldsymbol{z}}$ and S as defined in equation (5.22). The resulting approximation is shown in figure 5.1.

Some interesting things can be derived from figure 5.1. First of all, the optimal set of parameters within the ascribed interval can be obtained. For this test run, $\bar{V}(\boldsymbol{\theta})$ was at a maximum for $\log(K_p) = 3.61$ and $\log(K_d) = 2.57$. For these two controller parameters the approximated value was $\bar{V}(\boldsymbol{\theta}) = -0.0488$. Other experiment runs gave log-gain values that were on average 0.2 higher or lower, but the estimated value stayed at -0.0488, with deviations of roughly 10^{-4}.

Secondly, figure 5.1 also shows how well the controller performs for other parameter combinations. If K_p and K_d are both small, or are both big, then the system will have a reasonable performance. However, if K_p is large and K_d is small, then the system gets oscillations, resulting in a low value. Similarly, if K_p is small and K_d is large, then the system behaves sluggishly, again giving a low value. This is confirmed by figure 5.2, which shows simulation runs for various values of K_p and K_d.

5.4.3. COMPARISON WITH ANALYTIC RESULTS

To find out more about the accuracy of the algorithm, it is useful to compare it with analytic results. This is complicated, because the system that we examined was nonlinear. However, it can be linearized. (In equation (5.20) replace $\sin(\alpha)$ by α.) For this linear system, it is possible to derive the value function $V(\boldsymbol{z}, \boldsymbol{\theta})$ analytically. Also the expected value

$$E[V(\boldsymbol{z}, \boldsymbol{\theta}) | \boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, S)] \tag{5.24}$$

can be derived analytically. Numerically optimizing this with respect to $\boldsymbol{\theta}$ gave us the true optimal parameters for the linear system.

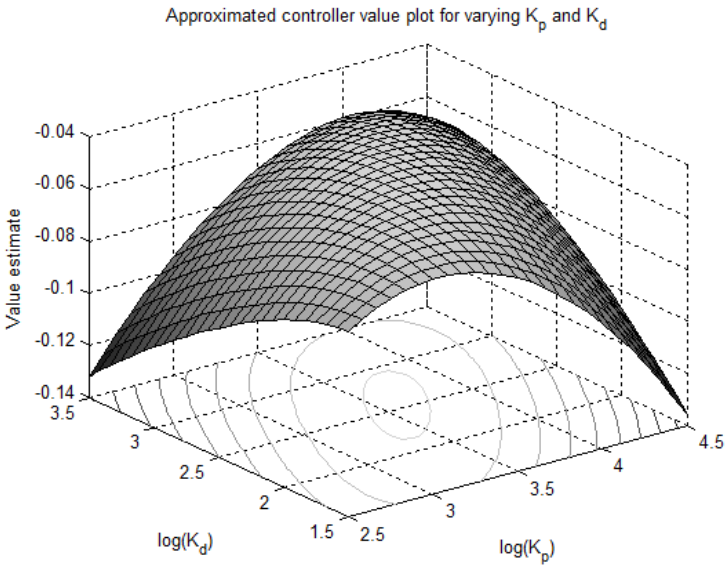

Figure 5.1: A plot of the approximated value with respect to the controller gains K_p and K_d. The plot was made based on 200 trial time steps. It is clear that the controller values have an optimum.

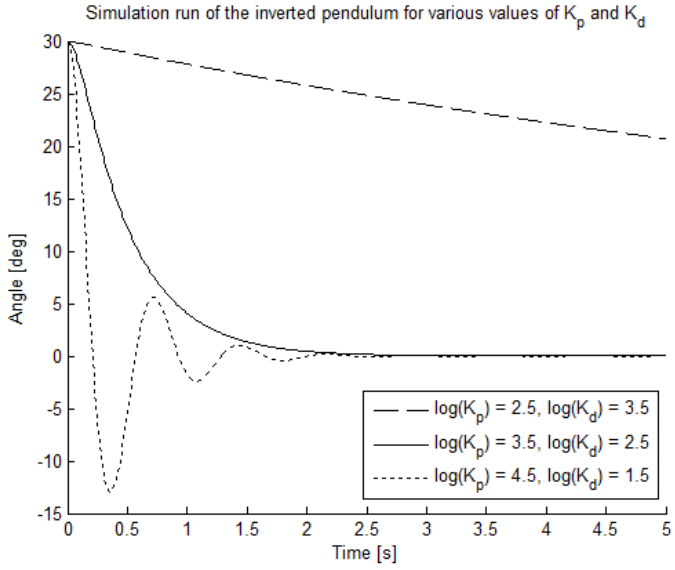

Figure 5.2: Simulation runs of the inverted pendulum for various values of K_p and K_d. The system was put in an initial state of $\alpha_0 = 30$ deg and $\dot{\alpha}_0 = 0$ deg/s. While some controllers are too passive, others are too aggressive, resulting in oscillations. The optimal controller provides fast convergence without oscillations.

The results were very similar to what our approximation predicted. The above quantity was at a maximum when $\log(K_p) = 3.58$ and $\log(K_d) = 2.46$. The corresponding value that was obtained was $V(\theta) = -0.0489$. So, while the exact gains giving the maximum value are slightly different from what we got from our approximation, this did not significantly affect the system value.

An interesting question is, 'What causes the difference between the approximation given by our algorithm and the analytical results for the linear system?' To find that out, the parameter tuning algorithm can also be applied to the linear system. This has been set up, and the results were not significantly different. This shows that the deviation was mainly caused by inaccuracies in the approximation, and was not due to linearizing the system.

5.5. CONCLUSIONS AND DISCUSSION

In this chapter an algorithm has been set up that can tune the parameters of a fixed-structure controller, when applied to a system. It can be used for any system type and no model of the system is required. Only a set of system experiment data is needed with varying states and varying controller parameters. When applied to a test system, the algorithm proved to be functional and to give accurate results, even when little system data was available.

The algorithm is computationally quite intensive. The run-time is $\mathcal{O}(n^3)$, with n the number of trial time steps. It is therefore not suitable for systems with lots of measurement data. Instead, the algorithm was designed for systems for which a limited amount of measurement data is available. The algorithm uses this measurement data quite efficiently. In a test case in which two controller gains needed to be tuned, the gains were put close to the optimum using only 200 random trial time steps.

Other than the run-time, the main downside of the algorithm is that it requires complete knowledge of the state $\boldsymbol{z}$. This data is required by the covariance function k when setting up the covariance matrix K. Possibly, the system may also have a reasonable performance when, instead of the state $\boldsymbol{z}$, some output vector $\boldsymbol{y}$ is passed to the covariance function k. In this case the controller C would of course also reduce from a state controller to an output controller. How well such a set-up would work is a possible subject for future research.

Another very interesting topic for future research would be to enable the algorithm to apply online learning. Currently, the algorithm takes a set of measurement data and uses it to tune control parameters offline. When this can be done online, during the operation of the system, then an extra set of applications opens up. One possibility here would be to implement the techniques discussed in Section 4. However, these techniques also cannot directly be applied to time-varying or parameter-varying systems. So this remains an avenue for further research.

6

A SEQUENTIAL MONTE CARLO APPROACH TO BAYESIAN OPTIMIZATION

Abstract — Bayesian optimization through Gaussian process regression is an effective method of optimizing an unknown function for which every measurement is expensive. It approximates the objective function and then recommends a new measurement point to try out. This recommendation is usually selected by optimizing a given acquisition function. After a sufficient number of measurements, a recommendation about the maximum is made. However, a key realization is that the maximum of a Gaussian process is not a deterministic point, but a random variable with a distribution of its own. This distribution cannot be calculated analytically. Our main contribution is an algorithm, inspired by sequential Monte Carlo samplers, that approximates this maximum distribution. Subsequently, by taking samples from this distribution, we enable Thompson sampling to be applied to (armed-bandit) optimization problems with a continuous input space. All this is done without requiring the optimization of a nonlinear acquisition function. Experiments have shown that the resulting optimization method has a competitive performance at keeping the cumulative regret limited.

Consider the problem of maximizing a continuous nonlinear reward function $f(\boldsymbol{x})$ (or equivalently minimizing a cost function) over a compact set X_f. Here $f(\boldsymbol{x})$ could for instance be the DEL from Section 1.2.4 and X_f be the set of possible controller parameters. In the case where $f(\boldsymbol{x})$ can be easily evaluated, where derivative data is available and where the function is convex (or concave), the solution is relatively straightforward, as is for instance discussed by (Boyd and Vandenberghe, 2004). However, we will consider the case where convexity and derivative data are not known. In addition, every function evaluation is expensive and we can only obtain noisy measurements of the function. In

this case we need a data-driven approach to optimize the function. That is what this chapter is about.

We open up with Section 6.1, formulating the exact problem and examining what solutions to this problem are available in literature. We then present a new algorithm, inspired by sequential Monte Carlo methods, that approximates the maximum distribution in Section 6.2. We also analyze it and examine how we can use it to apply Thompson sampling. Experimental results are presented in Section 6.3, with conclusions and recommendations given in Section 6.4.

6.1. PROBLEM FORMULATION AND LITERATURE OVERVIEW

The main set-up of *Bayesian optimization* is as follows. We have an unknown continuous function $f(\boldsymbol{x})$. For this function, we can try out certain inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ from the compact set X_f. After selecting a so-called *trial input* $\boldsymbol{x_k}$, we feed it into the function and obtain a noisy measurement $y_k = f(\boldsymbol{x_k}) + \varepsilon$, with $\varepsilon = \mathcal{N}\left(0, \sigma_n^2\right)$. We then use all measurements obtained so far to make a Bayesian approximation of the function $f(\boldsymbol{x})$, based on which we choose the next trial input $\boldsymbol{x_{k+1}}$.

In particular, we can approximate $f(\boldsymbol{x})$ through Gaussian process regression (Rasmussen and Williams, 2006). This gives us a mean $\mu(\boldsymbol{x})$ and a standard deviation $\sigma(\boldsymbol{x})$ of our estimate for $f(\boldsymbol{x})$, which can then be used to determine $\boldsymbol{x_{k+1}}$. The resulting optimization method is also known as *Gaussian process optimization*.

Bayesian methods like Gaussian process regression are known to efficiently deal with data, requiring only little data to make relatively accurate approximations. This makes these techniques suitable for a data-driven approach to problems in which data is expensive.

The main question is how to choose the trial inputs $\boldsymbol{x_k}$. There are two different problem set-ups here. In the first, after performing all n measurements, we have to give a recommendation $\hat{\boldsymbol{x}}^*$ of what we believe is the true optimum $\boldsymbol{x}^*$. The difference $f^* - \hat{f}^*$ between the corresponding function values $f^* \equiv f(\boldsymbol{x}^*)$ and $\hat{f}^* \equiv f(\hat{\boldsymbol{x}}^*)$ is known as the *error* or the *instantaneous regret*. As such, this problem set-up is known as the *error minimization set-up* or also as the *probabilistic global optimization* problem. It is useful in applications like sensor placement (Osborne, 2010) and controller tuning in damage-free environments (Lizotte et al., 2007, Marco et al., 2016). These are all applications in which every trial input (every experiment) has the same high set-up cost.

In the second problem set-up, our aim is to maximize the sum of all the rewards $f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_n})$, which is known as the *value* V. Equivalently, we could also minimize the *(cumulative) regret*

$$\sum_{k=1}^{n} \left(f^* - f(\boldsymbol{x_k})\right) = nf^* - V. \tag{6.1}$$

This set-up is known as the *regret minimization set-up* or also as the *continuous armed bandit problem*. It is useful in applications like advertisement optimization (Pandey and Olston, 2007) or controller tuning for damage minimization (see Section 6.3.4). These are applications where the reward or cost of an experiment actually depends on the result of the experiment. Early work on this is summarized by Jones et al. (1998). Because we will look at applications from the latter category, we will focus on the regret minimization

set-up. However, with the set-ups being similar, we also take error minimization strategies into account.

6.1.1. EXISTING ERROR MINIMIZATION METHODS

Several Bayesian optimization methods already exist. Good overviews are given by Lizotte (2008), Brochu et al. (2010), Shahriari et al. (2016), though we will provide a brief summary here. The recurring theme is that, when selecting the next input $\boldsymbol{x_k}$, we optimize an *Acquisition Function* (AF). In literature, the discussion mainly concerns selecting and tuning an AF.

The first to suggest the *Probability of Improvement* (PI) acquisition function was Kushner (1964). This function is defined as $\text{PI}(\boldsymbol{x}) = \mathbb{P}(f(\boldsymbol{x}) \geq y_+)$, where $\mathbb{P}(A)$ denotes the probability of event A to occur and y_+ denotes the highest value of the observation obtained so far. This was expanded by Torn and Zilinskas (1989), Jones (2001) to the form $\text{PI}(\boldsymbol{x}) = \mathbb{P}(f(\boldsymbol{x}) \geq y_+ + \xi)$, with ξ a tuning parameter trading off between exploration (high ξ) and exploitation (zero ξ).

Later on, Mockus et al. (1978) suggested an AF which also takes the magnitude of the potential improvement into account. It is known as the *Expected Improvement* (EI) acquisition function $\text{EI}(\boldsymbol{x}) = \mathbb{E}\left[\max(0, f(\boldsymbol{x}) - y_+)\right]$. Similar methods were used by others. For instance, multi-step lookahead was added by Osborne (2010), a trust region to ensure small changes to the tried inputs $\boldsymbol{x_k}$ was used by Park and Law (2015), and an additional exploration/exploitation parameter ξ similar to the one used in the PI AF was introduced by Brochu et al. (2010). An analysis was performed by Vazquez and Bect (2010).

Alternatively, Cox and John (1997) suggested the *Upper Confidence Bound* (UCB) acquisition function $\text{UCB}(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \kappa\sigma(\boldsymbol{x})$. Here the parameter κ determines the amount of exploration/exploitation, with high values resulting in more exploration. Often $\kappa = 2$ is used. The extreme case of $\kappa = 0$ is also known as the *Expected Value* (EV) acquisition function $\text{EV}(\boldsymbol{x}) = \mu(\boldsymbol{x})$. It applies only exploitation, so it is not very useful by itself. Methods to determine the value of κ optimizing regret bounds were studied by Srinivas et al. (2012).

A significant shift in focus was made through the introduction of *entropy search*. This method was first developed by Villemonteix et al. (2009), although Hennig and Schuler (2012) independently set up a similar method and introduced the name entropy search. A more efficient version of this method called *predictive entropy search* was developed by Hernández-Lobato et al. (2014). The main idea for both these methods is to look at the so-called *maximum distribution*: the probability $p_{\max}(\boldsymbol{x}) \equiv \mathbb{P}(\boldsymbol{x} = \boldsymbol{x}^*)$ that a certain point $\boldsymbol{x}$ equals the (unknown) optimum $\boldsymbol{x}^*$, or for continuous problems the corresponding probability density. We then focus on the relative entropy (the Kullback-Leibler divergence) of the maximum distribution compared to the flat probability density function over X_f. Initially this relative entropy is zero, but the more information is gained, the higher this relative entropy becomes. As such, it is useful to pick the trial point $\boldsymbol{x_k}$ which is expected to increase the relative entropy the most.

At the same time, *portfolio methods* were developed with the aim to optimally use a whole assortment (a portfolio) of acquisition functions. These methods were introduced by Hoffman et al. (2011), using results from Auer et al. (1995), Chaudhuri et al. (2009) and subsequently expanded on by Shahriari et al. (2014), who suggested to use the change in entropy as criterion to select recommendations.

6.1.2. EXISTING REGRET MINIMIZATION METHODS

In the error minimization set-up, the focus is on obtaining as much information as possible. The regret minimization set-up is more difficult, as it involves a trade-off between obtaining information and incurring costs (regret). Here, most of the research has focused on the case where the number of possible inputs $\boldsymbol{x}$ is finite. It is then known as the armed bandit problem and has been analyzed by for instance Kleinberg (2004), Grünewälder et al. (2010), de Freitas et al. (2012).

One of the more promising acquisition methods for the armed bandit problem is *Thompson sampling*. This method was first suggested by Thompson (1933) and has more recently been analyzed by Chapelle and Li (2011), Agrawal and Goyal (2012). It is fundamentally different from other methods, because it does not use an acquisition function. Instead, we select an input point $\boldsymbol{x}$ as the next trial point $\boldsymbol{x_k}$ with probability equal to the probability that $\boldsymbol{x}$ is the optimal input $\boldsymbol{x}^*$. This is equivalent to sampling $\boldsymbol{x_k}$ from the maximum distribution $p_{\max}(\boldsymbol{x})$. Generally this distribution is not known though. When only finitely many different input points $\boldsymbol{x}$ are possible, the solution is to consider the vector $\boldsymbol{f} \equiv f(X)$ of all possible function outputs. Using Bayesian methods, we approximate $\boldsymbol{f}$ as a random variable, take a sample $\hat{\boldsymbol{f}}$ from it, find for which input point $\boldsymbol{x}$ this sample has its maximum, and subsequently use that input $\boldsymbol{x}$ as the next trial point $\boldsymbol{x_k}$.

This method has proven to work well when the number of input points is finite. When there are infinitely many possible input points, like in continuous problems, it is impossible to sample from $\boldsymbol{f}$. This means that a new method to sample from the maximum distribution $p_{\max}(\boldsymbol{x})$ is needed. However, in literature this maximum distribution is not studied much at all. The idea of it was noted (but not evaluated) by Lizotte (2008). The maximum distribution was calculated by Villemonteix et al. (2009) through a brute force method. An expansion to this was developed by Hennig and Schuler (2012), who used a method from Minka (2001) to approximate the minimum distribution. Though the approximation method used is quite accurate, it has a runtime of $\mathcal{O}\left(n^4\right)$, making it infeasible to apply to most problems. An alternative method was described by Hernández-Lobato et al. (2014) who approximated function samples of a Gaussian process through a finite number of basis functions and then optimized these function samples to generate samples from the maximum distribution. Though effective, this method requires solving a nonlinear optimization problem for each sample, which is computationally expensive and subject to the risk of finding only a local optimum.

It is exactly for this reason that a computationally more efficient method of approximating the maximum distribution, for instance through numerical methods, can be very useful. So in the upcoming section we look at how such an algorithm can be set up.

6.2. FINDING THE MAXIMUM DISTRIBUTION

This section discusses the set-up of an algorithm to find/approximate the distribution of the maximum of a Gaussian process. We then apply this algorithm to implement Thompson sampling.

6.2.1. A GAUSSIAN PROCESS AND ITS MAXIMUM

Consider a continuous function $f(\boldsymbol{x})$ on a compact set X_f. We assume that we have taken n measurements $y_i = f(\boldsymbol{x_i}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}\left(0, \sigma_n^2\right)$ is Gaussian white noise. We merge all the measurement (training) input points $\boldsymbol{x_i}$ into a set X and all the measured output values y_i into a vector $\boldsymbol{y}$.

Now suppose that we want to predict the (noiseless) function values $\boldsymbol{f}_* = f(X_*)$ at a given set of test input points X_*. In this case we can use the GP regression equation from (Rasmussen and Williams, 2006). We use a mean function $m(\boldsymbol{x})$ and a covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$, and we shorten $m(X_a)$ to $\boldsymbol{m_a}$ and $k(X_a, X_b)$ to K_{ab} for any subscripts a and b, as usual. (The subscript for the training set is omitted.) We then have

$$\begin{aligned} \boldsymbol{f}_* &\sim \mathcal{N}\left(\boldsymbol{\mu}_*, \Sigma_{**}\right), \\ \boldsymbol{\mu}_* &= \boldsymbol{m}_* + K_*^T \left(K + \Sigma_n\right)^{-1} \left(\boldsymbol{y} - \boldsymbol{m}\right), \\ \Sigma_{**} &= K_{**} - K_*^T \left(K + \Sigma_n\right)^{-1} K_*. \end{aligned} \tag{6.2}$$

A Gaussian process can be seen as a distribution over functions. That is, we can take samples of $\boldsymbol{f}_*$ and plot those as if they are functions, as is for instance done in Figure 6.1 on page 80. These sample functions generally have their maximums at different locations $\boldsymbol{x}^*$. This implies that $\boldsymbol{x}^*$ is a random variable, and hence has a distribution $p_{\max}(\boldsymbol{x})$. An example of this is shown in Figure 6.2.

The distribution $p_{\max}(\boldsymbol{x})$ cannot be analytically calculated, but it can be approximated through various methods. The most obvious one is through brute force: for a finite number of test input points X_*, we take a large number of samples $\boldsymbol{f}_*$, for each of these samples we find the location of the maximum, and through a histogram we determine the distribution of $\boldsymbol{x}^*$. This method is far from ideal as it is computationally very intensive, even for low-dimensional functions. However, since $f(\boldsymbol{x})$ is assumed continuous, it is guaranteed to converge to the *true maximum distribution* for a sufficient number of points X_* and samples $\boldsymbol{f}_*$.

For larger problems the brute force method is too computationally intensive, motivating the need for a way of approximating the maximum distribution. Methods to do so already exist, like those used by Hennig and Schuler (2012), Hernández-Lobato et al. (2014). However, these methods are all also computationally intensive for larger problems, and so a different way to approximate $p_{\max}(\boldsymbol{x})$ would be beneficial.

6.2.2. APPROXIMATING THE MAXIMUM DISTRIBUTION

We consider a new algorithm, inspired by Sequential Monte Carlo (SMC) samplers, to find the maximum distribution $p_{\max}(\boldsymbol{x})$. Note that the algorithm presented here is *not* an actual SMC sampler, but merely uses techniques also found in SMC samplers. For more background, see Del Moral et al. (2006), Owen (2013).

The main idea is that we have n_p so-called particles at positions $\boldsymbol{x^1}, \ldots, \boldsymbol{x^{n_p}}$. Each of these particles has a corresponding weight $w^1, \ldots, w^{n_p}$. Eventually these particles are supposed to converge to the maximum distribution, at which time we can approximate this distribution through kernel density estimation as

$$p_{\max}(\boldsymbol{x}) \approx \frac{\sum_{i=1}^{n_p} w^i \, k_x(\boldsymbol{x}, \boldsymbol{x^i})}{\sum_{i=1}^{n_p} w^i}, \tag{6.3}$$

with $k_x(\boldsymbol{x}, \boldsymbol{x}')$ some manually chosen kernel. It is common to make use of a squared exponential kernel with a small length scale.

Initially we distribute these particles $\boldsymbol{x^i}$ at random positions across the input space. That is, we sample the particles $\boldsymbol{x^i}$ from the flat distribution $q(\boldsymbol{x}) = c$. Note that, because we have assumed that the input space X_f is compact, the constant c is nonzero.

To learn more about the position of the maximum, we will *challenge* existing particles. To challenge some existing particle $\boldsymbol{x^i}$, we first sample a number n_c of random challenger particles $\boldsymbol{x^i_{c_1}}, \ldots, \boldsymbol{x^i_{c_{nc}}}$ from a proposal distribution $q'(\boldsymbol{x})$. We then set up the joint distribution

$$\begin{bmatrix} f(\boldsymbol{x^i}) \\ f(\boldsymbol{x^i_{c_1}}) \\ \vdots \\ f(\boldsymbol{x^i_{c_{nc}}}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x^i}) \\ \mu(\boldsymbol{x^i_{c_1}}) \\ \vdots \\ \mu(\boldsymbol{x^i_{c_{nc}}}) \end{bmatrix}, \begin{bmatrix} \Sigma(\boldsymbol{x^i}, \boldsymbol{x^i}) & \Sigma(\boldsymbol{x^i}, \boldsymbol{x^i_{c_1}}) & \cdots & \Sigma(\boldsymbol{x^i}, \boldsymbol{x^i_{c_{nc}}}) \\ \Sigma(\boldsymbol{x^i_{c_1}}, \boldsymbol{x^i}) & \Sigma(\boldsymbol{x^i_{c_1}}, \boldsymbol{x^i_{c_1}}) & \cdots & \Sigma(\boldsymbol{x^i_{c_1}}, \boldsymbol{x^i_{c_{nc}}}) \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma(\boldsymbol{x^i_{c_{nc}}}, \boldsymbol{x^i}) & \Sigma(\boldsymbol{x^i_{c_{nc}}}, \boldsymbol{x^i_{c_1}}) & \cdots & \Sigma(\boldsymbol{x^i_{c_{nc}}}, \boldsymbol{x^i_{c_{nc}}}) \end{bmatrix} \right), \tag{6.4}$$

and subsequently generate a sample $\begin{bmatrix} \hat{f}^i & \hat{f}^i_{c_1} & \cdots & \hat{f}^i_{c_{nc}} \end{bmatrix}^T$ from it. Finally, we find the largest element from this vector. If this element equals $\hat{f}^i$, we do nothing. If, however, it equals $\hat{f}^i_{c_j}$, then we have $\hat{f}^i_{c_j} > \hat{f}^i$. In this case there is a challenger that has 'beaten' the current particle and it takes its place. In other words, we replace the particle $\boldsymbol{x^i}$ by $\boldsymbol{x^i_{c_j}}$.

The challenger particle also has a weight associated with. In SMC methods this weight is usually given by

$$w^i_c = \frac{q(\boldsymbol{x^i_c})}{q'(\boldsymbol{x^i_c})}. \tag{6.5}$$

However, to speed up convergence, we use a proposal distribution $q'(\boldsymbol{x})$ based on the ideas of mixture importance sampling and defensive importance sampling. Specifically, we use

$$q'(\boldsymbol{x}) = \alpha p_{\max}(\boldsymbol{x}) + (1-\alpha) q(\boldsymbol{x}). \tag{6.6}$$

Here, α is manually chosen (often roughly $\frac{1}{2}$) and $p_{\max}(\boldsymbol{x})$ is approximated through the mixture proposal distribution (6.3), based on the current particle distribution. To generate a challenger particle $\boldsymbol{x^i_{c_j}}$, we hence randomly (according to the particle weights) select one of the particles $\boldsymbol{x^k}$. Then, in a part α of the cases, we sample $\boldsymbol{x^i_{c_j}}$ from $k_x(\boldsymbol{x}, \boldsymbol{x^k})$, while in the remaining $(1-\alpha)$ part of the cases, we sample $\boldsymbol{x^i_{c_j}}$ from $q(\boldsymbol{x})$. If we sample our challenger particles in this way, it is computationally more efficient to use the weight

$$w^i_{c_j} = \frac{q(\boldsymbol{x^i_{c_j}})}{\alpha k_x(\boldsymbol{x^i_{c_j}}, \boldsymbol{x^k}) + (1-\alpha) q(\boldsymbol{x^i_{c_j}})}. \tag{6.7}$$

Based on this set-up, we will challenge every existing particle once. This is called one *round* of challenges. Afterwards, we apply systematic resampling to make sure all particles have the same weight again. We repeat this until the distribution of particles has mostly converged.

We call the resulting algorithm the *Monte Carlo Maximum Distribution* (MCMD) algorithm. Pseudo-code for it is given in Algorithm 3.

Data: A Gaussian process, user-defined parameters n_p, n_c, α and a kernel $k_x(\boldsymbol{x}, \boldsymbol{x}')$.
Result: An approximate distribution $p_{\max}(\boldsymbol{x})$ of the optimal input $\boldsymbol{x}^*$, given through (6.3).
Initialization:
  for $i \leftarrow 1$ **to** n_p **do**
    Sample $\boldsymbol{x^i}$ from $q(\boldsymbol{x})$. Assign $w^i = 1$.
  end
end
Iteration:
  repeat
    Apply systematic resampling to all particles.
    for $i \leftarrow 1$ **to** n_p **do**
      for $j \leftarrow 1$ **to** n_c **do**
        Select a random particle $\boldsymbol{x^k}$, taking into account weights w^k.
        if *we select a challenger based on* $\boldsymbol{x^k}$ *(probability* α*)* **then**
          Sample a challenger particle $\boldsymbol{x^i_{c_j}}$ from the kernel $k_x(\boldsymbol{x}, \boldsymbol{x^j})$.
        else
          Sample a challenger particle $\boldsymbol{x^i_{c_j}}$ from the flat distribution $q(\boldsymbol{x})$.
        end
      end
      Sample a vector $\begin{bmatrix} \hat{f}^i & \hat{f}^i_{c_1} & \cdots & \hat{f}^i_{c_{n_c}} \end{bmatrix}^T$ based on (6.4) and find its maximum.
      if *the maximum equals* $\hat{f}^i_{c_j} > \hat{f}^i$ **then**
        Replace particle $\boldsymbol{x^i}$ by its challenger $\boldsymbol{x^i_{c_j}}$.
        Set the new weight w^i according to (6.7).
      end
    end
  until *a sufficient number of rounds has passed*;
end

Algorithm 3: The Monte Carlo maximum distribution algorithm. Self-normalized importance sampling, mixture importance sampling and systematic resampling are used.

6.2.3. ANALYSING THE LIMIT DISTRIBUTION OF THE ALGORITHM

The distribution of the particles converges to a *limit distribution*. But does this limit distribution equal the true maximum distribution? We can answer this question for a few special cases.

First consider the case where $n_c \to \infty$. In this case, the algorithm is equivalent to the brute force method of finding the maximum distribution. Assuming that a sufficient number of particles n_p is used, it hence is guaranteed to find the true maximum distribution directly, in only a single round of challenges.

Using $n_c \to \infty$ challenger particles is infeasible, because generating a sample from (6.4) takes $\mathcal{O}(n_c^3)$ time. Instead, we consider a very simplified case with $n_c = 1$ and $\alpha = 0$. Additionally, we consider the discrete case, where there are finitely many possible input points $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$. With finitely many points, we can use the kernel $k_x(\boldsymbol{x}, \boldsymbol{x}') = \delta(\boldsymbol{x} - \boldsymbol{x}')$, with $\delta(\ldots)$ the delta function. In this simplified case, we can analytically calculate the distribution that the algorithm converges to.

Consider the given Gaussian process. Let us denote the probability $\mathbb{P}(f(\boldsymbol{x_i}) > f(\boldsymbol{x_j}))$, based on the data in this Gaussian process, as P_{ij}. Here we have

$$P_{ij} = \mathbb{P}(f(\boldsymbol{x_i}) > f(\boldsymbol{x_j})) = \Phi\left(\frac{\mu(\boldsymbol{x_i}) - \mu(\boldsymbol{x_j})}{\sqrt{\Sigma(\boldsymbol{x_i}, \boldsymbol{x_i}) + \Sigma(\boldsymbol{x_j}, \boldsymbol{x_j}) - 2\Sigma(\boldsymbol{x_i}, \boldsymbol{x_j})}}\right), \tag{6.8}$$

where $\Phi(\ldots)$ is the standard Gaussian cumulative density function. Through this expression we find the matrix P element-wise. Additionally, we write the part of the particles that will eventually be connected to the input $\boldsymbol{x_i}$ as p_i. In this case, the resulting vector $\boldsymbol{p}$ (with elements p_i) can be shown to satisfy

$$\big(P - \text{diag}\,(1_n P)\big)\,\boldsymbol{p} = \boldsymbol{0}, \tag{6.9}$$

where 1_n is an $n \times n$ matrix filled with ones, and diag$(\ldots)$ is the function which sets all non-diagonal elements to zero. If we also use the fact that the sum of all probabilities $\mathbf{1}^T \boldsymbol{p}$ equals 1, we can find $\boldsymbol{p}$ for this discrete problem.

If the algorithm would converge to the true maximum distribution in this simplified case (with $n_c = 1$) then we must have $p_i = p_{\max}(\boldsymbol{x_i})$. In other words, the vector $\boldsymbol{p}$ would then describe the maximum distribution. However, since we can calculate the values p_i analytically, while it is known to be impossible to find $p_{\max}(\boldsymbol{x_i})$ like this, we already know that this is not the case. p_i must be different from $p_{\max}(\boldsymbol{x_i})$, and the algorithm hence does *not* converge to the maximum distribution when $n_c = 1$. However, the example from Figure 6.2 on page 81 does show that the algorithm gives a fair approximation. The limit distribution of the algorithm is generally less peaked than the true maximum distribution, which means it contains less information about where the maximum is (lower relative entropy) but overall its predictions are accurate. Furthermore, the difference will decrease when the variance present within the Gaussian process decreases, or when we raise n_c.

To summarize, n_c can be seen as a trade-off between accuracy and computational efficiency. A large value of n_c will give a better result, but requires more computations. But we will see that even $n_c = 1$ gives decent performance.

6.2.4. APPLYING THE MCMD ALGORITHM FOR THOMPSON SAMPLING

We can now use the MCMD algorithm to apply Thompson sampling in a Gaussian process optimization setting. To do so, we sample an input point $\boldsymbol{x}$ from the approximated maximum distribution $p_{\max}(\boldsymbol{x})$ whenever we need to perform a new measurement.

The downside of this method is that samples are not drawn from the true maximum distribution, but only from an approximation of it. However, the upside is that this approximation can be obtained by making simple comparisons between function values. No large matrix equations need to be solved or nonlinear function optimizations need to be performed, providing a significant computational advantage over other methods that approximate the maximum distribution.

6.3. EXPERIMENT RESULTS

Here we show the results of the presented algorithms. First we study how the MCMD algorithm works for a fixed one-dimensional Gaussian process. Then we apply it through Thompson sampling to the same problem, expand to a two-dimensional problem and finally apply it to a real-life application.

6.3.1. EXECUTION OF THE MCMD ALGORITHM

Consider the function

$$f(x) = \cos(3x) - \frac{1}{9}x^2 + \frac{1}{6}x. \tag{6.10}$$

From this function, we take 20 noisy measurements, at random locations in the interval $[-3,3]$, with $\sigma_n = 0.3$ as standard deviation of the white noise. We then apply GP regression with a squared exponential covariance function with predetermined hyperparameters. The subsequent GP approximating these measurements is shown in Figure 6.1.

We can apply the MCMD algorithm to approximate the maximum distribution $p_{\max}(\boldsymbol{x})$ of this Gaussian process. This approximation, during successive challenge rounds of the algorithm with $\alpha = \frac{1}{2}$ and $n_c = 1$, is shown in Figure 6.2. (We always use $n_c = 1$ in these experiments, because it allows us to analytically calculate the limit distribution. For actual applications larger values are recommended.) In this Figure we see that the algorithm has mostly converged to the limit distribution after $n_r = 10$ rounds of challenges, but this limit distribution has a slightly higher entropy compared to the true maximum distribution.

6.3.2. APPLICATION TO AN OPTIMIZATION PROBLEM

We will now apply the newly developed method for Thompson sampling to Bayesian optimization. We will compare it with the UCB, the PI and the EI acquisition functions. After some tuning their parameters were set to $\kappa = 2$ and $\xi = 0.1$, which gave the best results that could be obtained for these algorithms. To optimize these acquisition functions, we use a multi-start optimization method, because otherwise we occasionally wind up with a local optimum of the acquisition function, resulting in a detrimental performance. We do not compare our results with entropy search or portfolio methods, because they are designed for the error minimization set-up.

The first problem we apply these methods to is the optimization of the function $f(x)$ from (6.10). We use $n = 50$ input points $x_1, \ldots, x_n$ and look at the obtained regret. To keep the memory and runtime requirements of the GP regression algorithm somewhat

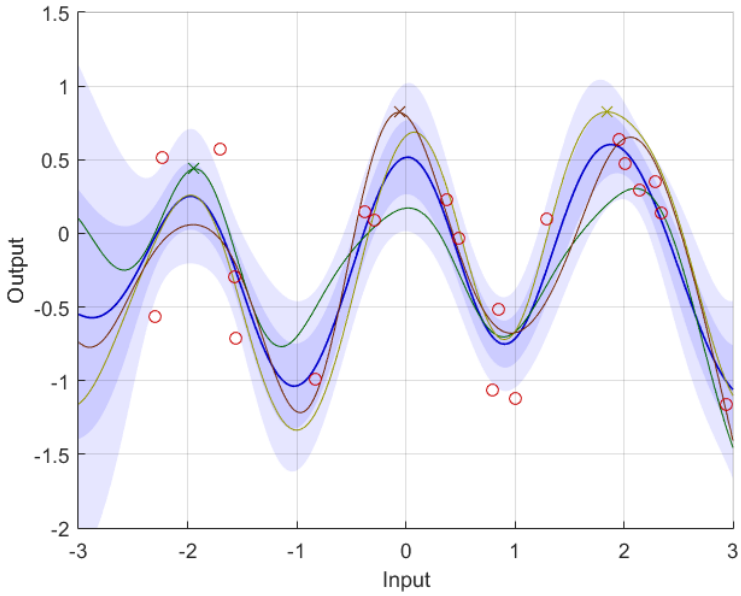

Figure 6.1: An example Gaussian process. The circles denote the measurements from which the GP was generated. The thick line denotes the (posterior) mean of the GP and the grey area represents the 95% certainty region. The three thinner lines are samples from the GP distribution. It is worthwhile to note that they have their maximum values (the crosses) at very different positions.

limited, given the large number of experiments that will be run, we will apply the FITC approximation described by Candela and Rasmussen (2005), implemented in an online fashion according to Bijl et al. (2015). As inducing input points, we use the chosen input points, but only when they are not within a distance d_u (decreasing from 0.3 to 0.02 during the execution of the algorithm) of any already existing inducing input point. For simplicity the hyperparameters are assumed known and are hence fixed to reasonable values. Naturally, it is also possible to learn hyperparameters on-the-go as well, using the techniques described by (Rasmussen and Williams, 2006).

The result is shown in Figure 6.3. In this particular case, it seems that Thompson sampling and the PI AF applied mostly exploitation: they have a better performance on the short term. On the other hand, the UCB and EI acquisition functions apply more exploration: the cost of quickly exploring is higher, but because the optimum is found sooner, it can also be exploited sooner.

It should also be noted that all algorithms occasionally wind up in the wrong optimum (near $x = 2$) every now and then. This can be seen from the fact that the regret graph does not level out. For this particular problem, the UCB AF seems to be the best at avoiding the local optimums, but it still falls for them every now and then. As noted earlier, only Thompson sampling has the guarantee to escape local optimums given infinitely many measurements.

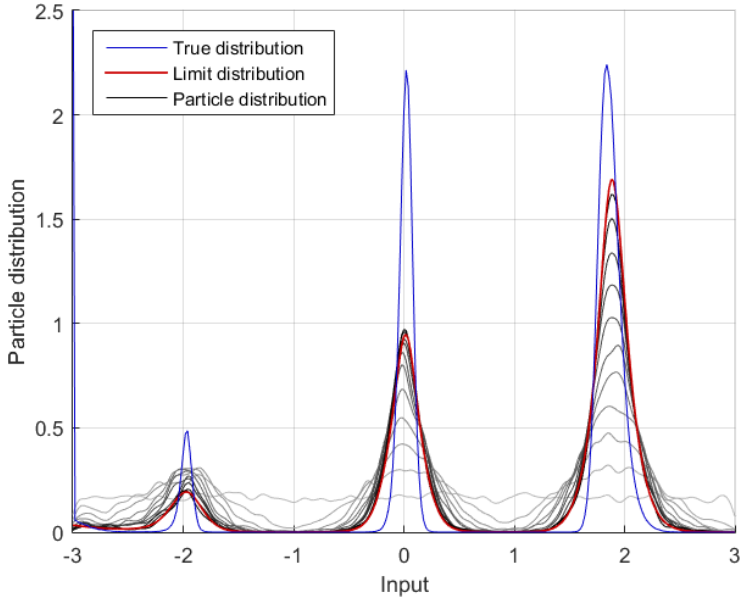

Figure 6.2: The maximum distribution for the Gaussian process shown in Figure 6.1. The black/grey lines represent the approximate maximum distribution after $1, 2, \ldots, 10$ rounds of challenges for $n_p = 10\,000$ particles. The thick line is the limit distribution of the particles as derived in Section 6.2.3. The peaked line is the true maximum distribution, found through brute force methods.

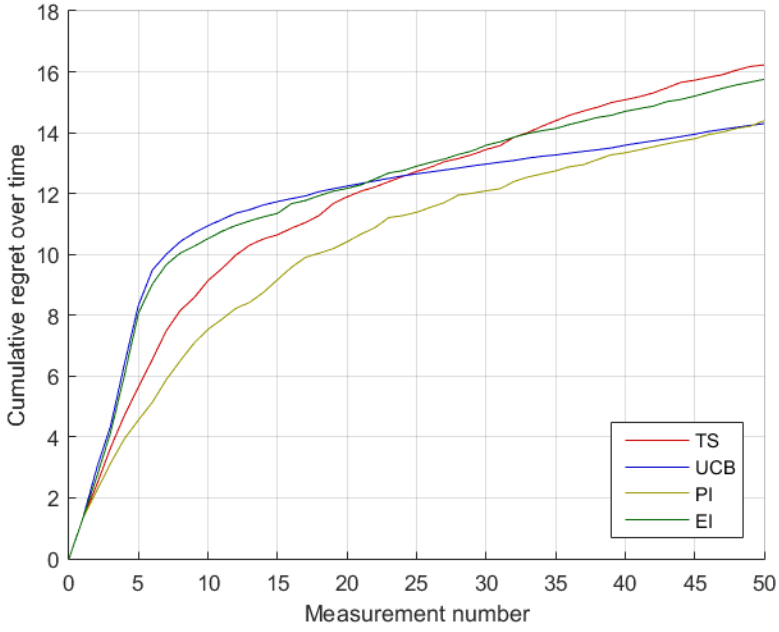

Figure 6.3: The cumulative regret (6.1) of the various Bayesian optimization algorithms for the function in (6.10). Results shown are the mean performance of fifty complete runs of each algorithm.

6.3.3. EXTENSION TO A TWO-DIMENSIONAL PROBLEM

Next, we apply the optimization methods to a two-dimensional problem. We will minimize the well-known Branin function from (among others) Dixon and Szegö (1978). Or equivalently, we maximize the negative Branin function,

$$f(x_1, x_2) = -\left(x_2 - \frac{51x_1^2}{40\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 - 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) - 10, \tag{6.11}$$

where $x_1 \in [-5, 10]$ and $x_2 \in [0, 15]$. This function is shown in Figure 6.4. We can find analytically that the optimums occur at $\left(-\pi, \frac{491}{40}\right)$, $\left(\pi, \frac{91}{40}\right)$ and $\left(3\pi, \frac{99}{40}\right)$, all with value $-\frac{5}{4\pi}$.

The performance of the various optimization methods, averaged out over fifty full runs, is shown in Figure 6.5. Here we see that Thompson sampling now performs significantly better at keeping the regret small compared to the UCB ($\kappa = 2$), the PI and the EI ($\xi = 2$) acquisition functions. We can find the reason behind this, if we look at which trial points the various algorithms select. When we do (not shown here), we see that all acquisition functions often try out points at the border of the input space, while Thompson sampling does not. In particular, the acquisition functions (nearly) always try out all four corners of the input space, including the very detrimental point $(-5, 0)$. It is this habit which makes these acquisition functions perform worse on this specific problem.

Other than this, it is also interesting to note that in all examined runs, all optimization methods find either two or three of the optimums. So while multiple optimums are always found, it does regularly happen that one of the three optimums is not found. All of the methods have shown to be susceptible to this. In addition, the three acquisition functions have a slightly lower average recommendation error than Thompson sampling, but since all optimization methods find various optimums, the difference is negligible. On the flip side, an advantage of using the MCMD algorithm is that it can provide us with the posterior distribution of the maximum, given all the measurement data. An example of this is shown in Figure 6.6.

6.3.4. OPTIMIZING A WIND TURBINE CONTROLLER

It is time to test our algorithm on an application: data-based controller tuning for load mitigation within a wind turbine. More specifically, we use a linearized version of the so-called TURBU model, described by van Engelen and Braam (2004). TURBU is a fully integrated wind turbine design and analysis tool. It deals with aerodynamics, hydrodynamics, structural dynamics and control of modern three bladed wind turbines, and as such gives very similar results as an actual real-life wind turbine.

We will consider the case where trailing edge flaps have been added to the turbine blades. These flaps should then be used to reduce the vibration loads within the blades. To do so, the Root Bending Moment (RBM) of the blades is used as input to the control system.

To determine the effectiveness of the controller, we look at two quantities. The first is the DEL (see Section 1.2.4). The second quantity to optimize is the mean rate of change of the input signal. The reason here is that the lifetime of bearings is often expressed in the number of revolutions, or equivalently in the angular distance traveled, and dividing this distance traveled by the time passed will result in the mean rate of change of the flap

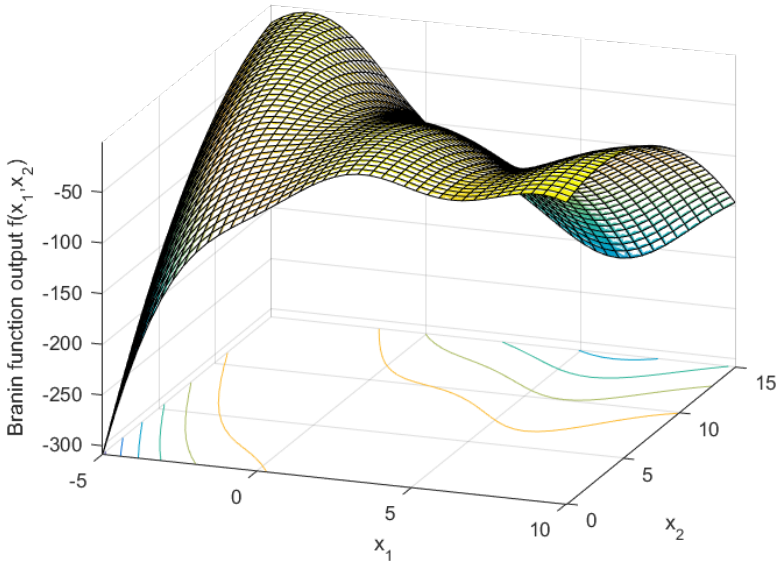

Figure 6.4: The (negative) Branin function, defined by (6.11). Its three optimums can be clearly noted.

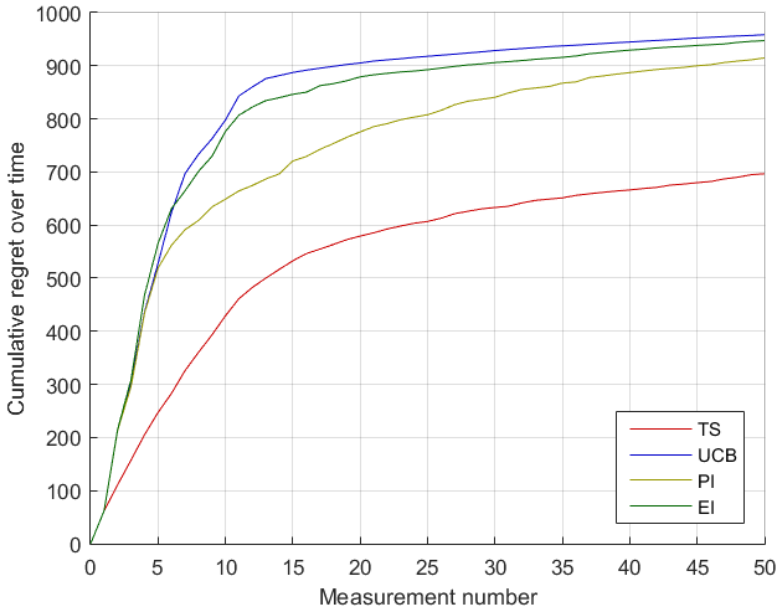

Figure 6.5: The cumulative regret (6.1) of the various Bayesian optimization algorithms for the Branin function (6.11). Results shown are the mean performance of fifty complete runs of each algorithm.

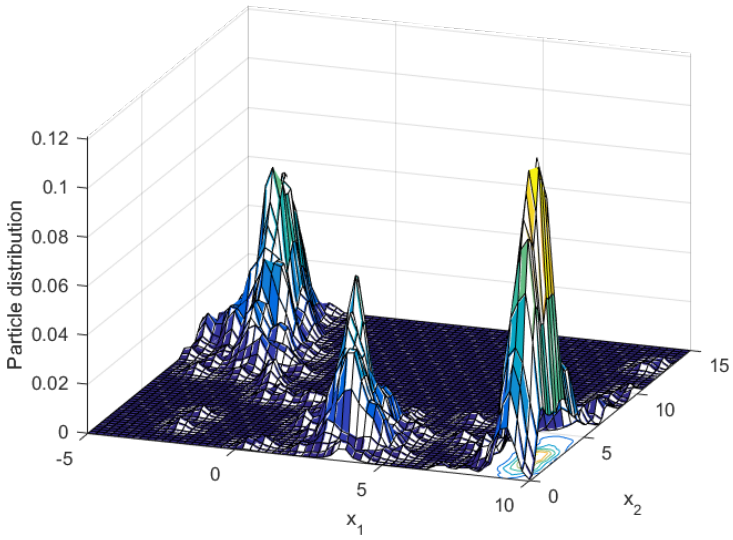

Figure 6.6: The probability distribution of the maximum of the GP approximating the Branin function, after generating measurements according to Thompson sampling. The three optimums have been identified, some stray particles still reside in the lesser explored regions, and no particles remain in the part of the input space that has been explored but was found suboptimal.

angle. The eventual performance score for a controller will now be a linearly weighted sum of these two parameters, where a lower score is evidently better.

As controller, we apply a proportional controller. That is, we take the RBM in the fixed reference frame (so after applying a Coleman transformation; see van Solingen and van Wingerden (2015)) and feed the resulting signal, multiplied by a constant gain, to the blade flaps. Since the wind turbine has three blades, there are three gains that we can apply. The first of these, the collective flap mode, interferes with the power control of the turbine. We will hence ignore this mode and only tune the gains of the tilt and yaw modes. Very low gains (in the order of 10^{-8}) will result in an inactive controller which does not reduce the RBM, while very high gains (in the order of 10^{-5}) will react to every small bit of turbulence, resulting in an overly aggressive controller with a highly varying input signal. Both are suboptimal, and the optimal controller will have gains somewhere between these two extreme values.

To learn more about the actual score function, we can apply a brute force method – just applying 500 random controller settings – and apply GP regression. This gives us Figure 6.7. Naturally, this is not possible in real life as it would cause unnecessary damage to the wind turbine. It does tell us, however, that the score function is mostly convex and that there does not seem to exist any local optimums apart from the global optimum.

The results from the Bayesian optimization experiments, which are remarkably similar to earlier experiments, are shown in Figure 6.8. (We used $\kappa = 1$ and $\xi = 0.005$ here.) They once more show that Thompson sampling has a competitive performance at keeping the regret limited.

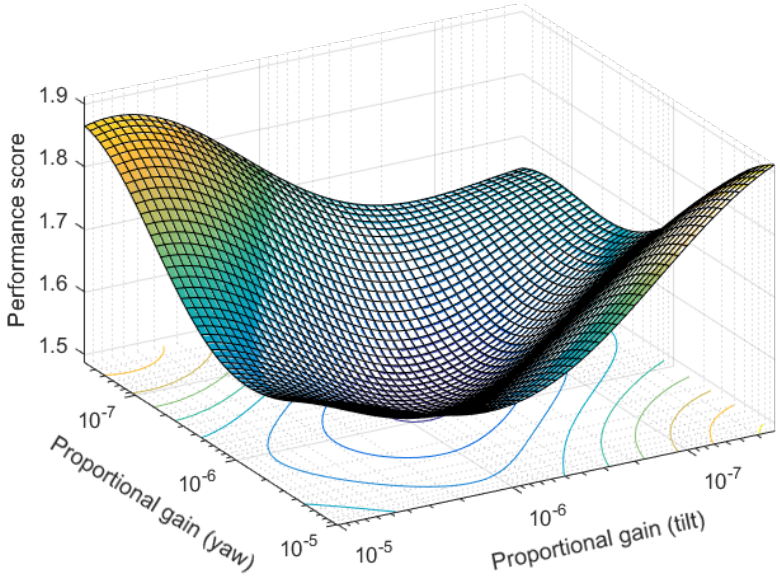

Figure 6.7: An approximation of the wind turbine controller score with respect to the controller gain. This approximation was made by taking 500 random points and applying a GP regression algorithm to the resulting measurements.

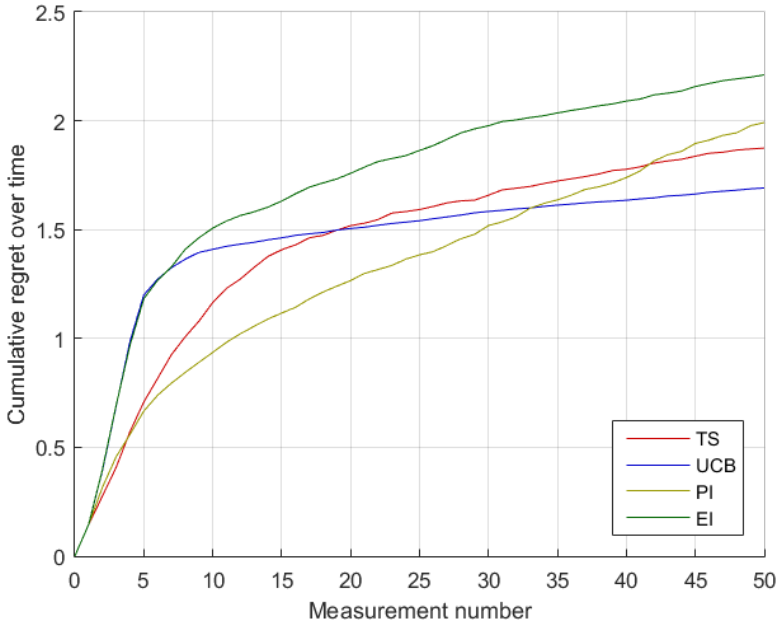

Figure 6.8: The cumulative regret (6.1) of the various Bayesian optimization algorithms for the wind turbine controller. Results shown are the mean performance of fifty complete runs of each algorithm.

6.3.5. APPLYING THE METHODS TO A WIND TUNNEL TEST

To see whether Gaussian process optimization works in real life as well, we can check out its performance in a wind tunnel experiment. For this, a two-bladed wind turbine with controllable flaps on the wind turbine blades was available, depicted in Figure 6.9. A two-bladed wind turbine works slightly different from a three-bladed turbine, but the differences are not significant to how GPO is applied.

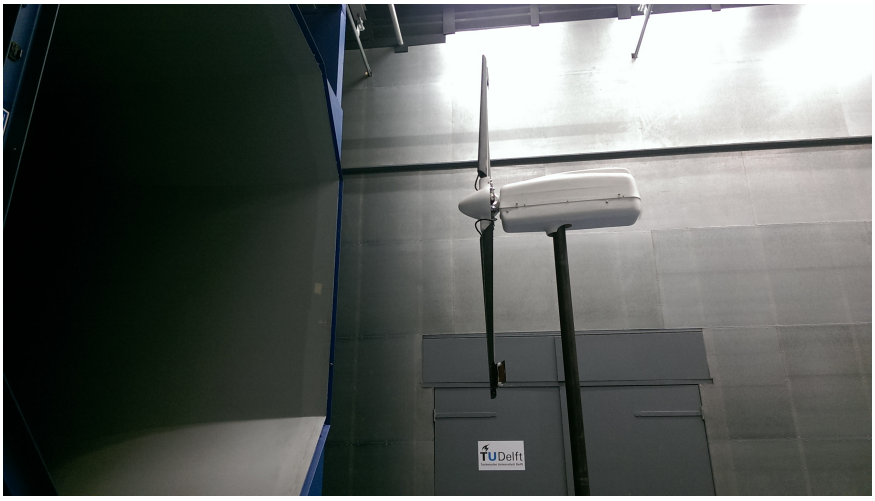

Figure 6.9: The wind turbine used in the experiment, placed in the open-jet wind tunnel of the Delft University of Technology. Note the trailing edge flaps near the turbine blade tips.

One thing that was very different was the absence of turbulence. The open-jet wind tunnel that was used has been set up with the specific goal of minimizing turbulence. As a result, setting up a controller to reduce the effects of turbulence, like in the previous experiment, did not work so well. The optimal strategy was quite close to 'Do nothing, because there is no turbulence.'

Instead, the focus was on optimizing the cyclic loads due to other factors, like tower shadow. For this, we gave a cyclic (sinusoidal) input signal to the wind turbine blade flaps, expressed in the azimuth ψ of the respective blade. That is, we used

$$u(t) = A\sin\left(\psi + \phi\right), \tag{6.12}$$

where A is the amplitude and ϕ is the phase shift of the input signal. Equivalently, we can also rewrite the above to the form

$$u(t) = \theta_1 \sin\left(\psi\right) + \theta_2 \cos\left(\psi\right). \tag{6.13}$$

The challenge now is to tune θ_1 and θ_2 such that the damage equivalent load is minimized. In this experiment, we did not take the loads on the bearing into account, because restricting the control signal to a sinusoidal form already removes the risk of winding up with an overly aggressive controller.

Because of limited wind tunnel time, only Thompson sampling has been applied for $n = 15$ measurements. These measurements, together with the resulting approximation of the damage equivalent load, is shown in Figure 6.10 (left). The corresponding belief about where the optimal parameters are is shown in Figure 6.10 (right). These parameters were close to the parameters found by colleagues Navalkar et al. (2016), who set up an identical control law and tuned it using iterative feedback tuning.

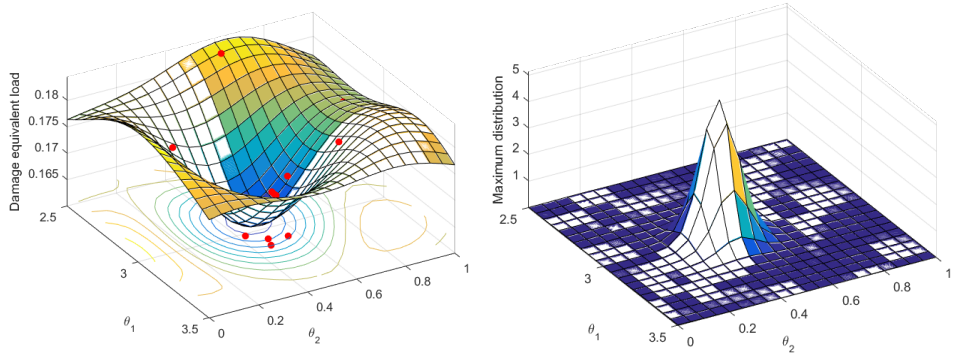

Figure 6.10: The results of the wind turbine experiment, with the approximated damage equivalent load mean (left) and the belief of where the optimal controller settings would be (right). A total of $n = 15$ measurements were performed using Thompson sampling before 'something' changed and the entire wind turbine behaved differently. During these experiments, the Thompson sampling algorithm first tried a few not-so-optimal controller settings, but quickly managed to find a better operating region and stay within it. During these experiments, it got more and more certain of where the optimal operating point would be.

6.4. CONCLUSIONS AND RECOMMENDATIONS

We have introduced the MCMD algorithm, which uses particles to approximate the distribution of the maximum of a Gaussian process. This particle approximation can then be used to set up a Bayesian optimization method using Thompson sampling. Such optimization methods are suitable for tuning the parameters of systems with large amounts of uncertainty in an online data-based way. As an example, we have tuned the controller parameters of a wind turbine (both in a simulation and in a wind tunnel test) to reduce the fatigue load using performance data that was obtained during the operation of the wind turbine.

The main advantage of Thompson sampling with the MCMD algorithm is that it does not require the optimization of a nonlinear function to select the next trial point. In addition, it has shown to have a competitive performance at keeping the cumulative regret limited. However, we cannot conclude that Thompson sampling, or any other optimization method, works better than its competitors. Which method works well depends on a variety of factors, like how much the method has been fine-tuned to the specific function that is being optimized, as well as which function is being optimized in the first place. Also the number of trial points used matters, where a lower number gives the advantage to exploitation-based methods, while a higher number benefits the more exploration-based methods. It is for this very reason that any claim that a Bayesian

optimization works better than its competitors may be accepted only after very careful scrutiny.

A recommendation for future work for this algorithm would be to study convergence rates. At the moment of publishing Bijl et al. (2017a), no results were known about convergence rates of Thompson sampling applied to problems with a continuous input space. However, with the work of Basu and Ghosh (2017) this is starting up. There is bound to be more to discover.

7

CONCLUSIONS AND RECOMMENDATIONS

In this final chapter we look back to the questions that we asked in the Introduction and evaluate the answers to them. We also look at possible directions for future work.

7.1. CONCLUSIONS

There were three types of questions that we posed. We look into them one by one.

7.1.1. LINEAR-QUADRATIC-GAUSSIAN SYSTEMS

In Chapters 2 and 3 we looked at LQG systems. The first question that we asked was the following.

Question 1a: *Within LQG control, how can we reduce the probability that the cost exceeds a given threshold?*

Chapter 2 provided us with analytical methods to calculate both the mean and the variance of the cost J of LQG systems for a variety of cases: finite time versus infinite time cost functions and discounted versus non-discounted cost functions. Using these expressions, it is possible to reduce the number of times that the cost exceeds a given threshold, for instance by minimizing the cost variance.

Question 1b: *In LQG control, how can we prescribe a degree of stability while also optimally applying a state estimator?*

This question was answered in Chapter 3. It is possible to prescribe a degree of stability by adding a positive discount exponent to the cost function. When doing so, we can find an optimal control law minimizing the expected discounted cost. When the state is not known, it can be estimated through a state estimator, where it is possible to tune the gains to again minimize the expected cost.

7.1.2. ONLINE SYSTEM IDENTIFICATION THROUGH GAUSSIAN PROCESSES

The second main question that we asked ourselves concerned system identification.

> **Question 2:** *How can Gaussian process regression be used for online black-box modeling of a nonlinear multi-variable system like a wind turbine?*

The main challenge of answering this question was that we needed to enable Gaussian process regression to deal with online updates, sparse methods and noisy input points simultaneously. Though promising methods like latent variable methods already exist, these methods have a large number of parameters to optimize, making the computation of the derivatives rather slow.

As a result, an extension of the NIGP method was developed, called the SONIG algorithm. Through various extensions, it was capable of identifying the dynamics of non-linear systems, outperforming existing state-of-the-art non-linear system identification algorithms. It has also shown to be capable of identifying the dynamics of a pitch-plunge system.

7.1.3. FIXED-STRUCTURE CONTROLLER TUNING

The final question we posed in the Introduction concerned the tuning of fixed-structure controllers. This was split up into two subquestions.

> **Question 3a:** *How can Gaussian process regression be used to approximate and optimize the cost function of a system with a fixed-structure control law?*

In Chapter 5 we found a method to do so. This method was also capable of incorporating knowledge of the state, which turned out to be useful for short experiment durations, in which the initial state has a significant influence on the cost occurred. However, taking into account state data did require a large number of experiments. When the number of experiments, and specifically the damage (cost) obtained during these experiments, needs to be limited, we run into our final subquestion.

> **Question 3b:** *How can the gains of a fixed-structure control law be tuned online, in such a way as to minimize the damage sustained by the wind turbine?*

This question was tackled in Chapter 6, where we looked into methods of efficiently choosing which controller gains to try out next. To do so, a novel way of determining the maximum of a Gaussian process was set up: the Monte Carlo maximum distribution algorithm. This SMC-inspired method approximates the maximum distribution of a Gaussian process. Using it, it is now possible to apply Thompson sampling to Bayesion optimization problems with a continuous input space. The resulting optimization method has shown to have a competitive performance at keeping the cumulative regret (i.e., the wind turbine fatigue damage) limited.

Overall, this thesis has provided a variety of methods through which uncertainty can be taken into account while analyzing and controlling wind turbines.

7.2. RECOMMENDATIONS

Research is never finished. There are still several new avenues that can be explored.

For LQG systems, these avenues mainly lie in the application. There is much ongoing research about applying LQG controllers to wind turbines. The techniques developed in Chapters 2 and 3 can potentially improve these controllers. How that works in detail will strongly depend on the way that the LQG controllers are actually applied, so this is left for the specific researchers and companies involved.

For Gaussian process regression, though a significant amount of process has been made in the last decade, there is still a huge challenge in efficiently incorporating large data sets in making predictions. Methods using inducing input points are promising, but for high-dimensional problems the number of inducing input points often also grows exponentially with the dimension, resulting in longer run-times. If, while incorporating a new measurement, we only update the inducing input points with a large covariance with respect to that measurement, then significant gains in the run-time will likely be obtained. How to do so in detail is a question for future research.

The most promising technique developed in this thesis is Thompson sampling applied to fixed-structure controller optimization. This technique has been shown to work on optimizing the controller parameters of a scaled wind turbine in a wind tunnel. There are many ways through which this technique can be improved further.

- It may happen during an experiment that we immediately realize that the controller parameters used will result in a high cost. In that case, stopping the experiment prematurely would be wise. How can the results of this experiment then still be incorporated? Probably the obtained DEL during the experiment can still be used, if the relatively short experiment duration is taken into account by increasing the measurement noise present in the DEL.
- It may also be necessary to apply 'safe exploring'. That is: only try controller gains for which you are nearly certain that they will not result in instability. To do so, we could add the restriction to only try controllers for which the probability $p(\text{DEL} > \text{threshold})$, that the DEL falls above a given threshold, is sufficiently small. This is similar to the ideas proposed by Sui et al. (2015), Berkenkamp et al. (2016). Alternatively, we could initialize the GP approximating the DEL with a very high mean function. All these ideas are likely to slow down the exploration of the space of possible controller gains, since 'novel' gains far from anything that has been tried before cannot be experimented with anymore, but this is the cost required for reducing the chance at obtaining instability.
- Finally, it is known that the dynamics of a wind turbine can change during the aging of the turbine. Currently the algorithm assumes that the cost function is a function that only depends on the controller gains and does not vary over time. It is possible to take a slowly time-varying function into account, for instance by adding extra measurement noise variance to older measurements. Or, when inducing input points are used, by adding a small amount of extra variance to the inducing function values as time passes. How much this should be done is a question for further research. Is it something that needs to be specified in advance, or can the amount of shifting of the cost function also be determined automatically from measurement data?

Looking at these ideas for future research, it is clear that GP regression and Bayesian optimization have the potential for an interesting future in the field of wind energy.

A

SUPPORTING THEOREMS

This appendix contains mathematical theorems necessary to derive some of the main results in this thesis.

A.1. EVOLUTION OF THE STATE

Consider a linear system described by

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + \boldsymbol{v}(t), \tag{A.1}$$

with $\boldsymbol{x}(t)$ the state, A the system matrix, and $\boldsymbol{v}(t)$ zero-mean Gaussian white noise with intensity V. (For details, see Section 2.2.) Solving (A.1) for $\boldsymbol{x}(t)$ results in

$$\boldsymbol{x}(t) = e^{At}\boldsymbol{x}_0 + \int_0^t e^{A(t-s)}\boldsymbol{v}(s)\,ds. \tag{A.2}$$

We use this to derive statistical properties for $\boldsymbol{x}(t)$. We assume here that the initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ has a Gaussian distribution satisfying $\boldsymbol{\mu}_0 \equiv \mathbb{E}[\boldsymbol{x}_0]$ and $\Sigma_0 \equiv \mathbb{E}[\boldsymbol{x}_0\boldsymbol{x}_0^T]$, so that its variance equals $\Sigma_0 - \boldsymbol{\mu}_0\boldsymbol{\mu}_0^T$. The statistical properties we will find are well-known (see for instance Bosgra et al. (2008)), but they are included to give a good overview of existing theory.

Theorem A.1. *When $\boldsymbol{x}(t)$ satisfies system (A.1), with the corresponding assumptions on $\boldsymbol{x}(0)$ and $\boldsymbol{v}$, then $\boldsymbol{x}(t)$ is a Gaussian random variable satisfying*

$$\boldsymbol{\mu}(t) \equiv \mathbb{E}[\boldsymbol{x}(t)] = e^{At}\boldsymbol{\mu}_0, \tag{A.3}$$

$$\Sigma(t) \equiv \mathbb{E}[\boldsymbol{x}(t)\boldsymbol{x}^T(t)] = e^{At}(\Sigma_0 - X^V)e^{A^T t} + X^V. \tag{A.4}$$

Proof. Because $\boldsymbol{x}(t)$ is the sum of Gaussian variables, it will have a Gaussian distribution at all times t. From (A.2), its mean equals

$$\boldsymbol{\mu}(t) \equiv \mathbb{E}[\boldsymbol{x}(t)] = e^{At}\mathbb{E}[\boldsymbol{x}_0] = e^{At}\boldsymbol{\mu}_0. \tag{A.5}$$

The expected squared value is found similarly through

$$\begin{aligned}\Sigma(t) &= e^{At}\mathbb{E}[\boldsymbol{x}_0\boldsymbol{x}_0^T]e^{A^Tt} + \int_0^t\int_0^t e^{A(t-s_1)}\mathbb{E}[\boldsymbol{v}(s_1)\boldsymbol{v}^T(s_2)]e^{A^T(t-s_2)}\,ds_1\,ds_2 \\ &= e^{At}\Sigma_0 e^{A^Tt} + \int_0^t e^{A(t-s)}Ve^{A^T(t-s)}\,ds. \end{aligned} \tag{A.6}$$

(The reduction of $\mathbb{E}[\boldsymbol{v}(s_1)\boldsymbol{v}^T(s_2)]$ to $V\delta(s_1-s_2)$ is formally an application of the Itô isometry, as explained in Øksendal (1985).) Next, by substituting s by $t-\tau$, we find that

$$\begin{aligned}\Sigma(t) &= e^{At}\Sigma_0 e^{A^Tt} + \int_0^t e^{A\tau}Ve^{A^T\tau}\,ds \\ &= e^{At}\Sigma_0 e^{A^Tt} + X^V(t) = e^{At}(\Sigma_0 - X^V)e^{A^Tt} + X^V,\end{aligned} \tag{A.7}$$

where in the end we have also applied Theorem A.7. □

Theorem A.2. *The expected squared value $\Sigma(t)$ satisfies*

$$\dot{\Sigma}(t) = A\Sigma(t) + \Sigma(t)A^T + V. \tag{A.8}$$

Proof. The derivative of (A.4) equals

$$\begin{aligned}\dot{\Sigma}(t) &= A\left(e^{At}(\Sigma_0 - X^V)e^{A^Tt}\right) + \left(e^{At}(\Sigma_0 - X^V)e^{A^Tt}\right)A^T \\ &= A\left(\Sigma(t) - X^V\right) + \left(\Sigma(t) - X^V\right)A^T \\ &= A\Sigma(t) + \Sigma(t)A^T - \left(AX^V + X^VA^T\right).\end{aligned} \tag{A.9}$$

Applying $AX^V + X^VA^T + V = 0$ completes the proof. □

Theorem A.3. *For $t_1 < t_2$ we have*

$$\Sigma(t_1,t_2) = e^{At_1}(\Sigma_0 - X^V)e^{A^Tt_2} + X^Ve^{A^T(t_2-t_1)}. \tag{A.10}$$

Furthermore, $\Sigma(t_1,t_2) = \Sigma(t_2,t_1)^T$ and $\Sigma(t,t) = \Sigma(t)$.

Proof. The proof is identical to that of Theorem A.1. □

A.2. PROPERTIES OF LYAPUNOV EQUATION SOLUTIONS

This section is about solutions to Lyapunov equations. It uses the notation described in Section 2.3.1.

Theorem A.4. *There is a unique solution for X^Q, and identically for $\bar{X}^Q$, if and only if the matrix A is Sylvester.*

Proof. In literature it is known (see Bartels and Stewart (1972)) that the Sylvester Equation $AX + XB = Q$ has a unique solution if and only if A and $-B$ do not have a common eigenvalue. Substituting $B = A^T$ directly proves the theorem. □

Theorem A.5. *Assume that A is Sylvester. In this case X^Q is symmetric if and only if Q is symmetric.*

Proof. If we take the Lyapunov equation $AX^Q + X^Q A^T + Q = 0$ and subtract its transpose, we find that

$$A\left(X^Q - (X^Q)^T\right) + \left(X^Q - (X^Q)^T\right)A^T + (Q - Q^T) = 0. \tag{A.11}$$

This equation has a unique solution (Theorem A.4) directly implying that $Q = Q^T$ if and only if $X^Q = (X^Q)^T$. □

Theorem A.6. *Assume that A is stable. Then A is Sylvester and the Lyapunov equation $AX^Q + X^Q A^T + Q = 0$ has a unique solution X^Q which equals*

$$X^Q = \int_0^\infty e^{At} Q e^{A^T t}\, dt. \tag{A.12}$$

Proof. The assumption that A is stable directly implies that A is Sylvester and hence (Theorem A.4) that X^Q exists and is unique. Now we only need to prove (A.12). Because A is stable, we know that $\lim_{t\to\infty} e^{At} = 0$. We can hence write Q as

$$\begin{aligned} Q &= -\left[e^{At} Q e^{A^T t}\right]_0^\infty \\ &= -\int_0^\infty \frac{d}{dt}\left(e^{At} Q e^{A^T t}\right) dt \\ &= -\int_0^\infty \left(Ae^{At} Q e^{A^T t} + e^{At} Q e^{A^T t} A^T\right) dt \\ &= -A\left(\int_0^\infty e^{At} Q e^{A^T t}\, dt\right) - \left(\int_0^\infty e^{At} Q e^{A^T t}\, dt\right) A^T. \end{aligned} \tag{A.13}$$

The equation above is a Lyapunov equation with the quantity between brackets as its unique solution X^Q. □

Theorem A.7. *When A is Sylvester, $X^Q(t_1, t_2)$ can either be found by solving the Lyapunov equation*

$$AX^Q(t_1,t_2) + X^Q(t_1,t_2)A^T + e^{At_1} Q e^{A^T t_1} - e^{At_2} Q e^{A^T t_2} = 0 \tag{A.14}$$

or by first finding X^Q and then using

$$X^Q(t_1,t_2) = e^{At_1} X^Q e^{A^T t_1} - e^{At_2} X^Q e^{A^T t_2}. \tag{A.15}$$

Proof. We first prove (A.14) through

$$\begin{aligned} e^{At_1} Q e^{A^T t_1} - e^{At_2} Q e^{A^T t_2} &= -\left[e^{At} Q e^{A^T t}\right]_{t_1}^{t_2} \\ &= -\int_{t_1}^{t_2} \frac{d}{dt}\left(e^{At} Q e^{A^T t}\right) dt \\ &= -A\left(\int_{t_1}^{t_2} e^{At} Q e^{A^T t}\, dt\right) - \left(\int_{t_1}^{t_2} e^{At} Q e^{A^T t}\, dt\right) A^T \\ &= -AX^Q(t_1,t_2) - X^Q(t_1,t_2)A^T. \end{aligned} \tag{A.16}$$

To prove (A.15) too, we will use $Q = -AX^Q - X^Q A^T$ and the matrix property $e^{At}A = Ae^{At}$ to find that

$$e^{At_1}Qe^{A^Tt_1} - e^{At_2}Qe^{A^Tt_2} = -A\Big(e^{At_1}X^Qe^{A^Tt_1} - e^{At_2}X^Qe^{A^Tt_2}\Big) - \Big(e^{At_1}X^Qe^{A^Tt_1} - e^{At_2}X^Qe^{A^Tt_2}\Big)A^T. \tag{A.17}$$

The above expression actually equals (A.14), except that the part between brackets is replaced by $X^Q(t_1,t_2)$. Because A is Sylvester, the expression has a unique solution $X^Q(t_1,t_2)$, which must equal the part between brackets. □

Theorem A.8. *Assume that A is Sylvester and that $AC = CA$. For any Q and V we then have*

$$X^{CQ+V} = CX^Q + X^V. \tag{A.18}$$

Proof. Per definition, $AX^Q + X^QA^T + Q = 0$ and $AX^V + X^VA^T + V = 0$. Left-multiplying the first expression by C and adding it to the second gives us

$$A\big(CX^Q + X^V\big) + \big(CX^Q + X^V\big)A^T + (CQ + V) = 0. \tag{A.19}$$

This is a Lyapunov equation with X^{CQ+V} as its solution. □

Theorem A.9. *Assume that A is Sylvester. For matrices F and G satisfying $AF = FA$ and $A^TG = GA^T$, and for any Q and V, we have*

$$tr\big(QFX^VG\big) = tr\big(\bar{X}^QFVG\big). \tag{A.20}$$

Proof. This is directly proven by

$$\begin{aligned}\mathrm{tr}\big(QFX^VG\big) &= \mathrm{tr}\big((-A^T\bar{X}^Q - \bar{X}^QA)FX^VG\big)\\ &= \mathrm{tr}\big((-A^T\bar{X}^QFX^VG - \bar{X}^QAFX^VG)\big)\\ &= \mathrm{tr}\big((-G\bar{X}^QFX^VA^T - G\bar{X}^QFAX^V)\big)\\ &= \mathrm{tr}\big(G\bar{X}^QF(-X^VA^T - AX^V)\big)\\ &= \mathrm{tr}\big(\bar{X}^QFVG\big).\end{aligned} \tag{A.21}$$

□

Theorem A.10. *Assume that both A and A_α are Sylvester. For X^Q, X_α^Q, $X_\alpha^{X^Q}$ and $X^{X_\alpha^Q}$ we have*

$$X_\alpha^{X^Q} = \frac{X_\alpha^Q - X^Q}{2\alpha} = X^{X_\alpha^Q}. \tag{A.22}$$

Proof. Per definition, we have

$$(A + \alpha I)X_\alpha^Q + X_\alpha^Q(A + \alpha I)^T + Q = 0, \tag{A.23}$$

$$AX^Q + X^QA^T + Q = 0. \tag{A.24}$$

By subtracting the two equations, and by using $A_\alpha = A + \alpha I$, we can get either of two results

$$A(X_\alpha^Q - X^Q) + (X_\alpha^Q - X^Q)A^T + 2\alpha X_\alpha^Q = 0, \tag{A.25}$$

$$A_\alpha(X_\alpha^Q - X^Q) + (X_\alpha^Q - X^Q)A_\alpha^T + 2\alpha X^Q = 0. \tag{A.26}$$

Next, we divide the above equations by 2α. The resulting Lyapunov equations have (A.22) as their solution. □

A.3. POWER FORMS OF GAUSSIAN RANDOM VARIABLES

This last section examines the expected value of various power forms of Gaussian random variables.

Theorem A.11. *Consider a Gaussian random variable $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and expected squared value $\Sigma \equiv \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. For symmetric matrices P and Q we have*

$$\begin{aligned}\mathbb{E}[\boldsymbol{x}^T P\boldsymbol{x}\boldsymbol{x}^T Q\boldsymbol{x}] = {} & tr(\Sigma P)\,tr(\Sigma Q) + 2\,tr(\Sigma P\Sigma Q) \\ & - 2\boldsymbol{\mu}^T P\boldsymbol{\mu}\boldsymbol{\mu}^T Q\boldsymbol{\mu}.\end{aligned} \tag{A.27}$$

Proof. We know from Kendrick (1981) (Appendix F.3) that, for symmetric P and Q, and for a *zero-mean* process $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{\mu}$ with covariance $Y = \mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] = \Sigma - \boldsymbol{\mu}\boldsymbol{\mu}^T$, we have

$$\mathbb{E}[\boldsymbol{y}^T P\boldsymbol{y}\boldsymbol{y}^T Q\boldsymbol{y}] = \mathrm{tr}(YP)\mathrm{tr}(YQ) + 2\mathrm{tr}(YPYQ). \tag{A.28}$$

If we apply this result to the expansion of

$$\mathbb{E}[\boldsymbol{x}^T P\boldsymbol{x}\boldsymbol{x}^T Q\boldsymbol{x}] = \mathbb{E}[(\boldsymbol{y}+\boldsymbol{\mu})^T P(\boldsymbol{y}+\boldsymbol{\mu})(\boldsymbol{y}+\boldsymbol{\mu})^T Q(\boldsymbol{y}+\boldsymbol{\mu})] \tag{A.29}$$

and rewrite the result, (A.27) follows. □

Theorem A.12. *Consider Gaussian random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ with joint distribution*

$$\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu_x}\\ \boldsymbol{\mu_y}\end{bmatrix}, \begin{bmatrix}K_{xx} & K_{xy}\\ K_{yx} & K_{yy}\end{bmatrix}\right). \tag{A.30}$$

Also define $\Sigma_{ab} = K_{ab} + \boldsymbol{\mu_a}\boldsymbol{\mu_b}^T$, where the subscripts a and b can be substituted for x and/or y. For symmetric matrices P and Q we now have

$$\begin{aligned}\mathbb{E}[\boldsymbol{x}^T P\boldsymbol{x}\boldsymbol{y}^T Q\boldsymbol{y}] = {} & tr(\Sigma_{xx}P)\,tr(\Sigma_{yy}Q) + 2\,tr(\Sigma_{yx}P\Sigma_{xy}Q) \\ & - 2\boldsymbol{\mu_x}^T P\boldsymbol{\mu_x}\boldsymbol{\mu_y}^T Q\boldsymbol{\mu_y}.\end{aligned} \tag{A.31}$$

Proof. This follows directly from Theorem A.11 with

$$\boldsymbol{x}' = \begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix}, \quad P' = \begin{bmatrix}P & 0\\ 0 & 0\end{bmatrix}, \quad Q' = \begin{bmatrix}0 & 0\\ 0 & Q\end{bmatrix}. \tag{A.32}$$

□

B

Derivatives for the SONIG algorithm

The SONIG update law (4.17) contains various derivatives of matrices. Using (4.8) and (4.9) we can find them. To do so, we first define the scalar quantity

$$P = \Sigma^n_{++} + \sigma_n^2 = K_{++} + \sigma_n^2 - K_{+u}K_{uu}^{-1}\left(K_{uu} - \Sigma^n_{uu}\right)K_{uu}^{-1}K_{u+}. \tag{B.1}$$

We also assume that $m(\boldsymbol{x}) = 0$ for ease of notation. (If not, this can of course be taken into account.) The derivatives of $\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n+1}}$ and Σ^{n+1}_{uu} can now be found element-wise through

$$\begin{aligned}
\frac{d\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n+1}}}{dx_{+_j}} &= \Sigma^n_{uu}K_{uu}^{-1}\left(\frac{dK_{u+}}{dx_{+_j}}P^{-1}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right) + K_{u+}\frac{dP^{-1}}{dx_{+_j}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right)\right.\\
&\quad \left. - K_{u+}P^{-1}\frac{dK_{+u}}{dx_{+_j}}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right),\\
\frac{d^2\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n+1}}}{dx_{+_j}dx_{+_k}} &= \Sigma^n_{uu}K_{uu}^{-1}\left(\frac{d^2K_{u+}}{dx_{+_j}dx_{+_k}}P^{-1}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right) + \frac{dK_{u+}}{dx_{+_j}}\frac{dP^{-1}}{dx_{+_k}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right)\right.\\
&\quad - \frac{dK_{u+}}{dx_{+_j}}P^{-1}\frac{dK_{+u}}{dx_{+_k}}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}} + \frac{dK_{u+}}{dx_{+_k}}\frac{dP^{-1}}{dx_{+_j}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right)\\
&\quad + K_{u+}\frac{d^2P^{-1}}{dx_{+_j}dx_{+_k}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right) - K_{u+}\frac{dP^{-1}}{dx_{+_j}}\frac{dK_{+u}}{dx_{+_k}}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\\
&\quad \left. - \frac{dK_{u+}}{dx_{+_k}}P^{-1}\frac{dK_{+u}}{dx_{+_j}}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}} - K_{u+}\frac{dP^{-1}}{dx_{+_k}}\frac{dK_{+u}}{dx_{+_j}}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}} - K_{u+}P^{-1}\frac{d^2K_{+u}}{dx_{+_j}dx_{+_k}}K_{uu}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{n}}\right),
\end{aligned} \tag{B.2}$$

$$\frac{d\Sigma_{uu}^{n+1}}{dx_{+_j}} = -\Sigma_{uu}^{n} K_{uu}^{-1} \left(\frac{dK_{u+}}{dx_{+_j}} P^{-1} K_{+u} + K_{u+} \frac{dP^{-1}}{dx_{+_j}} K_{+u} + K_{u+} P^{-1} \frac{dK_{+u}}{dx_{+_j}} \right) K_{uu}^{-1} \Sigma_{uu}^{n},$$

$$\begin{aligned}\frac{d^2\Sigma_{uu}^{n+1}}{dx_{+_j} dx_{+_k}} = -\Sigma_{uu}^{n} K_{uu}^{-1} \Bigg(& \frac{d^2 K_{u+}}{dx_{+_j} dx_{+_k}} P^{-1} K_{+u} + \frac{dK_{u+}}{dx_{+_j}} \frac{dP^{-1}}{dx_{+_k}} K_{+u} + \frac{dK_{u+}}{dx_{+_j}} P^{-1} \frac{dK_{+u}}{dx_{+_k}} \\ & + \frac{dK_{u+}}{dx_{+_k}} \frac{dP^{-1}}{dx_{+_j}} K_{+u} + K_{u+} \frac{d^2 P^{-1}}{dx_{+_j} dx_{+_k}} K_{+u} + K_{u+} \frac{dP^{-1}}{dx_{+_j}} \frac{dK_{+u}}{dx_{+_k}} \\ & + \frac{dK_{u+}}{dx_{+_k}} P^{-1} \frac{dK_{+u}}{dx_{+_j}} + K_{u+} \frac{dP^{-1}}{dx_{+_k}} \frac{dK_{+u}}{dx_{+_j}} + K_{u+} P^{-1} \frac{d^2 K_{+u}}{dx_{+_j} dx_{+_k}} \Bigg) K_{uu}^{-1} \Sigma_{uu}^{n}. \end{aligned} \tag{B.3}$$

These expressions contain various additional derivatives. To find them, we need to choose a covariance function. (The above expressions are valid for any covariance function.) If we use the squared exponential covariance function of (4.4), we can derive

$$\begin{aligned}
\frac{dK_{u_i+}}{d\boldsymbol{x}_+} &= \alpha^2 \exp\left(-\frac{1}{2} \left(\boldsymbol{x}_{\boldsymbol{u_i}} - \boldsymbol{x}_+ \right)^T \Lambda^{-1} \left(\boldsymbol{x}_{\boldsymbol{u_i}} - \boldsymbol{x}_+ \right) \right) \left(\boldsymbol{x}_{\boldsymbol{u_i}} - \boldsymbol{x}_+ \right)^T \Lambda^{-1}, \\
\frac{d^2 K_{u_i+}}{d\boldsymbol{x}_+^2} &= \alpha^2 \exp\left(-\frac{1}{2} \left(\boldsymbol{x}_{\boldsymbol{u_i}} - \boldsymbol{x}_+ \right)^T \Lambda^{-1} \left(\boldsymbol{x}_{\boldsymbol{u_i}} - \boldsymbol{x}_+ \right) \right) \left(\Lambda^{-1} \left(\boldsymbol{x}_{\boldsymbol{u_i}} - \boldsymbol{x}_+ \right) \left(\boldsymbol{x}_{\boldsymbol{u_i}} - \boldsymbol{x}_+ \right)^T \Lambda^{-1} - \Lambda^{-1} \right), \\
\frac{dP^{-1}}{d\boldsymbol{x}_+} &= -P^{-2} \frac{dP}{d\boldsymbol{x}_+} = 2P^{-2} \left(K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}^{n}) K_{uu}^{-1} \frac{dK_{u+}}{d\boldsymbol{x}_+} \right), \\
\frac{d^2 P^{-1}}{d\boldsymbol{x}_+^2} &= \frac{d}{d\boldsymbol{x}_+} \left(-P^{-2} \frac{dP}{d\boldsymbol{x}_+} \right) = 2P^{-3} \left(\frac{dP}{d\boldsymbol{x}_+} \right)^T \left(\frac{dP}{d\boldsymbol{x}_+} \right) - P^{-2} \frac{d^2 P}{d\boldsymbol{x}_+^2}, \\
\frac{dP}{d\boldsymbol{x}_+} &= -2K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}^{n}) K_{uu}^{-1} \frac{dK_{u+}}{d\boldsymbol{x}_+}, \\
\frac{d^2 P}{d\boldsymbol{x}_+^2} &= -2 \frac{dK_{+u}}{d\boldsymbol{x}_+} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}^{n}) K_{uu}^{-1} \frac{dK_{u+}}{d\boldsymbol{x}_+} - 2K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}^{n}) K_{uu}^{-1} \frac{d^2 K_{u+}}{d\boldsymbol{x}_+^2}.
\end{aligned} \tag{B.4}$$

CURRICULUM VITÆ

Hildo BIJL

May 31, 1989 Born in Heerhugowaard, The Netherlands

EDUCATION

2006 – 2009	Bachelor in Aerospace Engineering at the Delft University of Technology (TU Delft). Graduated Cum Laude. (Average 9.3 out of 10.)
2007	Minor in Computer Science at the TU Delft.
2008	Minor in Aerospace Analysis and Development at the TU Delft.
2009 – 2012	Master in Dynamics and Control of Aerospace Vehicles at the Control & Simulation department of the TU Delft. Graduated Cum Laude. (Average 9.4 out of 10.)
2009 – 2010	Honors track in Learning and Intelligent Systems at the TU Delft.
2011 – 2012	Graduation project on self-learning systems at the Control & Simulation department of the TU Delft. The final thesis was awarded a 10.
2012 – 2017	Ph.D. at the Delft University of Technology.

AWARDS AND ACHIEVEMENTS

2006	Honorary mention at the International Mathematical Olympiad in Ljubljana, Slovenia.
2008 – 2011	Bronze and silver medals at the North-Western Regional Algorithm Programming Contest (NWERC).
2009	Winner of the Design and Synthesis Exercise symposium for best bachelor project of the faculty of Aerospace Engineering of the TU Delft.
2012	TEDxDelft presentation titled The learning revolution starts here.
2012	PEGASUS award for best Aerospace-related graduate of Europe.

OTHER ACTIVITIES

2006 – 2017	Founded and presided over the Aerostudents community, with the goal of sharing helpful educational resources among students.
2012 – 2014	Board member of the Dutch National Frisbee Association.
2013	Published the Ultimate Trainer's Manual with the goal of helping aspiring Ultimate Frisbee trainers getting started giving trainings.
2012 – 2016	Member of the Dutch National Ultimate Frisbee team, competing at world cups in Osaka, Japan and London, UK.
2015	Coach of the Dutch National Under-23 Ultimate Frisbee team, competing at the world cup in London, UK.
2015	Published the sci-fi novel First Thoughts.

LIST OF SCIENTIFIC PUBLICATIONS

9. **Hildo Bijl**, Thomas Bo Schön, *Optimal controller/observer gains of discounted-cost LQG systems*, submitted for publication to Automatica; also available on arXiv

8. **Hildo Bijl**, Thomas Bo Schön, Jan-Willem van Wingerden & Michel Verhaegen, *A sequential Monte Carlo approach to Thompson sampling for Bayesian optimization*, available on arXiv

7. **Hildo Bijl**, Thomas Bo Schön, Jan-Willem van Wingerden & Michel Verhaegen, *Online Sparse Gaussian Process Training with Input Noise*, IFAC Journal of Systems and Control, Volume 2, December 2017, Pages 1-11; also available on arXiv

6. **Hildo Bijl**, Jan-Willem van Wingerden, Thomas Bo Schön & Michel Verhaegen, *Mean and variance of the LQG cost function*, Automatica, Volume 67, May 2016, Pages 216-223; also available on arXiv

5. **Hildo Bijl**, Jan-Willem van Wingerden, Thomas Bo Schön & Michel Verhaegen, *Online Sparse Gaussian Process Regression Using FITC and PITC Approximations*, 17th IFAC Symposium on System Identification, SYSID 2015, October 19-21, Beijing, China

4. Ioannis Proimadis, **Hildo Bijl** & Jan-Willem van Wingerden, *A Kernel Based Approach for LPV Subspace Identification*, 1st IFAC Workshop on Linear Parameter Varying Systems, LPVS 2015, October 7-9, Grenoble, France

3. **Hildo Bijl**, Jan-Willem van Wingerden & Michel Verhaegen, *Applying Gaussian Processes to Reinforcement Learning for Fixed-Structure Controller Synthesis*, 19th World Congress of the International Federation of Automatic Control, IFAC 2014, August 24-29, Cape Town, South Africa

2. **Hildo Bijl**, Erik-Jan Van Kampen, Ping Chu & Bob Mulder, *Guaranteed Globally Optimal Continuous Reinforcement Learning*, 52nd Aerospace Sciences Meeting, AIAA SciTech 2014, January 13-17, National Harbor, Maryland

1. Vahram Stepanyan, Kalmanje Krishnakumar, Jonathan Barlow & **Hildo Bijl**, *Adaptive Estimation Based Loss of Control Detection and Mitigation*, AIAA Guidance, Navigation and Control Conference, GNC 2011, August 8-11, Portland, Oregon

REFERENCES

REFERENCES

Shipra Agrawal and Navin Goyal, *Analysis of Thompson sampling for the multi-armed bandit problem,* in *JMLR Workshop and Conference Proceedings*, Volume 23 (2012) pages 39.1–39.26.

Mauricio A. Álvarez, Lorenzo Rosasco and Neil D. Lawrence, *Kernels for vector-valued functions: A review,* Foundations and Trends in Machine Learning 4, pages 195–266 (2012).

Peter Bjørn Andersen, Mac Gaunaa, Christian Bak and Thomas Buhl, *Load alleviation on wind turbine blades using variable airfoil geometry,* in *Proceedings of the EWEC, Athens, Greece* (2006).

Brian D. O. Anderson and John B. Moore, *Linear system optimisation with prescribed degree of stability,* in *Proceedings of the Institution of Electrical Engineers*, Volume 116 (1969) pages 2083–2087.

Brian D. O. Anderson and John B. Moore, *Optimal Control: Linear Quadratic Methods* (Prentice Hall, 1990).

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund and Robert E. Schapire, *Gambling in a rigged casino: The adversarial multi-armed bandit problem,* in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science* (1995) pages 322–331.

Christian Bak, Mac Gaunaa, Peter B. Andersen, Thomas Buhl, Per Hansen, Kasper Clemmensen and Rene Moeller, *Wind tunnel test on wind turbine airfoil with adaptive trailing edge geometry,* in *Proceedings of the 45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada* (2007).

R.H. Bartels and G.W. Stewart, *Solution of the matrix equation $AX + XB = C$,* Communications of the ACM 15, pages 820–826 (1972).

Kinjal Basu and Souvik Ghosh, *Analysis of Thompson Sampling for Gaussian Process Optimization in the Bandit Setting*, Technical Report (arXiv.org, 2017).

Santiage Basualdo, *Load alleviation on wind turbine blades using variable airfoil geometry,* Journal of Wind Engineering 29, pages 169–182 (2005).

Jonathan Berg, Dale Berg and Jon White, *Fabrication, integration, and initial testing of a SMART rotor,* in *Proceedings of the 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Nashville, Tennessee* (2012).

Felix Berkenkamp, Angela P. Schoellig and Andreas Krause, *Safe controller optimization for quadrotors with Gaussian processes,* in *Proceedings of the International Conference on Robotics and Automation (ICRA), 2016* (2016).

Dimitri P. Bertsekas and John N. Tsitsiklis, *Neuro-Dynamic Programming* (Athena Scientific, 1996).

Fernando D. Bianchi, Hernán de Battista and Ricardo J. Mantz, *Wind Turbine Control Systems* (Springer, 2007).

Hildo Bijl, *SONIG source code,* (2016), https://github.com/HildoBijl/SONIG.

Hildo Bijl and Thomas B. Schön, *Optimal controller/observer gains of discounted-cost LQG systems,* Submitted for publication (2017).

Hildo Bijl, Thomas B. Schön, Jan-Willem van Wingerden and Michel Verhaegen, *A sequential Monte Carlo approach to Thompson sampling for Bayesian optimization,* arXiv.org (2017a).

Hildo Bijl, Thomas B. Schön, Jan-Willem van Wingerden and Michel Verhaegen, *System identification through online sparse Gaussian process regression with input noise,* IFAC Journal of Systems and Control 2, pages 1–11 (2017b).

Hildo Bijl, Jan-Willem van Wingerden, Thomas B. Schön and Michel Verhaegen, *Online sparse Gaussian process regression using FITC and PITC approximations,* in *Proceedings of the IFAC symposium on System Identification, SYSID, Beijing, China* (2015).

Hildo Bijl, Jan-Willem van Wingerden, Thomas B. Schön and Michel Verhaegen, *Mean and variance of the LQG cost function,* Automatica 67, pages 216–223 (2016).

Hildo Bijl, Jan-Willem van Wingerden and Michel Verhaegen, *Applying Gaussian processes to reinforcement learning for fixed-structure controller synthesis,* in *Proceedings of the 19th IFAC World Congress* (2014) pages 10391–10396.

Okko H. Bosgra, Huibert Kwakernaak and Gjerrit Meinsma, *Design Methods for Control Systems* (Dutch Institute of Systems and Control (DISC), 2008).

Ervin A. Bossanyi, *The design of closed loop controllers for wind turbines,* Wind Energy 3, pages 149–163 (2000).

Boubekeur Boukhezzar and Houria Siguerdidjane, *Comparison between linear and nonlinear control strategies for variable speed wind turbines,* Control Engineering Practice 18, pages 1357–1368 (2010).

Stephen Boyd and Lieven Vandenberghe, *Convex Optimization* (Cambridge University Press, 2004).

Corentin Briat, *Linear Parameter-Varying and Time-Delay Systems* (Springer, 2015).

Eric Brochu, Vlad M Cora and Nando de Freitas, *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, Technical Report (University of British Columbia, 2010).

Thomas Buhl, Mac Gaunaa and Christian Bak, *Potential load reduction using airfoils with variable trailing edge geometry,* Journal of Solar Energy Engineering 127, pages 503–516 (2005).

Joaquin Q. Candela, Agathe Girard, Jan Larsen and Carl E. Rasmussen, *Propagation of uncertainty in Bayesian kernel models - application to multiple-step ahead forecasting,* in *International Conference on Acoustics, Speech and Signal Processing*, Volume 2 (MIT Press, 2003) pages 701–704.

Joaquin Q. Candela and Carl E. Rasmussen, *A unifying view of sparse approximate Gaussian process regression,* Journal of Machine Learning Research 6, pages 1939–1959 (2005).

Damien Castaignet, Thanasis Barlas, Thomas Buhl, Niels K. Poulsen, Jens Jakob Wedel-Heinen, Niels A. Olesen, Christian Bak and Taeseong Kim, *Full-scale test of trailing edge flaps on a Vestas V27 wind turbine: active load reduction and system identification,* Journal of Wind (2012).

Krzysztof Chalupka, Christopher K. I. Williams and Iain Murray, *A framework for evaluating approximation methods for Gaussian process regression,* Machine Learning Research 14, pages 333–350 (2013).

Olivier Chapelle and Lihong Li, *An empirical evaluation of Thompson sampling,* in *Advances in Neural Information Processing Systems*, Volume 24 (Curran Associates, Inc., 2011) pages 2249–2257.

Kamalika Chaudhuri, Yoav Freund and Daniel J. Hsu, *A parameter-free hedging algorithm,* in *Advances in Neural Information Processing Systems*, Volume 22 (2009) pages 297–305.

Tianshi Chen, Henrik Ohlsson and Lennart Ljung, *On the estimation of transfer functions, regularizations and Gaussian processes–revisited,* Automatica 48, pages 1525–1535 (2012).

Bogdan D. Ciubotaru, Marcel Staroswiecki and Nicolai D. Christov, *Extended hybrid technique for control redesign with stabilization and correction,* in *American Control Conference (ACC)* (2013) pages 5152–5158.

Emmanual G. Collins and Majura F. Selekwa, *Fuzzy quadratic weights for variance constrained LQG design,* in *Proceedings of the 38th IEEE Conference on Decision and Control, Phoenix, Arizona, USA* (1999).

Richard Conway and Roberto Horowitz, *A quasi-Newton algorithm for LQG control design with variance constraints,* in *Proceedings of the Dynamic Systems and Control Conference, Ann Arbor, Michigan, USA* (2008).

Dennis D. Cox and Susan John, *SDO: A statistical method for global optimization,* in *Multidisciplinary Design Optimization: State-of-the-Art* (1997) pages 315–329.

Legel Csató and Manfred Opper, *Sparse online Gaussian processes,* Neural Computation 14, pages 641–669 (2002).

Patrick Dallaire, Camille Besse and Brahim Chaib-draa, *Learning Gaussian process models from uncertain data,* in *Proceedings of the 16th International Conference on Neural Information Processing* (2009).

Andreas C. Damianou, Michalis K. Titsias and Neil D. Lawrence, *Variational inference for latent variables and uncertain inputs in Gaussian processes,* Journal of Machine Learning Research 17, pages 1–62 (2016).

Robbert B. Davies, *Algorithm AS 155: The distribution of a linear combination of χ^2 random variables,* Journal of the Royal Statistical Society. Series C (Applied Statistics) 29, pages 323–333 (1980).

Richard Dearden, Nir Friedman and Stuart Russell, *Bayesian Q-learning,* in *In AAAI/IAAI* (AAAI Press, 1998) pages 761–768.

Marc P. Deisenroth, *Efficient Reinforcement Learning using Gaussian Processes*, Ph.D. thesis, Karlsruhe Institute of Technology (2010).

Marc P. Deisenroth and Jun W. Ng, *Distributed Gaussian processes,* in *Proceedings of the International Conference on Machine Learning (ICML)* (Lille, France, 2015).

Marc P. Deisenroth and Carl E. Rasmussen, *PILCO: A model-based and data-efficient approach to policy search,* in *Proceedings of the International Conference on Machine Learning (ICML), Bellevue, Washington, USA* (ACM Press, 2011) pages 465–472.

Pierre Del Moral, Arnaud Doucet and Ajay Jasra, *Sequential Monte Carlo samplers,* Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68, pages 411–436 (2006).

L.C.W. Dixon and G.P. Szegö, *The global optimisation problem: an introduction,* in *Towards global optimization*, Volume 2, edited by L.C.W. Dixon and G.P. Szegö (North-Holland Publishing, 1978) pages 1–15.

Yaakov Engel, Shie Mannor and Ron Meir, *Bayes meets Bellman: The Gaussian process approach to temporal difference learning,* in *Proceedings of the 20th International Conference on Machine Learning*, edited by Tom Fawcett and Nina Mishra (2003).

Yaakov Engel, Shie Mannor and Ron Meir, *Reinforcement learning with Gaussian processes,* in *Proceedings of the 22nd International Conference on Machine Learning* (ACM Press, 2005) pages 201–208.

Tim van Engelen and Henk Braam, *TURBU Offshore; Computer program for frequency domain analysis of horizontal axis offshore wind turbines - Implementation*, Technical Report Report ECN-C-04-079 (ECN, 2004).

Simone Formentin, Klaske van Heusden and Alireza Karimi, *A comparison of model-based and data-driven controller tuning,* International Journal of Adaptive Control and Signal Processing 28, pages 882–897 (2014).

Gregg Freebury and Walter Musial, *Determining equivalent damage loading for full-scale wind turbine blade fatigue tests,* in *Proceedings of the 19th American Society of Mechanical Engineers (ASME) Wind Energy Symposium, Reno, Nevada* (2000).

Nando de Freitas, Alex Smola and Masrour Zoghi, *Regret Bounds for Deterministic Gaussian Process Bandits*, Technical Report (arXiv.org, 2012).

Roger Frigola, Yutian Chen and Carl E. Rasmussen, *Variational Gaussian process state-space models,* in *Advances in Neural Information Processing Systems (NIPS)* (2014).

Roger Frigola, Fredrik Lindsten, Thomas B. Schön and Carl E. Rasmussen, *Bayesian inference and learning in Gaussian process state-space models with particle MCMC,* in *Advances in Neural Information Processing Systems (NIPS) 26* (Lake Tahoe, NV, USA, 2013).

Yarin Gal, Mark van der Wilk and Carl E. Rasmussen, *Distributed variational inference in sparse Gaussian process regression and latent variable models,* in *Advances in Neural Information Processing Systems (NIPS)* (2014).

Pieter M. O. Gebraad, *Data-Driven Wind Plant Control*, Ph.D. thesis, Delft University of Technology (2014).

Agathe Girard and Roderick Murray-Smith, *Learning a Gaussian Process Model with Uncertain Inputs*, Technical Report 144 (Department of Computing Science, University of Glasgow, 2003).

Agathe Girard, Carl E. Rasmussen, Joaquin Q. Candela and Roderick Murray-Smith, *Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting,* in *Advances in Neural Information Processing Systems* (MIT Press, 2003) pages 545–552.

Paul W. Goldberg, Christopher K. I. Williams and Christopher M. Bishop, *Regression with input-dependent noise: A Gaussian process treatment,* in *Advances in Neural Information Processing Systems (NIPS)*, Volume 10 (MIT Press, 1997) pages 493–499.

Michael Green and David J. N. Limebeer, *Linear Robust Control* (Dover Publications, 1995).

Michael J. Grimble and Michael A. Johnson, *Optimal Control and Stochastic Estimation* (Wiley, 1988).

Steffen Grünewälder, Jean-Yves Audibert, Manfred Opper and John Shawe-Taylor, *Regret bounds for Gaussian process bandit problems,* in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)* (2010).

Kurt S. Hansen, Rebecca J. Barthelmie, Leo E. Jensen and Anders Sommer, *The impact of turbulence intensity and atmospheric stability on power deficits due to wind turbine wakes at horns rev wind farm,* Wind Energy 15, pages 183–196 (2012).

Erich Hau, *Wind Turbines: Fundamentals, Technologies, Application, Economics* (Springer, 2006).

Joachim Heinz, Niels N. Sørensen and Frederik Zahle, *Investigation of the load reduction potential of two trailing edge flap controls using CFD,* Journal of Wind Energy (2010), 10.1002/we.435.

Philipp Hennig and Christian J. Schuler, *Entropy search for information-efficient global optimization,* Journal of Machine Learning Research 13, pages 1809–1837 (2012).

James Hensman, Fusi Nicoló and Neil D. Lawrence, *Gaussian processes for big data,* in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI), Bellevue, Washington, USA* (2013).

José M. Hernández-Lobato, Matthew W. Hoffman and Zoubin Ghahramani, *Predictive entropy search for efficient global optimization of black-box functions,* in *Advances in Neural Information Processing Systems 27* (Curran Associates, Inc., 2014).

Matthew Hoffman, Eric Brochu and Nando de Freitas, *Portfolio allocation for Bayesian optimization,* in *Uncertainty in Artificial Intelligence (UAI)* (2011) pages 327–336.

Marco F. Huber, *Recursive Gaussian process regression,* in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, Canada* (2013) pages 3362–3366.

Marco F. Huber, *Recursive Gaussian process: On-line regression and learning,* Pattern Recognition Letters 45, pages 85–91 (2014).

Anton W. Hulskamp, *The Smart Rotor Concept on Wind Turbines*, Ph.D. thesis, TU Delft (2011).

Sung-ho Hur and William E. Leithead, *Model predictive and linear quadratic gaussian control of a wind turbine,* Optimal Control Applications and Methods 38, pages 88–111 (2017).

Carl Jidling, Niklas Wahlström, Adrian Wills and Thomas B. Schön, *Linearly constrained Gaussian processes*, Technical Report (arXiv.org, 2017).

Donald R. Jones, *A taxonomy of global optimization methods based on response surfaces,* Journal of Global Optimization 21, pages 345–383 (2001).

Donald R. Jones, Matthias Schonlau and William J. Welch, *Efficient global optimization of expensive black-box functions,* Journal of Global Optimization 13, pages 455–492 (1998).

Rudolf E. Kalman, *A new approach to linear filtering and prediction problems,* Journal of Basic Engineering 82, pages 35–45 (1960).

Rudolf E. Kalman and Richard S. Bucy, *New results in linear filtering and prediction theory,* Journal of Basic Engineering 83, pages 95–107 (1961).

Bei Kang, Chukwuemeka Aduba and Chang-Hee Won, *Statistical control for performance shaping using cost cumulants,* IEEE Transactions on Automatic Control 59, pages 249–255 (2014).

David A. Kendrick, *Stochastic Control for Economic Models* (McGraw-Hill, 1981).

Robert D. Kleinberg, *Nearly tight bounds for the continuum-armed bandit problem,* in *Advances in Neural Information Processing Systems 17* (MIT Press, 2004) pages 697–704.

Jeonghwan Ko, Andrew J. Kurdila and Thomas W. Strganac, *Nonlinear control of a prototypical wing section with torsional nonlinearity,* Journal of Guidance, Control and Dynamics 20, pages 1181–1189 (1997).

Jeonghwan Ko, Thomas W. Strganac and Andrew J. Kurdila, *Stability and control of a structurally nonlinear aeroelastic system,* Journal of Guidance, Control and Dynamics 21, page 718/725 (1998).

Jus Kocijan, *Modelling and control of dynamic systems using Gaussian process models* (Springer, Basel, Switzerland, 2016).

Jus Kocijan, Agathe Girard, Blaz Banko and Roderick Murray-Smith, *Dynamic systems identification with Gaussian processes,* Mathematical and Computer Modelling of Dynamical Systems 11, pages 411—424 (2005).

Peng Kou, Feng Gao and Xiaohong Guan, *Sparse online warped Gaussian process for wind power probabilistic forecasting,* Applied Energy 108, pages 410–428 (2013).

Harold J. Kushner, *A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise,* Journal of Basic Engineering 86, pages 97–106 (1964).

Vladimír Kučera and Jiří Cigler, *Optimal control systems with prescribed eigenvalues,* International Symposium on Communications, Control and Signal Processing (ISCCSP) , pages 1–6 (2010).

Huibert Kwakernaak and Raphael Sivan, *Linear optimal control systems* (Wiley Interscience, 1972).

Ioan D. Landau, Tudor-Bogdan Airimitoaie, Abraham Castellanos-Silva and Aurelian Constantinescu, *Adaptive and Robust Active Vibration Control: Methodology and Tests* (Springer, 2016).

Quoc V. Le, Alex J. Smola and Stéphane Canu, *Heteroscedastic Gaussian process regression,* in *Proceedings of the International Conference on Machine Learning (ICML), Bonn, Germany* (2005) pages 489–496.

Rick Lind and Dario H. Baldelli, *Identifying parameter-dependent volterra kernels to predict aeroelastic instabilities,* AIAA Journal 43, pages 2496–2502 (2005).

Daniel Lizotte, Tao Wang, Michael Bowling and Dale Schuurmans, *Automatic gait optimization with Gaussian process regression,* in *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (2007) pages 944–949.

Daniel James Lizotte, *Practical Bayesian Optimization*, Ph.D. thesis, University of Alberta (2008).

Lennart Ljung, *System Identification: Theory for the User* (Prentice Hall, Upper Saddle River, NJ, USA, 1999).

Charles F. van Loan, *Computing integrals involving the matrix exponential,* IEEE Transactions on Automatic Control 23, pages 395–404 (1978).

Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal and Sebastian Trimpe, *Automatic LQR tuning based on Gaussian process global optimization,* in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2016* (IEEE, 2016).

Arakaparampil M. Mathai and Serge B. Provost, *Quadratic Forms in Random Variables* (Taylor & Francis, 1992).

Andrew McHutchon, *Nonlinear Modelling and Control using Gaussian Processes*, Ph.D. thesis, Churchill College (2014).

Andrew McHutchon and Carl E. Rasmussen, *Gaussian process training with input noise,* in *Advances in Neural Information Processing Systems (NIPS), Granada, Spain* (2011) pages 1341–1349.

Yunhe Meng, Qifeng Chen and Qing Ni, *A new geometric guidance approach to spacecraft near-distance rendezvous problem,* Acta Astronautica 129, pages 374–383 (2016).

Thomas P. Minka, *Expectation propagation for approximate Bayesian inference,* in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence* (2001).

Jonas Mockus, Vytautas Tiesis and Antanas Zilinskas, *The application of Bayesian methods for seeking the extremum* (Elsevier, Amsterdam, 1978) pages 117–129.

Cristopher Moné, Aaron Smith, Ben Maples and Maureen Hand, *2013 Cost of Wind Energy Review*, Technical Report NREL/TP-5000-63267 (National Renewable Energy Laboratory, 2015).

Sachin T. Navalkar, *Iterative Data-Driven Load Control for Flexible Wind Turbine Rotors*, Ph.D. thesis, Delft University of Technology (2016).

Sachin T. Navalkar, Lars O. Bernhammer, Jurij Sodja, Edwin van Solingen, Gijs A. M. van Kuik, and Jan-Willem van Wingerden, *Wind tunnel tests with combined pitch and free-floating flap control: Data-driven iterative feedforward controller tuning,* Wind Energy Science (2016), 10.5194/wes-2016-14.

Adam Niesłony, *Determination of fragments of multiaxial service loading strongly influencing the fatigue of machine components,* Mechanical Systems and Signal Processing 23, pages 2712–2721 (2009).

Bernt Øksendal, *Stochastic Differential Equations* (Springer-Verlag, 1985).

Todd O'Neil, Heather Gilliatt and Thomas W. Strganac, *Investigations of aeroelastic response for a system with continuous structural nonlinearities,* in *Proceedings of the 37th Structures, Structural Dynamics and Materials Conference, Salt Lake City, Utah* (1996).

Todd O'Neil and Thomas W. Strganac, *Nonlinear aeroelastic response - Analyses and experiments,* in *Proceedings of the 36th Structures, Structural Dynamics and Materials Conference, New Orleans, Louisiana* (1996).

Todd O'Neil and Thomas W. Strganac, *Aeroelastic response of a rigid wing supported by nonlinear springs,* Journal of Aircraft 35, pages 616–622 (1998).

Michael Osborne, *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature,* Ph.D. thesis, University of Oxford (2010).

Art B. Owen, *Monte Carlo theory, methods and examples* (2013).

Sandeep Pandey and Christopher Olston, *Handling advertisements of unknown quality in search advertising,* in *Advances in Neural Information Processing Systems*, Volume 19 (MIT Press, 2007) pages 1065–1072.

Jinkyoo Park and Kincho H. Law, *Bayesian Ascent (BA): A data-driven optimization scheme for real-time control with application to wind farm power maximization,* IEEE Transactions on Control Systems Technology (2015).

Georgios Pechlivanoglou, *Passive and active flow control solutions for wind turbine blades,* Ph.D. thesis, Technischen Universität Berlin (2012).

Cédric Philibert and Hannele Holttinen, *Technology Roadmap: Wind Energy – 2013 edition*, Technical Report (International Energy Agency, 2013).

Gianluigi Pillonetto, Alessandro Chiuso and Giuseppe De Nicolao, *Prediction error identification of linear systems: a nonparametric Gaussian regression approach,* Automatica 47, pages 291—305 (2011).

Gianluigi Pillonetto and Giuseppe De Nicolao, *A new kernel-based approach for linear system identification,* Automatica 46, pages 81—93 (2010).

Sara C. Pryor and Rebecca J. Barthelmie, *Comparison of potential power production at on- and offshore sites,* Wind Energy 4, pages 173–181 (2001).

Sara C. Pryor and Rebecca J. Barthelmie, *Statistical analysis of flow characteristics in the coastal zone,* Journal of Wind Engineering and Industrial Aerodynamics 90, pages 201–221 (2002).

Ananth Ranganathan, Ming-Hsuan Yang and Jeffrey Ho, *Online sparse Gaussian process regression and its applications,* IEEE Transactions on Image 20, pages 391–404 (2011).

Carl E. Rasmussen and M. Kuss, *Gaussian processes in reinforcement learning,* in *Advances in Neural Information Processing Systems 16* (MIT Press, 2004) pages 751–759.

Carl E. Rasmussen and Christopher K.I. Williams, *Gaussian Processes for Machine Learning* (MIT Press, 2006).

Karl J. Åström, *Introduction to Stochastic Control Theory* (Academic Press, 1970).

Stephen O. Rice, *Mathematical analysis of random noise,* Bell System Technical Journal 23, pages 282–332 (1944).

Silvio Rodrigues, Carlos Restrepo, George Katsouris, Rodrigo Teixeira Pinto, Maryam Soleimanzadeh, Peter Bosman and Pavol Bauer, *A multi-objective optimization framework for offshore wind farm layouts and electric infrastructures,* Energies 9, page 216 (2016).

Ilan Rusnak, *Least mean squares error based filter of linear system with prescribed convergence rate,* in *IEEE International Conference on the Science of Electrical Engineering (ICSEE)* (2016) pages 1–5.

Michael K. Sain and Stanley R. Liberty, *Performance-measure densities for a class of LQG control systems,* IEEE Transactions on Automatic Control 16, pages 431–439 (1971).

Mathieu Salzmann and Raquel Urtasun, *Implicitly constrained Gaussian process regression for monocular non-rigid pose estimation,* in *Advances in Neural Information Processing Systems*, Volume 23, edited by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel and A. Culotta (Curran Associates, Inc., 2010) pages 2065–2073.

Feike J. Savenije and Johan M. Peeringa, *Aero-elastic simulation of offshore wind turbines in the frequency domain*, Technical Report Report ECN-E-09-060 (Energy research centre ECN, The Netherlands, 2009).

Morton I. Schwartz, *Distribution of the time-average power of a Gaussian process,* IEEE Transactions on Information Theory 16, pages 17–26 (1970).

Matthias Seeger, Christopher K.I. Williams and Neil D. Lawrence, *Fast forward selection to speed up sparse Gaussian process regression,* in *Workshop on AI and Statistics* (2003).

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams and Nando de Freitas, *Taking the human out of the loop: A review of Bayesian optimization,* Proceedings of the IEEE 104, pages 148–175 (2016).

Bobak Shahriari, Ziyu Wang, Matthew W. Hoffman, Alexandre Bouchard-Côté and Nando de Freitas, *An Entropy Search Portfolio for Bayesian Optimization*, Technical Report (University of Oxford, 2014).

Sigurd Skogestad and Ian Postlethwaite, *Multivariable Feedback Control: Analysis and Design* (John Wiley & Sons, 2005).

Alex J. Smola and Peter Bartlett, *Sparse greedy Gaussian process regression,* in *Advances in Neural Information Processing Systems (NIPS), Vancouver, Canada* (2001) pages 619–625.

Edward Snelson and Zoubin Ghahramani, *Sparse Gaussian processes using pseudo-inputs,* in *Advances in Neural Information Processing Systems (NIPS)* (2006) pages 1257–1264.

Edward Snelson and Zoubin Ghahramani, *Variable noise and dimensionality reduction for sparse Gaussian processes,* in *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI), Cambridge, Massachusetts, USA* (2006).

Niranjan Srinivas, Andreas Krause, Sham M. Kakade and Matthias W. Seeger, *Information-theoretic regret bounds for Gaussian process optimization in the bandit setting,* IEEE Transactions on Information Theory 58, pages 3250 – 3265 (2012).

Robert F. Stengel, *Optimal Control and Estimation* (Dover Publications, 1994).

Yanan Sui, Alkis Gotovos, Joel W. Burdick and Andreas Krause, *Safe exploration for optimization with Gaussian processes,* in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, ICML'15, Volume 37 (2015) pages 997–1005.

Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 1998).

Andreas Svensson and Thomas B. Schön, *A flexible state space model for learning nonlinear dynamical systems,* Automatica 80, pages 189–199 (2017).

Andreas Svensson, Arno Solin, Simo Särkkä and Thomas B. Schön, *Computationally efficient Bayesian learning of Gaussian process state space models,* in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS), Cadiz, Spain* (2016).

Nobuya Takahashi and Osamu Sato, *Guaranteed cost control of robot manipulator with prescribed degree of stability,* Artificial Life and Robotics 21, pages 520–524 (2016).

The MathWorks Inc., *Nonlinear modeling of a magneto-rheological fluid damper.* Example file provided by Matlab ® R2015b System Identification Toolbox™ (2015), available at http://mathworks.com/help/ident/examples/nonlinear-modeling-of-a-magneto-rheological-fluid-damper.html.

William R. Thompson, *On the likelihood that one unknown probability exceeds another in view of the evidence of two samples,* Biometrika 25, pages 285–294 (1933).

Michalis K. Titsias, *Variational learning of inducing variables in sparse Gaussian processes,* in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (Clearwater Beach, FL, USA, 2009).

Michalis K. Titsias and Neil D. Lawrence, *Bayesian Gaussian process latent variable model,* in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 13)* (2010) pages 844–851.

Aimo Torn and Antanas Zilinskas, *Global Optimization* (Springer-Verlag New York, Inc., 1989).

Roland Tóth, *Modeling and Identification of Linear Parameter-Varying Systems* (Springer, 2010).

Harry L. Trentelman, Anton A. Stoorvogel and Malo Hautus, *Control Theory for Linear Systems* (Springer, 2001).

Matthis Türk and Stefan Emeis, *The dependence of offshore turbulence intensity on wind speed,* Journal of Wind Engineering and Industrial Aerodynamics 98, pages 466–471 (2010).

Gijs J. van der Veen, Jan-Willem van Wingerden, Paul A. Fleming, Andrew K. Scholbrock and Michael Verhaegen, *Global data-driven modeling of wind turbines in the presence of turbulence,* Control Engineering Practice 21, pages 441–454 (2013).

Edwin van Solingen, *Control design for two-bladed wind turbines*, Ph.D. thesis, Delft University of Technology (2015).

Edwin van Solingen and Jan-Willem van Wingerden, *Linear individual pitch control design for two-bladed wind turbines,* Wind Energy 18, pages 677–697 (2015).

Emmanuel Vazquez and Julien Bect, *Convergence properties of the expected improvement algorithm with fixed mean and covariance functions,* Journal of Statistical Planning and Inference 140, pages 3088–3095 (2010).

Michel Verhaegen and Vincent Verdult, *Filtering and System Identification: A Least Squares Approach* (Cambridge University Press, 2007).

Julien Villemonteix, Emmanuel Vazquez and Eric Walter, *An informational approach to the global optimization of expensive-to-evaluate functions,* Journal of Global Optimization 43, pages 373–389 (2009).

Draguna Vrabie, Kyriakos G. Vamvoudakis and Frank L. Lewis, *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles* (The Institution of Engineering and Technology, 2013).

Hermann-Josef Wagner and Jyotirmay Mathur, *Introduction to Wind Energy Systems* (Springer, 2013).

Niklas Wahlström, Patrix Axelsson and Fredrik Gustafsson, *Discretizing stochastic dynamical systems using Lyapunov equations,* in *Proceedings of the 19th IFAC World Congress* (2014).

Chunyi Wang and Radford M. Neal, *Gaussian Process Regression with Heteroscedastic or Non-Gaussian Residuals*, Technical Report (arXiv.org, 2012).

Jiandong Wang, Akira Sano, Tongwen Chen and Biao Huang, *Identification of Hammerstein systems without explicit parameterization of nonlinearity,* International Journal of Control 82, pages 937–952 (2009).

Christopher J.C.H. Watkins, *Learning from Delayed Rewards*, Ph.D. thesis, King's College (1989).

Jan-Willem van Wingerden, *Control of Wind Turbines with 'Smart' Rotors: Proof of Concept & LPV Subspace Identification*, Ph.D. thesis, TU Delft (2008).

Paul H. Wirsching, Thomas L. Paez and Keith Ortiz, *Random Vibrations, Theory and Practice* (John Wiley & Sons, Inc., 1995).

Chang-Hee Won, Cheryl B. Schrader and Anthony N. Michel, *Advances in Statistical Control, Algebraic Systems Theory, and Dynamic Systems Characteristics* (Birkhäuser Boston, 2008).