

GAUSSIAN PROCESS REGRESSION TECHNIQUES

WITH APPLICATIONS TO WIND TURBINES

A student version of the thesis

by **HILDO BIJL**

This is not my official thesis but an adjusted ‘student’ version, made for people aspiring to learn more about Gaussian process regression. It provides more explanations, background information and intuitive approaches, as compared to my official thesis.

The latest version can be downloaded at <http://www.hildobijl.com/Research.php>. Source code has been uploaded at <https://github.com/HildoBijl/GPRT>.

*Life is about understanding the world well enough
that making a difference suddenly becomes easy.*

CONTENTS

Preface	xii
The goal of this student version	xii
Thoughts behind the writing style	xii
The set-up of the student version and its chapters	xiii
Final words	xv
1 Introduction	1
1.1 Issues behind Gaussian process regression	2
1.1.1 What does Gaussian process regression do?	2
1.1.2 Limitations of Gaussian process regression	3
1.2 Why wind turbine control?	4
1.2.1 The issue of cost	4
1.2.2 The issue of size	5
1.2.3 The issue of vibrations	5
1.2.4 The issue of frequencies	7
1.2.5 The issue of control	8
1.3 Overview of this booklet	9
2 An intuitive introduction to Gaussian process regression	11
2.1 Approximating a variable	12
2.1.1 The prior distribution	12
2.1.2 Making measurements of f	13
2.1.3 Merging distributions	14
2.1.4 Multiple variables to approximate	14
2.1.5 Measuring only a single variable	16
2.2 Approximating variables we have not measured	17
2.2.1 Making prior assumptions on function values	17
2.2.2 Making predictions about other function values	19
2.2.3 Splitting up the measurement and trial points	19
2.2.4 Implementing noisy measurements	20
2.3 Different views on Gaussian processes	22
2.3.1 The formal definition of a Gaussian process	23
2.3.2 The intuitive view of a Gaussian process	24
2.3.3 The mathematical view of Gaussian process regression	24
2.4 Multi-dimensional inputs and outputs	25
2.4.1 Using multi-dimensional input points	25
2.4.2 Using multi-dimensional output points	26
2.4.3 A simplification when using multi-dimensional outputs	27
2.4.4 The covariance functions within the covariance matrix	27

2.5	The derivative and integral of a Gaussian process	28
2.5.1	The derivative of a Gaussian process	28
2.5.2	The mean and covariance of the derivative	28
2.5.3	Implementing derivative measurements	29
2.5.4	Integrals of Gaussian processes	30
2.6	Identifying the dynamics of a pitch-plunge system	32
2.6.1	The pitch-plunge system set-up	32
2.6.2	The pitch-plunge system equations of motion	33
2.6.3	A first approximation of the state transition function	35
2.6.4	Making a higher-dimensional approximation	37
2.7	Literature – A short history of GP regression	37
3	Details of the covariance function	39
3.1	The basics of tuning hyperparameters	40
3.1.1	Probabilities and likelihoods	41
3.1.2	Integrating over hyperparameters	42
3.1.3	The maximum likelihood method	42
3.1.4	Optimizing the log-likelihood	44
3.1.5	Using different hyper-priors	47
3.2	Other covariance functions and tuning methods	48
3.2.1	Different kinds of covariance functions	48
3.2.2	The covariance function for linear functions	49
3.2.3	Trading off covariance functions	52
3.2.4	Combining covariance functions	54
3.3	Applying GP regression to linear relations	56
3.3.1	Measuring linear relations of function values	56
3.3.2	Applying constrained Gaussian process regression	56
3.3.3	Hyperparameter tuning for constrained GP regression	57
3.3.4	A practical use of constrained GP regression	58
3.4	Linearized modeling of the pitch-plunge system	60
3.4.1	The linearized discrete equations of motion	60
3.4.2	Applying the linear covariance function	61
3.4.3	Switching to the nonlinear system	62
3.5	Approximating a quadratic value function	64
3.5.1	The LQG problem set-up	64
3.5.2	Approximating the value function for a single controller	65
3.5.3	Adding noise to the problem	66
3.5.4	Varying the controller settings	68
3.5.5	Further extensions: unstable and nonlinear systems	70
3.6	Overview of literature	72
3.6.1	Literature on hyperparameter tuning	72
3.6.2	Literature on covariance functions	72
3.6.3	Literature on constrained GP regression	73

4 Sparse and online Gaussian process regression	75
4.1 Sparse Gaussian process regression	76
4.1.1 A notation for discussing computational requirements	76
4.1.2 Analyzing the computational requirements	77
4.1.3 Faster prediction: using inducing input points	78
4.1.4 Faster training: using measurements individually	81
4.1.5 More flexibility: using measurements in subgroups	83
4.1.6 An incorrect alternative view on sparse GP regression	84
4.2 Online Gaussian process regression	87
4.2.1 Regular online Gaussian process regression	87
4.2.2 Sparse online Gaussian process regression	88
4.2.3 Online FITC regression	88
4.2.4 Online PITC regression	90
4.2.5 Numerical stability of the online methods	92
4.3 Choosing the inducing input points	93
4.3.1 Manually choosing the inducing input points offline	93
4.3.2 Automatically tuning the inducing input points offline	94
4.3.3 Adjusting the inducing input points online	97
4.3.4 A different merging order	98
4.4 Applications of sparse online GP regression	99
4.4.1 A comparison between algorithms	100
4.4.2 Application to the pitch-plunge system	103
4.5 Overview of literature and contributions	106
4.5.1 Literature overview	106
4.5.2 Suggestions for further research	107
5 Noisy input Gaussian process regression	109
5.1 Using stochastic trial points	110
5.1.1 Integrating over possible trial points	110
5.1.2 Intermezzo: background behind moment matching	112
5.1.3 The expected value of the trial function value	113
5.1.4 The variance of the trial function value	114
5.1.5 The mean and variance of sparse predictions	116
5.1.6 Using multiple trial input points	117
5.2 Using stochastic measurement points	119
5.2.1 The problem behind integrating over measurement points	119
5.2.2 The noisy input Gaussian process regression algorithm	120
5.2.3 Sparse and online algorithms – the main ideas	121
5.2.4 The posterior distribution of the measurement point	122
5.2.5 Updating the distribution of the inducing function values	125
5.2.6 Derivatives needed for the SONIG algorithm	127
5.3 Extensions to the SONIG algorithm	128
5.3.1 Applying hyperparameter tuning	129
5.3.2 Using multiple outputs	129
5.3.3 The posterior distribution of the measured output	130
5.3.4 The posterior covariance between input and output	132

5.3.5	An online system identification algorithm	132
5.4	Experiments	135
5.4.1	Application to a test function	135
5.4.2	Identification of a magneto-rheological fluid damper	138
5.4.3	Noisy state measurements of the pitch-plunge system	140
5.5	Overview of literature	143
5.5.1	Literature overview	143
5.5.2	Suggestions for further research	144
6	Gaussian process optimization	147
6.1	Finding the maximum of a Gaussian process	148
6.1.1	The maximum distribution	148
6.1.2	An analytical approach to finding the maximum	150
6.1.3	A derivative approach to finding the maximum	151
6.1.4	A particle approach to finding the maximum	153
6.1.5	Finding the limit distribution of the particles	154
6.1.6	An intuitive view on the two different distributions	156
6.1.7	Using multiple challengers	157
6.2	Intermezzo: sequential Monte Carlo samplers	157
6.2.1	The idea behind Monte Carlo methods	159
6.2.2	Importance sampling	161
6.2.3	Self-normalized importance sampling	162
6.2.4	Sequential importance sampling	163
6.2.5	Resampling	164
6.2.6	Mixture importance sampling	166
6.2.7	Defensive importance sampling	167
6.3	Applying SMC ideas to find the maximum	168
6.3.1	Notations and definitions	168
6.3.2	Improving the convergence rate	169
6.3.3	Adding weights to particles	169
6.3.4	Resampling of particles	170
6.3.5	Ensuring correct convergence	170
6.3.6	Expanding the algorithm to continuous functions	171
6.3.7	The distribution of the maximum value	172
6.4	Gaussian process optimization	174
6.4.1	The Gaussian process optimization problem set-up	175
6.4.2	Basic GPO methods: acquisition functions	176
6.4.3	Intermezzo: the entropy of distributions	179
6.4.4	Entropy search	180
6.4.5	Portfolio methods	181
6.4.6	Thompson sampling	182
6.5	Experiments	183
6.5.1	Optimizing a one-dimensional function	183
6.5.2	Optimizing a two-dimensional function	185
6.5.3	A wind turbine simulation system	189
6.5.4	Multiple reference frames: the Coleman transformation	189

6.5.5	A quality criterion: the damage equivalent load	191
6.5.6	Optimizing the controller settings of a wind turbine	194
6.5.7	Applying the methods to a wind tunnel test	196
6.5.8	Lessons learned from the optimization experiments	198
6.6	Overview of literature and contributions	198
6.6.1	Literature overview	199
6.6.2	Suggestions for further research	200
7	Conclusion and recommendations	203
A	Matrix algebra	205
A.1	Matrix operations	206
A.1.1	The trace operator	206
A.1.2	Matrix derivatives	207
A.1.3	Vectorization and the Kronecker product	208
A.2	Matrix inverses	209
A.2.1	Blockwise matrix inverses	210
A.2.2	Inverting sums of matrices	212
A.3	Gaussian exponentials	214
A.3.1	Multiplying Gaussian exponentials	214
A.3.2	Multiplying/dividing Gaussian exponential functions	216
A.3.3	Joint Gaussian exponential functions	218
A.3.4	Other Gaussian exponential relations	219
A.4	Lyapunov equations	223
A.4.1	Notations and definitions	224
A.4.2	Finding the Lyapunov solution	225
A.4.3	Basic properties of Lyapunov solutions	226
A.4.4	Combinations of Lyapunov solutions	229
A.4.5	More integral expressions	231
A.5	Using matrix exponentials to solve integrals	234
A.5.1	Integrals within matrix exponentials	234
A.5.2	Using matrix exponentials to solve Lyapunov equations	240
A.5.3	A comparison between the two methods	241
A.6	Miscellaneous	243
B	Probability theory	245
B.1	Introducing the probability density function	246
B.1.1	Definition of the probability density function	246
B.1.2	Joint distributions	247
B.1.3	Conditional distributions	247
B.1.4	Special cases of the probability density function	248
B.2	The mean and the covariance	249
B.2.1	The fundamentals behind the mean and the covariance	249
B.2.2	Linear transformations of random variables	250
B.2.3	Further properties of the mean and the covariance	250

B.3	Transformations of probability density functions	252
B.3.1	Linear transformations of a random variable	252
B.3.2	Nonlinear transformations of random variables	253
B.3.3	Merging distributions	254
B.4	The Gaussian distribution	256
B.4.1	The Gaussian probability density function	257
B.4.2	The standard Gaussian distribution	257
B.4.3	Linear transformations of Gaussian distributions	259
B.4.4	Marginalization and conditioning of Gaussian distributions	260
B.4.5	Special cases of the Gaussian distribution	262
B.4.6	Power forms of Gaussian random variables	263
B.5	Manipulating Gaussian distributions	266
B.5.1	Merging Gaussian distributions	266
B.5.2	Measuring linear relations of Gaussian variables	270
B.5.3	Linear functions with Gaussian weights	272
B.6	Conditionally independent Gaussian variables	274
B.6.1	Conditional independence of random vectors	274
B.6.2	Conditional independence between vector elements	279
B.6.3	Conditional independence of parts of a vector	283
B.6.4	Online updating of distributions	284
C	Linear systems theory	289
C.1	Linear systems and their evolution	290
C.1.1	System definition	290
C.1.2	Evolution of the system state	291
C.2	The expected cost	296
C.2.1	The infinite-time cost function	296
C.2.2	The cost of a system without noise	297
C.2.3	The finite-time cost function	297
C.2.4	The discounted cost function	298
C.3	Linear quadratic Gaussian control	301
C.3.1	The input that optimizes the cost function	301
C.3.2	Differences for the discounted cost function	306
C.3.3	Reintroducing process noise	307
C.3.4	Estimating the state from noisy measurements	308
C.3.5	Optimal control based on the state estimate	310
C.4	The variance of the LQG cost	316
C.4.1	The infinite-time case	317
C.4.2	The non-discounted case	321
C.4.3	The general case	323
C.4.4	Solutions using matrix exponentials	325
C.5	Applications of the derived expression	329
C.5.1	A simulation verifying the derived equations	329
C.5.2	A simulation applying the derived equations	330
C.6	Overview of literature and contributions	332

References	333
Index	347

PREFACE

Summary — The main rule while reading this student version of my thesis is ‘Only read the parts that you are interested in.’ If something becomes boring, skip that part and go to the next section/chapter.

I have set up this student version mostly as an educational book, because I feel that this is the most effective way to contribute to science. It has been written with a different mindset than most scientific papers. I have not tried to minimize the word or page count, but rather the ‘science per cognitive load’, making the theory as intuitive as possible.

To accomplish this, each chapter focuses mainly on the ideas behind the theories, relegating mathematical derivations to the appendices. In addition, each chapter starts with a summary, and it is relatively easy to jump between chapters, especially after obtaining the intuitive view on Gaussian process regression explained in Sections 2.1, 2.2 and 2.4.1. Finally, all the source code for the examples and applications in this student version can be found online, ready for you to play around with. For this, see [Bijl \(2016a\)](#).

This student version of my Ph.D. thesis has been written with as main philosophy to save the reader time and energy. ‘How can that be? It is huge!’ you may be thinking. But it is overly long exactly because of this goal. All the information you may need is right here in this file, and more. And that is why the main rule while reading this thesis version is simple.

Only read the parts that you are interested in.

In this preface I will tell you more about why I set up the thesis in this way, as well as give you some tips and tricks on how best to read it. If you are interested, then read on. If not? Then skip it. You will not miss anything crucial. Just check out the [Table of Contents](#) and find something that sparks your imagination.

THE GOAL OF THIS STUDENT VERSION

‘The goal of a Ph.D. is to contribute to a certain scientific field,’ they told me when I started my project. I liked that. Having a clear goal is always a good thing.

Then they told me how to do that. ‘You have to publish at least three journal papers.’ And that’s where things went awry. Because the way I think highly educated people should be managed – or at least how I should be managed – is not by telling them *how* to do things. They’re smart enough to figure that out for themselves. You just tell them the *goals* they need to reach, and let them figure out the ideal way for them to reach those goals, of course with sufficient coaching. So my goal was still to contribute to science.

And quickly I realized that publishing papers did *not* feel like the best way to contribute. To me it felt like the scientific world was already flooded with papers. In fact, there were so many that I doubt anyone has a proper overview of the field anymore.

It all reminded me of some of my software projects. Sometimes these projects get so many additions from just as many different people, at different times, from different perspectives, that the whole project becomes a mess. It is nearly impossible to add stuff then, because no one knows how it affects other parts of the system, or how all functionalities relate to each other.

The solution there? A massive clean-up. A version 2.0. The best thing to do is to sort things out. To actually remove code. To simplify. To make notations and conventions uniform again. To create an overview. You don't actually build new stuff during that time, but the process itself is invaluable. Because after the clean-up, suddenly everything is easier and new contributions, even big ones which were previously unimaginable, often follow without much effort.

So what is different in the scientific world? Not much actually. I think the best thing we can do in almost every scientific field, at least for now, is not to add more papers. It is to sort things out. To get a uniform notation and language. To create an overview of what we know, allowing new people to easily enter the field. And to keep this overview updated with new developments as we go, though always keeping it structured. And that is exactly my goal for the student version of my thesis. It serves as an introduction into the wondrous world of Gaussian process regression. Of course it's far from a complete overview, but I hope it's a step in the right direction. And hopefully I can find the time to add updates every now and then to the online version, keeping track of developments in the field.

THOUGHTS BEHIND THE WRITING STYLE

Throughout the years I have learned the ideas behind Gaussian process regression, as well as various other mathematical techniques. In hindsight, learning these things took more time and energy than it should have. So while writing this student version, I have adopted a writing style whose goal is to save you time and energy while learning. However, the question ‘How can you save a reader time?’ is a hard one. How can I do this? And how does it work in general, in the scientific world?

Within scientific writing, there have been different writing mindsets throughout the years. In the old days, it was not about saving time at all. The focus was on minimizing the page count of a paper. The reason for this was obvious. With the peer review system, every paper had to be sent across the globe multiple times, resulting in significant costs. The less pages a paper had, the lower the costs. Hence the double column formats with tiny margins.

But with the advent of the digital era, this focus no longer makes sense. Yes, there are still many journals making use of page limits, although in my eyes any journal which desperately holds on to this criterion shows an ineptitude of using any of the other focuses we will discuss. If not for the very conservative and monopolized publishing world, I believe these journals would have become obsolete several years ago.

Luckily other journals are making advances. Some are for instance focusing on the word count and comparing this to the strength of the scientific contribution. This amount

of ‘science per word’ is then used as measure for the efficiency of the writing. This seems like a much better criterion. The word count is a measure of the time needed to read the paper. So focusing on this would mean that any reader would get as much ‘science per minute’ as possible. Right?

Well, partly. Though it is a significant improvement, I still find flaws in this philosophy. I have often read a paper in only thirty minutes, only to spend the rest of the day figuring out what the contents actually meant. I had to redo derivations (which were not fully written down), I had to track down a one-line proof from a nearly intractable reference (which could have also been incorporated into the paper) and I had to figure out the hidden assumptions made in the experiment section. Though the word count may be low in such a paper, a quantity that I call *cognitive load* is often very high.

Instead, I think scientific writing should focus on optimizing the ‘science per cognitive load’. The easier something is to read, the better you will understand it and also the faster you can read it through. The only downside is that the cognitive load is not something which can easily be measured by a computer program or (worse) a manager. It depends on the prior knowledge of the intended audience, estimating it requires expert knowledge, and still the measure remains rather subjective and personal. Nevertheless, the current peer review system already has an abundance of both expert knowledge and subjective judgments. So such a system should be possible to set up.

To give the right example, I have written this thesis with the same focus: reducing the cognitive load required to read it, while maximizing the amount of concepts conveyed. This mindset has a few consequences for how parts of the chapter are set up, which is what we will look at now.

THE SET-UP OF THE STUDENT VERSION AND ITS CHAPTERS

This student version of my thesis consists of a main body and appendices. In the main body, each chapter starts with a summary, follows up with theory and ends with an application and/or a literature overview. Let’s take a look at what the idea behind this whole subdivision is, and how the appendices fit into this.

THE MAIN BODY VERSUS THE APPENDICES

There is a very clear distinction between the main body and the appendices. The main body contains the intuitive explanation; so lots of pictures, clarifying viewpoints and the most important equations needed to implement the methods. However, the amount of derivations is strongly limited.

The large derivations and proofs for the theory have been put in the appendices. Naturally, when the main body brings up a theorem, I will refer to the appendix where the theorem resides, so you can easily look things up. And for the math fanatics out there, these appendices can also be read by themselves.

One thing which I do as little as possible is bring in theorems from external references. I know this is contrary to regular academic procedures, which always recommend referring to other papers for everything. But personally I think that asking a reader to look up a proof in an obscure and enormous reference, while it could have been included in a few lines, is a very bad habit. Of course I do attribute my sources, but I want to

prevent you from ever being stuck wondering ‘Where can I easily find more information about this?’

JUMPING BETWEEN CHAPTERS AND SECTIONS

All the chapters and most of the sections in this booklet can mostly be read independently. This makes it easier to look up things you want to learn more about, without first having to plow through numerous other texts. When prior knowledge is required, or when there are links to other subjects, this is indicated by referring to the appropriate sections. Sometimes there are also small intermezzos to cover any potentially missing prior knowledge.

However, all chapters do assume that you have some intuitive feeling of what a Gaussian process entails, and that you are familiar with the Gaussian process regression equation. If you are not familiar with GP regression yet, make sure to at least read Sections 2.1, 2.2 and 2.4.1. Once you have mastered this, you can jump to any chapter you like.

SUMMARIES

Every chapter, including this preface, starts with a summary. If you are a first-time reader of the subject, you should *not* read the summary though. So what is it for?

The summary is mostly for people already familiar with the subject, to quickly figure out what is in the chapter and check whether it matches their expectations. It is also useful to read after reading the chapter, to better memorize what you have learned.

After the summary you find an overview of what will be discussed in the chapter. This is useful if you want to know how the subjects within the chapter relate to each other, but it can also just as well be skipped.

MATLAB SCRIPTS AND APPLICATIONS

Each chapter contains a variety of plots showing the ideas discussed within that chapter. To make sure you can play around with the theory for yourself, all Matlab scripts used to generate the plots are available online. You can find them through [Bijl \(2016a\)](#). By playing around, you can gain more of an intuitive feeling for the theory than you can get from just reading explanations, so I would definitely encourage you to do so.

At the end of each chapter, I also apply the theory discussed in the chapter to a practical example. Each chapter uses a different set of examples and will introduce these itself. The source files for each of these applications are also available in the above repository. Wherever possible, the code has been set up such that each small experiment can be run by itself, so you can easily get started with any piece of code.

ON EQUATIONS AND PROOFS

This thesis is about a mathematical subject. Naturally there are a lot of equations. This especially holds for the appendices, where all the derivations and proofs reside. The goal of these derivations is that you can understand how we got to our results. You need to know what is going on. As such, I will explain every step that happens along the way.

I know that this is contrary to what most scientific articles do. I have read many papers in which the average ‘proof’ consisted of a single sentence: ‘Through ..., this is trivial.’ Often it took me more than a day to figure out what was so trivial. And sure,

after I understood it, it was not all that difficult. In fact, the most difficult thing was the process of obtaining that understanding. That was far from trivial.

I promise that you will not find a proof ‘This is trivial’ in this booklet. After all, if the matter really is trivial, I could explain it in a single sentence, and I should write that sentence instead. And if I could not explain it in a single sentence, then obviously it is not trivial and I should not say that it is.

When dealing with mathematical derivations, I usually add a separate line for every single step along the way. I have tuned the ‘cognitive size’ of these steps to what I think you can reasonably follow. At the start of the chapter each adjustment will be small, though as you get used to the theory, derivations will take bigger steps. So if a derivation does go too fast for you to follow, I recommend you to read earlier parts of the chapter. After doing so, I think you will be able to follow the derivation after all.

THE OVERVIEWS OF LITERATURE

The last section of most chapters is an overview of literature and contributions. This section mainly discusses existing literature on the subject. It can be seen as the *history* of the field. Who made which contribution at which time? It can also give suggestions for further reading. Because let’s face it: no single file can contain all available knowledge. I won’t pretend that this student version gives a complete overview of all GP knowledge.

You may wondering, ‘Should a literature overview not be at the start, as is normal in papers?’ But personally I think that set-up is only useful for papers (and still not for all papers). For educational resources, a literature overview functions more as background information for interested people. It does not convey understanding. In fact, the literature review often already uses terminology that is only introduced in the main body of the text, which makes it very hard to understand for readers new to the field.

Finally, these sections may outline suggestions for future research. Basically, they are questions which I am still curious about and would like to look into further, if only I had the time. They are also excellent starting points for master graduation projects. If you happen to look into these matters, do send me a note about your discoveries.

FINAL WORDS

I just want to point out that the goal of this student version is to teach as many people as possible the intricacies of Gaussian process regression. That works better if it reaches as many people as possible. So I encourage you to spread this thesis among anyone who might be interested in the subject.

Knowledge should be freely available, in a suitable format, to anyone who desires it.

*Hildo Bijl
Delft, August 2016*

1

INTRODUCTION

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, these flaps need to be controlled by an algorithm that can adapt itself. As such, we need a machine learning algorithm that can take stochastic effects into account.

Gaussian process regression has its basis in Bayesian probability theory, making it a very suitable technique. However, it also has its limitations. It cannot feasibly be applied to big and growing data sets, as well as to data with uncertainty in all its measurements. Efficiently optimizing a Gaussian process is also a complicated problem. The goal of this thesis is to solve these issues, and teach people how this is done, paving the way for future applications of Gaussian process regression.

This thesis is mainly about Gaussian process regression, with some applications in wind turbine technologies. You are probably reading this because you want to learn more about Gaussian process regression. In Section 1.1 we take a look, from a very global perspective, at what Gaussian process regression can do, as well as what its limitations are. Afterwards (Section 1.2) we examine why there are applications to wind turbines in this thesis. We end with an overview (Section 1.3) of the thesis: what can be found where?

1.1. ISSUES BEHIND GAUSSIAN PROCESS REGRESSION

We start off by looking at what *Gaussian Process regression* (GP regression) roughly comes down to (Section 1.1.1). When doing so, we also discover a few of the strengths of GP regression. Subsequently, we also consider its limitations (Section 1.1.2).

1.1.1. WHAT DOES GAUSSIAN PROCESS REGRESSION DO?

You may wonder, ‘What is regression?’ The idea of regression roughly comes down to predicting the value of certain variables, based on measurements of other variables. As such, it is useful when approximating some function $f(x)$. We measure the function value $f(x_m)$ at a few measurement input points x_m (sometimes called training points) and use those measurements, which may be corrupted by measurement noise, to predict the function value $f(x_*)$ for any trial (test) input point x_* . An example outcome is shown in Figure 1.1.

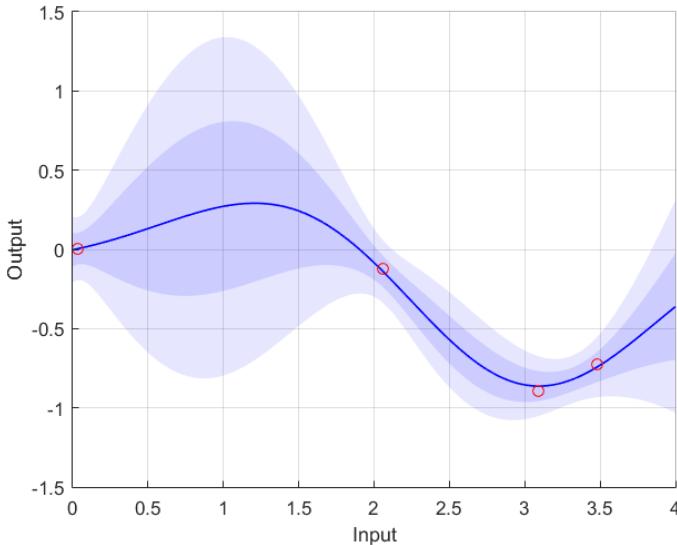


Figure 1.1: An example of Gaussian process regression, applied to a sinusoid. The circles denote noisy measurements, the line is the mean of the prediction, the dark region is one time the standard deviation of the prediction and the light region is two times the standard deviation. In regions with few measurements, the regression algorithm knows its predictions are more uncertain.

Many regression methods are already available. For linear functions, using the least-

squares method (see Lay et al. (2015), Björck (1996)) is often a good idea. For nonlinear functions things get a lot more complicated. The obvious but often suboptimal approach is to go for the nonlinear least-squares method (see Strutz (2016), Kelley (1995)). More promising methods are usually based on some sort of kernel or basis function. Examples include artificial neural networks (see Haykin (1999), Lawrence (1994)), support vector machines (see Steinwart and Christmann (2008), Campbell and Ying (2011)), splines (see Chapra and Canale (2015), Smith (1979)), and of course Gaussian process regression.

Mathematically, most of these methods are very similar: they all use some kind of basis functions. (See Section 2.3.3 for more information on this.) The philosophy and starting point of all these methods is very different though. Gaussian process regression uses Bayesian probability theory to make predictions. Because of this, its basis functions have a more intuitive meaning (see Section 2.2.1 for the basic idea or Section 3.2 for details), it has a built-in protection from overfitting (see the complexity penalty from Section 3.1.3) and when making actual predictions, GP regression does not only provide a mean estimate, but also a variance. This variance (see the colored area of Figure 1.1) is an indication of the uncertainty within the prediction, which can be very valuable data for certain applications.

These features all make Gaussian process regression a very powerful and useful regression method. But the question remains whether it can be applied to practical applications. It turns out that, while GP regression is a very promising technique, there are of course still various limitations.

1.1.2. LIMITATIONS OF GAUSSIAN PROCESS REGRESSION

There are several important questions we should ask ourselves, when we want Gaussian process regression techniques to be applied to practical applications, like wind turbines.

1. *How can GP regression be applied to a big and constantly growing data set?*

When GP regression is applied to a data set, a large covariance matrix needs to be inverted. This takes a lot of computational time. (Cubic in the number of measurements.) Then, when new data is added, this covariance matrix needs to be inverted all over again. This makes GP regression practically infeasible, unless methods can be found to work around this. We will look into this in Chapter 4.

2. *How can GP regression be applied subject to uncertainty in all measurements?*

GP regression has been set up with its basis in probability theory. So when the measurement of a function value $f(x_m)$ is corrupted by noise, this is not a significant problem at all. However, if the function input point x_m itself is corrupted by noise, we get a completely different story. Now the regular GP regression algorithm fails. Can we work around this? This is the subject of Chapter 5.

3. *How can Gaussian processes be optimized with respect to various parameters?*

GP regression is generally applied to approximate nonlinear functions, and finding the maximum (or equivalently the minimum) of a nonlinear function can be a daunting task. How does that work for GP regression? Can we find methods to effectively and efficiently optimize a Gaussian process? That is the main question of Chapter 6.

4. How can other people apply GP regression algorithms to practical applications?

If GP regression algorithms are to become more widespread within various industries, like the wind turbine industry, then it should be relatively easy to learn more about them and to start applying them. Otherwise the algorithms may exist, but no one has the skills to implement them. So is there an educational resource through which for instance master students and starting control engineers can learn more about GP regression and how it can be applied? That is an issue which this entire thesis is meant to tackle.

In general we can say that GP regression is a promising technique. However, before it can be widely applied, there are still some issues that need to be solved. The goal of the student version of this thesis is to help solve these issues; especially the latter one. Altogether, this should pave the way for future applications of Gaussian process regression.

1.2. WHY WIND TURBINE CONTROL?

Why does this thesis have applications to wind turbines? The easy answer here is, ‘That’s where the funding of my Ph.D. project came from.’ However, wind energy research is also important for our planet as a whole. After all, we need more renewable energy to keep our planet developing, without this development being hampered by serious problems like global warming, extreme pollution or running out of fossil fuels.

Luckily, most people agree that wind energy is important. What people don’t understand is why I would spend four years on studying wind turbine control. It seems ridiculous. ‘Why do we need to control a wind turbine in the first place? Just rotate it into the wind! How hard can that be?’

Well, things turned out to be a ‘bit’ more complicated than that. Initially I thought controlling an aircraft was hard. Compared to wind turbine control, it is actually a cakewalk¹. When controlling wind turbines, there are a lot more issues that are crucial to take into account. So let’s first look at what kind of issues there are concerning wind turbine control.

1.2.1. THE ISSUE OF COST

An important concept in the world of wind energy is the cost of wind energy. How much does it cost to produce 1 kWh? The key to a more sustainable world is to get this cost below the cost of producing 1 kWh of coal energy.

Some may argue that this is already the case. When producing coal energy, there will be side-effects. For example, the emissions of coal plants cause health issues and environmental issues, which the population in the end will pay for, in one way or another. Because the population effectively pays for the coal energy pollution, this can be seen as *hidden subsidies*. And when they are quantified and taken into account (see for instance Holland et al. (2011), Alberici et al. (2014)) then coal energy is generally already far more expensive than land-based wind energy and also slightly more expensive than offshore wind energy.

However, governments have so far proven to be incapable of properly taking these

¹To be honest, I have been wanting to use the word ‘cakewalk’ in my thesis for quite a while now.

hidden subsidies into account. To compensate, they provide financial subsidies to clean energies like wind energy, through a variety of programs. However, as a consequence of various financial crises, these subsidies are slowly being reduced. So the challenge for the wind energy community is to reduce the price of wind energy so much that, even when hidden subsidies are being ignored, wind energy is still cheaper than coal energy.

1.2.2. THE ISSUE OF SIZE

To reduce the cost of wind energy, wind turbines are gradually growing bigger. If we look at the size of new wind turbines that are brought to the market, with respect to the year they were introduced, then this trend is very obvious. Just take a look at Figure 1.2.

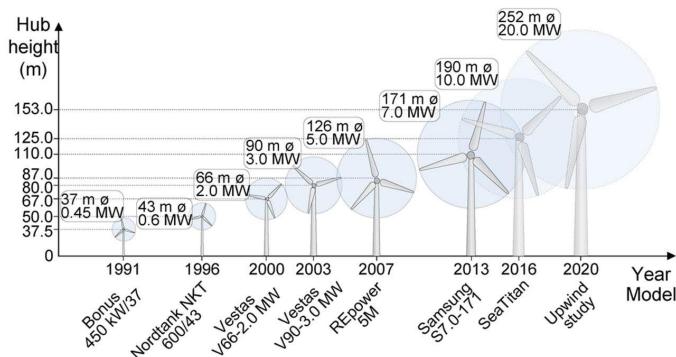


Figure 1.2: Development of the size of wind turbines over the years. The future sizes are a prognosis, whose validity strongly depends on whether the technical challenges related to larger wind turbines can be solved. Source: [Rodrigues et al. \(2016\)](#). Similar results are given by [Philibert and Holttinen \(2013\)](#).

The reason for this trend is related to scaling. The energy output of a wind turbine is roughly proportional to the (circular) area of the actuator disc; so the area spanned by the wind turbine blades. (See for instance [Wagner and Mathur \(2013\)](#), [Bianchi et al. \(2007\)](#), also for further background on wind turbine technology.) This means that, if the rotor diameter becomes twice as big, then the energy produced by the wind turbine becomes four times as big.

The costs generally scale differently. A bigger wind turbine still only needs one generator, albeit one with a somewhat larger power range. It still needs one cable to transport the produced energy. For land-based turbines it still requires one location to be placed at, or for offshore wind turbines one foundation to be installed on. As such, bigger wind turbines are generally more cost-efficient.

1.2.3. THE ISSUE OF VIBRATIONS

There are limits to how big a wind turbine can become though, and these limits are mainly present due to the blades.

For aerodynamic reasons, these blades are very long and thin. And anyone who has watched a piece of barrier tape hanging in the wind knows what happens with such long and slender objects: it starts to *flutter*. Although this phenomenon is already present in airplane wings, it occurs more in large wind turbines, whose blades are more than twice

as long as the largest airplane wings being used today. This flutter behavior is the result of a variety of factors, among which the current wind speed and how turbulent the wind actually is. This makes it very hard to predict.

But flutter is only a small part of the problem. Another problem occurs because the wind speed varies across the turbine. At high altitudes the wind speed is generally higher than at lower altitudes; a phenomenon known as *wind shear*. (See Figure 1.3.) In addition, near the tower of the wind turbine the wind is mostly stopped. This so-called *tower shadow* causes the wind speed near the tower to be roughly zero. These large variations in wind speed which a blade encounters during a full rotation also cause vibrations within the turbine blade.

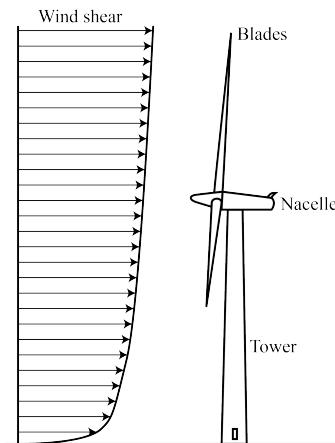


Figure 1.3: A wind turbine with the incoming wind visualized. Because the wind speed is lower on lower altitudes, the turbine experiences wind shear. In addition, because the tower blocks the incoming wind, tower shadow also occurs.

Added to this are still more vibrations, because the wind field itself is also constantly fluctuating, both in speed and direction. 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.

All these vibrations have various effects. First of all, they cause fatigue damage within the blades. The bigger the blade, the more severe this damage will be. But the vibrations also find their way through to the generator, which also takes damage. And in addition, these vibrations also affect the energy that is produced. It is not (significantly) the case that vibrations cause less energy to be produced, but they do cause the energy to be more fluctuating, and hence of less value.

You may be wondering ‘Why don’t airplanes have this vibration problem?’ Well, they do. However, for some reason airplanes often pass by airports, where they get regular inspections and maintenance. Wind turbines are usually in remote locations, making their inspections and maintenance rather expensive. That is why wind turbines need to be built to last.

Currently, most wind turbines have a lifetime of roughly fifteen to twenty years. If the blades become bigger, and the vibrations hence become worse, then this lifetime

will decrease, making the turbine economically less feasible. The conclusion of all this is obvious. When wind turbines need to grow bigger, the vibrations must be reduced.

1.2.4. THE ISSUE OF FREQUENCIES

To reduce the vibrations, we need some way to control the blades. Luckily, there are various methods of doing so.

The first is through the *pitch angle* of the blade. A pitch angle of 0° here means the blades of the turbine are all in the same plane, while a pitch angle of 90° means the blades are all pointing forward. In practice, the pitch angles are always small. You only set the pitch angle of the turbine to 90° (called *feathering* of the turbine) when a storm is coming and you want the turbine to ignore the wind.

In traditional wind turbines, the pitch angles of all the blades are equal. This is an easy way to control the turbine. However, it is also possible to vary the pitch angle of each individual blade. This technique is called *Individual Pitch Control* (IPC) and it is gradually starting to be applied in modern wind turbines. Research has shown (see for instance [Selvam et al. \(2008\)](#)) that it is possible to reduce the fatigue loads in this way.

The main problem with IPC is that it is relatively slow. It takes some time (in the order of a full second) to adjust the pitch angle of a very long blade. So while this technique can be useful to reduce the more predictable low-frequency vibrations, it cannot be used to get rid of less predictable high-frequency vibrations.

Instead, we can also install flaps on the trailing edge of the blades, as shown in Figure 1.4. These flaps can then be actuated using small motors (see [Berg et al. \(2012\)](#)), using shape memory alloys (see [Hulskamp \(2011\)](#)) or using piezo-actuators (see [Bak et al. \(2007\)](#), [Hulskamp \(2011\)](#)). For a complete overview of control methods, see [Pechli-vanogloou \(2012\)](#).

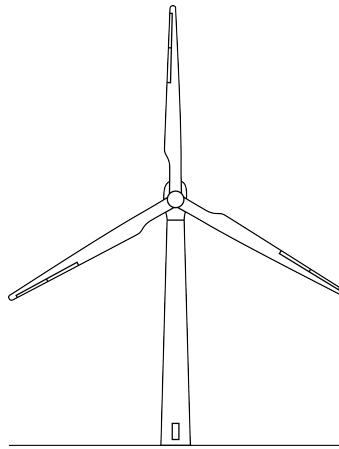


Figure 1.4: A wind turbine with trailing-edge flaps attached to the blades. Contrary to individual pitch control, these flaps are capable of reducing relatively high-frequency oscillations.

As input data, we can use a variety of data, including pitot tubes to measure air pressure (see [Heinz et al. \(2010\)](#)), strain gauges to measure moments (see [Castaignet et al.](#)

(2012)), accelerometers to measure motion, or LiDAR technology to measure the incoming airflow (see [Pechlivanoglou \(2012\)](#)). Through this data, we can then actuate the flaps.

These flaps can react much faster than any IPC method. So with these flaps it should, in theory, be possible to also reduce the high-frequency vibrations. In fact, it has already been shown (see for instance [Castaignet et al. \(2012\)](#), [Andersen et al. \(2006\)](#), [Basualdo \(2005\)](#), [Buhl et al. \(2005\)](#)) that this method can significantly reduce the vibration load. The main question that remains is: what is the best way to control these flaps? How do we determine how much the flaps should deflect at each point in time? Or in other words, what is the optimal control algorithm?

1.2.5. THE ISSUE OF CONTROL

In the field of control theory, the common way to develop a control algorithm for some kind of system, is to first set up a model of the system. Once we have such a model, and we know how everything works, we can set up an algorithm to control the system.

For our problem, this approach does not work very well. The first problem is that the dynamics of the wind turbine strongly depend on one very important external parameter: the wind speed. To take this into account, we will have to look into parameter-varying control techniques. Specifically, *Linear Parameter-Varying control* (LPV control) has seen significant developments in recent years. See for instance [Tóth \(2010\)](#), [Briat \(2015\)](#) for general theory, or [van Wingerden \(2008\)](#), [Adegas and Stoustrup \(2012\)](#) for applications to wind turbines.

However, LPV methods cannot solve all our problems here. This is partly because turbulence and flutter are inherently hard to predict. But also the more predictable oscillations – the ones due to wind shear and tower shadow – are hard to get rid of with only a single LPV controller. The reason is that, while manufacturing the wind turbine and its blades, there are always minor variations in the manufacturing process. Every wind turbine is inherently different. And this difference is also likely to grow as the wind turbine ages. Added to this, the dynamics of the turbine are also made significantly more complicated due to the addition of flaps. So devising any single controller that works optimally for every wind turbine, even with robust control techniques as described by [Skogestad and Postlethwaite \(2005\)](#), [Green and Limebeer \(1995\)](#), is pretty much impossible. The system uncertainties are too large.

What we need instead is a control algorithm that adapts to the wind turbine it is running on. We need some kind of data-based approach. An algorithm that learns by itself how to reduce vibrations. This method should be able to take into account uncertainties which are inherently the result of a stochastic wind field.

Ideally, such a control algorithm should take into account the actual cost of energy. It should be able to trade off the value of more energy being generated to the damage that is caused to the turbine. Naturally, if energy had no value, we would simply let the wind turbine stand still, giving us the least amount of damage. And on the other hand, when the damage to the turbine was irrelevant, vibrations would be irrelevant and we would simply maximize the generated power. We need a control algorithm that can find the optimum between these two extremes.

The problem we are facing here is that the damage calculations are inherently non-linear (see Section [6.5.5](#) for more background on this) while most control algorithms use

linear or quadratic cost functions. Because of this, we need a technique that can approximate nonlinear cost functions.

To summarize, we need a learning algorithm which can approximate nonlinear cost functions subject to noise and uncertainties. The ideal method here seems to be Gaussian process regression. This method can approximate a variety of functions, including nonlinear functions, and because it has its basis in Bayesian probability theory, it inherently takes uncertainty into account. The question remains whether this machine learning technique is sufficiently developed to be applied to wind turbine problems. Answering that question has resulted in this thesis.

1.3. OVERVIEW OF THIS BOOKLET

In this thesis we start off with a basic introduction into Gaussian process regression (Chapter 2). If you are new to GP regression, I most certainly recommend you to read Sections 2.1, 2.2 and 2.4.1. The other sections are also useful but less critical for understanding the other chapters.

Afterwards, you can jump to any chapter you like. Every section mentions it whenever you require certain prior knowledge. Chapter 3 will look at already existing but somewhat more advanced tricks behind GP regression. In chapter 4 we examine how we can apply Gaussian process regression to large data sets without having to wait an eternity. We also consider how we can efficiently incorporate new measurements as they come. Chapter 5 then discusses what we can do when we run into input points that are subject to measurement noise. In Chapter 6 we figure out how to optimize a Gaussian process. Finally, we conclude the thesis in chapter 7 by looking at whether we have overcome the four mentioned limitations of GP regression.

Also worth mentioning are the appendices, which contain all the mathematics behind this thesis. Appendix A contains several theorems on Matrix algebra, ranging from various matrix operations up to solutions of Lyapunov equations and their properties. Appendix B discusses probability theory, starting from the basic definitions of the probability density function, and working up to incorporating new measurements into conditionally independent Gaussian distributions. Finally, Appendix C goes into depth on linear systems theory, discussing how to optimally control a linear system subject to Gaussian noise, even when the quadratic cost function is discounted. It also gives expressions to calculate the mean and variance of the resulting cost.

2

AN INTUITIVE INTRODUCTION TO GAUSSIAN PROCESS REGRESSION

Summary — The main idea behind GP regression is to consider function values $f(x)$ as Gaussian random variables. To be precise, any set of function values $f(x_1), f(x_2), \dots$ is assumed to have a joint Gaussian distribution. For the input points x , we can include measurement points x_m (where we will do measurements) and trial points x_ (where we want to predict the function value).*

When applying GP regression, we first need to define the prior distribution of the vector of function values f . We do this by defining a mean and a covariance function. Then we start doing measurements. These measurements give us additional distributions over the function values f . By merging all available distributions together, we get our predictions for the trial function values f_ .*

Intuitively, a Gaussian process can be seen as a distribution over functions. And by taking the derivative, we wind up with another Gaussian process. It is also possible to take into account multi-input functions and multi-output functions, though in the latter case it is often more efficient to approximate each function output by a separate Gaussian process.

In this chapter we examine the basics behind Gaussian process regression from a very intuitive point of view. It serves as a first introduction to the field. The theory here is not new – it is already outlined by [Rasmussen and Williams \(2006\)](#) – but the method with which we derive our equations is new. Those already familiar with Gaussian process regression can skip this chapter altogether, or quickly browse through Sections 2.1 and 2.2 to check out the notation and method that we are using.

We will start this chapter by looking at a simple problem: approximating a single variable (Section 2.1). These exact same concepts are then used for regression to approximate functions (Section 2.2). That is where we suddenly wind up with the Gaussian process regression equations. Next, we check out various ways of looking at Gaussian processes, from the official definition to a more intuitive view (Section 2.3). Afterwards we look at a few extensions of our methods, from multi-input and multi-output functions (Section 2.4) to derivatives and integrals of functions (Section 2.5). Finally, we apply the ideas that we have learned in an application (Section 2.6). At the back end of this chapter, a short literature overview is given (Section 2.7).

If you are short on time, then I would recommend to read only Sections 2.1, 2.2 and 2.4.1. These form the basis of the theory and will be crucial for the rest of this thesis. The rest of this chapter provides interesting but not crucial background information and extensions.

2.1. APPROXIMATING A VARIABLE

In this section we will not look at Gaussian process regression just yet. We will look at a simple case of approximating a single variable. This simple case teaches us a few basic concepts which will be essential in Section 2.2, when we do look at GP regression.

We will start off (Section 2.1.1) with the concept of a prior distribution of a variable. Then we look at what a measurement of that variable will tell us (Section 2.1.2) and how to incorporate that data through a concept called merging distributions (Section 2.1.3). We then expand this to the case where we have multiple variables (Section 2.1.4) and what we can do when we only measure one of them (Section 2.1.5).

2.1.1. THE PRIOR DISTRIBUTION

Suppose that we have some number f . We want to know what it is, but at the moment we do not know. We expect it to be roughly zero, but it is likely to be anywhere between -2 and 2 , although it might actually also fall outside of that range in a few special cases.

The way we treat this mathematically, is that we treat f as a random variable. We hence write it as \underline{f} . This random variable has a certain distribution. Based on our *prior knowledge* of the number – knowledge that we have without doing any measurements – we can for instance say that \underline{f} is a Gaussian random variable with mean $m = 0$ and standard deviation¹ $\lambda_f = 2$. (For further details about the Gaussian distribution, see Section B.4.) Mathematically, we then say that the *prior distribution* of f , expressed through the *Probability Density Function* (PDF; for details, see Section B.1) equals

$$\underline{f} \sim \mathcal{N}(f|m, \lambda_f^2). \quad (2.1)$$

¹In literature, often the symbol α or σ_f is used here instead of λ_f . We will use λ_f here to indicate that it is a length scale related to the parameter f .

The \sim sign officially means ‘has as probability density function’ though is more often read as ‘is distributed according to’. The function $\mathcal{N}(f|m, \lambda_f^2)$ is the (one-dimensional) *Gaussian probability density function* with mean m and variance λ_f^2 . It per definition equals

$$\mathcal{N}(f|m, \lambda_f^2) \equiv \frac{1}{\sqrt{2\pi\lambda_f^2}} \exp\left(-\frac{1}{2} \frac{(f-m)^2}{\lambda_f^2}\right). \quad (2.2)$$

Note that the first parameter f in the function definition $\mathcal{N}(f|m, \lambda_f^2)$ is only a running variable in the Gaussian PDF. We could have used any symbol here. Often we simply omit it and write $\mathcal{N}(m, \lambda_f^2)$, which means the same. The parameter λ_f in the above function definition can be seen as a *length scale* for f . The larger it is, the larger the range of values which f will take.

2.1.2. MAKING MEASUREMENTS OF f

To learn more about the exact value of f , we will do measurements. If we have an infinitely precise measurement, which tells us that f equals some number² \hat{f} , then we would be done. We would exactly know f .

But in the real world this is almost never the case. There is always some *measurement noise*. As a result, we would only get a *measured value* \hat{f} , which is different from the *true value* f due to the noise. Specifically, if we know the noise v , we have

$$\hat{f} = f + v. \quad (2.3)$$

Before we do the measurement though, we can still see the measured value \hat{f} as a random variable $\underline{\hat{f}}$, and then we still have

$$\underline{\hat{f}} = \underline{f} + \underline{v}. \quad (2.4)$$

We generally assume that we are dealing with Gaussian white noise. That is, v has a zero-mean Gaussian distribution with variance³ $\hat{\sigma}_f^2$,

$$\underline{v} \sim \mathcal{N}(0|0, \hat{\sigma}_f^2), \quad (2.5)$$

and its exact value does not depend on the measurement noise of any other measurement which we might do. As a result, prior to doing our measurement, we have

$$\underline{\hat{f}} \sim \mathcal{N}(\hat{f}|m+0, \lambda_f^2 + \hat{\sigma}_f^2). \quad (2.6)$$

(See Theorem B.4 to learn how to find the mean/variance of the sum of random variables.) Next, when we perform a measurement, we get to know \hat{f} deterministically.

²In literature measurements are often denoted by y . However, later on (Chapter 5) we will also start measuring the input x , in which case this notation reaches its limits. That is why, in this thesis, we will denote measurements through the hat-notation \hat{f} .

³In literature, often the symbol σ_n is used instead of $\hat{\sigma}_f$. In addition, literature often uses σ_f as output length scale – where we use λ_f . This results in a lot of ambiguity. In this thesis I will consistently use λ for length scales and σ for noise scales, and I will use the hat-notation to indicate measurements or parameters indicating properties of measurements.

However, we do not know the measurement noise v that was involved. As a result, we have

$$\underline{f} = \hat{f} - v \sim \mathcal{N}\left(f|\hat{f}, \hat{\sigma}_f^2\right). \quad (2.7)$$

So this measurement has actually given us another distribution for f . And now we have two distributions for f , being (2.1) and (2.7). Both have been obtained independently and both are correct. We need to find a way to merge them.

2.1.3. MERGING DISTRIBUTIONS

So how likely is it that our random variable \underline{f} actually equals some value f ? This probability depends on the two distributions that we have. Specifically, a possible value f should correspond to the first distribution *and* the second distribution. This word ‘and’ means we need to *multiply* the two distributions. And to make sure that we get an actual probability density, we also need to normalize the result. So,

$$\underline{f} \sim \frac{\mathcal{N}(f|m, \lambda_f^2) \mathcal{N}(f|\hat{f}, \hat{\sigma}_f^2)}{\int_{-\infty}^{\infty} \mathcal{N}(f|m, \lambda_f^2) \mathcal{N}(f|\hat{f}, \hat{\sigma}_f^2) df}. \quad (2.8)$$

Note that this reasoning is not an official mathematical proof. It is the intuitive point of view. For the official proof, see Theorem B.9.

This new distribution of \underline{f} takes all measurements into account. As such, we call it the *posterior distribution*. An example of calculating this posterior distribution can be seen in Figure 2.1. The source code of this example (and of every other example) can also be found online through [Bijl \(2016a\)](#).

Of course, if we had performed multiple measurements $\hat{f}_1, \hat{f}_2, \dots$, this expression would have been bigger. We would have wound up (see Theorem B.10) with a posterior distribution

$$\underline{f} \sim \frac{\mathcal{N}(f|m, \lambda_f^2) \mathcal{N}(f|\hat{f}_1, \hat{\sigma}_{f_1}^2) \mathcal{N}(f|\hat{f}_2, \hat{\sigma}_{f_2}^2) \dots}{\int_{-\infty}^{\infty} \mathcal{N}(f|m, \lambda_f^2) \mathcal{N}(f|\hat{f}_1, \hat{\sigma}_{f_1}^2) \mathcal{N}(f|\hat{f}_2, \hat{\sigma}_{f_2}^2) \dots df}. \quad (2.9)$$

It is worthwhile to note that in practice the measurement noise variance is often the same between measurements. So then $\hat{\sigma}_{f_1} = \hat{\sigma}_{f_2} = \dots$

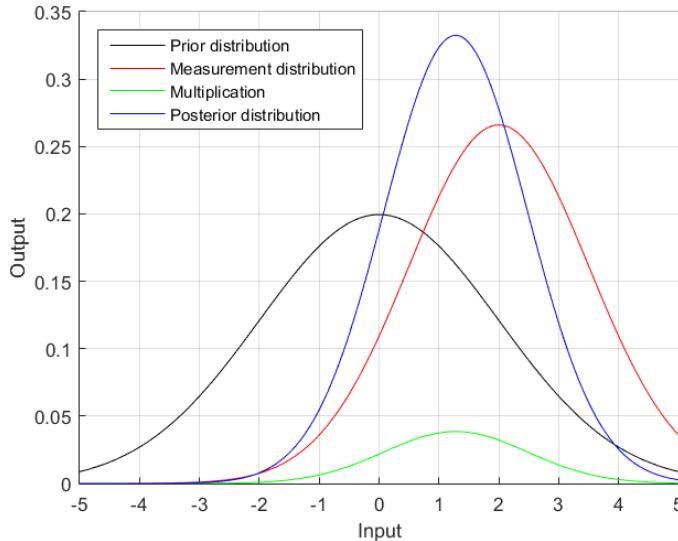
Finally, because we will see this idea of merging distributions more often, we introduce a shorthand notation for it. We use the operator \oplus and say that (2.9) is (per definition) equivalent to

$$\underline{f} \sim \mathcal{N}(f|m, \lambda_f^2) \oplus \mathcal{N}(f|\hat{f}_1, \hat{\sigma}_{f_1}^2) \oplus \mathcal{N}(f|\hat{f}_2, \hat{\sigma}_{f_2}^2) \oplus \dots \quad (2.10)$$

For more details about this, see Appendix B.3.3 (merging distributions in general) or Appendix B.5.1 (merging Gaussian distributions).

2.1.4. MULTIPLE VARIABLES TO APPROXIMATE

Now suppose that we have multiple variables f_1, f_2 and f_3 that we want to find. We can put them all into a vector \mathbf{f} , where the bold face indicates that \mathbf{f} is a vector. In this case,



2

Figure 2.1: An example of merging distributions. The prior has distribution $\mathcal{N}(0, 2^2)$ and the measurement has distribution $\mathcal{N}(2, 1.5^2)$. To find the posterior (merged) distribution, we can multiply the probabilities and subsequently normalize the result. This result is always more peaked than any of the original distributions. Note that the source code behind all the plots in this thesis is online and can be found through [Bijl \(2016a\)](#).

we can treat \underline{f} as a *random vector* \underline{f} . Our prior distribution is now written as

$$\underline{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \mid \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}, \begin{bmatrix} \lambda_{f_1}^2 & 0 & 0 \\ 0 & \lambda_{f_2}^2 & 0 \\ 0 & 0 & \lambda_{f_3}^2 \end{bmatrix} \right) = \mathcal{N}(\underline{f} | \underline{m}, K), \quad (2.11)$$

where \underline{m} is the prior mean and K is the prior covariance matrix. (Why we have chosen the symbol K here will become clear in Section 2.2.) Note that here we're using a multivariate distribution, and as such we should also use the *multivariate Gaussian distribution*

$$\mathcal{N}(\underline{f} | \boldsymbol{\mu}, \Sigma) \equiv \frac{1}{\sqrt{|2\pi\Sigma|}} \exp \left(-\frac{1}{2} (\underline{f} - \boldsymbol{\mu})^T \Sigma^{-1} (\underline{f} - \boldsymbol{\mu}) \right). \quad (2.12)$$

This is in fact a multi-dimensional generalization of (2.2).

We should also update the notation for our measurements. Suppose that we have measured values $\hat{f}_1, \hat{f}_2, \dots$. We can write any measurement \hat{f}_i , with i our measurement index, as

$$\underline{f} \sim \mathcal{N}(\underline{f} | \hat{f}_i, \hat{\Sigma}_{f_i}). \quad (2.13)$$

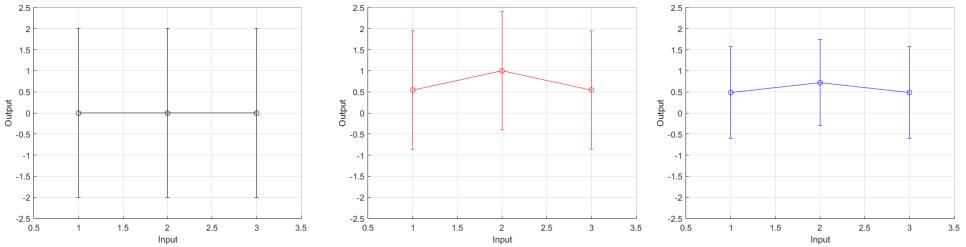


Figure 2.2: An example of merging multi-dimensional distributions. The left figure shows the prior distribution $\mathcal{N}(0, 1^2)$ for three measurement points. The middle figure shows the measurement distribution $\mathcal{N}(\hat{f}, 0.7^2)$, for varying \hat{f} . The right figure shows the merged (posterior) distribution. Error bars have the size of two times the standard deviation, to indicate the 95% region.

Now we have as posterior distribution,

$$\begin{aligned}\underline{\mathbf{f}} &\sim \frac{\mathcal{N}(\mathbf{f}|\mathbf{m}, K) \mathcal{N}(\mathbf{f}|\hat{\mathbf{f}}_1, \hat{\Sigma}_{\mathbf{f}_1}) \mathcal{N}(\mathbf{f}|\hat{\mathbf{f}}_2, \hat{\Sigma}_{\mathbf{f}_2}) \dots}{\int_F \mathcal{N}(\mathbf{f}|\mathbf{m}, K) \mathcal{N}(\mathbf{f}|\hat{\mathbf{f}}_1, \hat{\Sigma}_{\mathbf{f}_1}) \mathcal{N}(\mathbf{f}|\hat{\mathbf{f}}_2, \hat{\Sigma}_{\mathbf{f}_2}) \dots d\mathbf{f}} \\ &= \mathcal{N}(\mathbf{f}|\mathbf{m}, K) \oplus \mathcal{N}(\mathbf{f}|\hat{\mathbf{f}}_1, \hat{\Sigma}_{\mathbf{f}_1}) \oplus \mathcal{N}(\mathbf{f}|\hat{\mathbf{f}}_2, \hat{\Sigma}_{\mathbf{f}_2}) \oplus \dots\end{aligned}\quad (2.14)$$

where the integral is taken over F , which is the space of all possible values of \mathbf{f} . An example is shown in Figure 2.2.

The great thing is that the resulting distribution will again be Gaussian. The posterior distribution (see Theorem B.21) will satisfy

$$\begin{aligned}\underline{\mathbf{f}} &\sim \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \Sigma), \\ \Sigma &= \left(K^{-1} + \hat{\Sigma}_{\mathbf{f}_1}^{-1} + \hat{\Sigma}_{\mathbf{f}_2}^{-1} + \dots \right)^{-1}, \\ \boldsymbol{\mu} &= \Sigma \left(K^{-1} \mathbf{m} + \hat{\Sigma}_{\mathbf{f}_1}^{-1} \hat{\mathbf{f}}_1 + \hat{\Sigma}_{\mathbf{f}_2}^{-1} \hat{\mathbf{f}}_2 + \dots \right).\end{aligned}\quad (2.15)$$

We write the expression for Σ here before the expression for $\boldsymbol{\mu}$, because we often use Σ within the expression for $\boldsymbol{\mu}$. It is interesting to know here that (apart from a few special cases discussed in Appendix B.4.5) covariance matrices K and Σ are always positive definite. This means that, the more measurements are added, the smaller Σ will get. (At least, its determinant, being the product of the eigenvalues, will become smaller.) And a smaller variance means a more accurate estimate. Adding more data hence means you will get estimates which are more accurate. This does not always mean your estimates become closer to the true value of what you are estimating. It is always possible to get a single instance of bad noise. But adding an additional measurement is (on average) *expected* to bring your estimate closer to the true value, which always makes it worthwhile.

2.1.5. MEASURING ONLY A SINGLE VARIABLE

In the previous section, we have always measured the full vector \mathbf{f} . But what should we do if we only measure a single variable f_1 ? It turns out that we can then still use (2.15).

Our measurement basically gives us a distribution $\underline{f}_1 \sim \mathcal{N}(\underline{f}_1 | \hat{f}_1, \sigma_{f_1}^2)$. The measurement hasn't told us anything about \underline{f}_2 or \underline{f}_3 though. And when we know absolutely nothing about a Gaussianly distributed random variable – it can literally be anything – we also say that this random variable has an infinitely large variance. (See Section B.4.5 for further background on this.) It follows that we can also write our measured distribution as

$$\underline{f} \sim \mathcal{N}\left(\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \middle| \begin{bmatrix} \hat{f}_1 \\ * \\ * \end{bmatrix}, \begin{bmatrix} \sigma_{f_1}^2 & * & * \\ * & \infty & * \\ * & * & \infty \end{bmatrix}\right). \quad (2.16)$$

So what do the stars $*$ here mean? Simply put, they are inconsequential values⁴. We could have put numbers in place of these stars. But when we plug the above into (2.15), then these numbers would have dropped out of the equations, because of the presence of the infinities.

Next, we can apply the above distribution to (2.15). When we work out the mathematics, we do have to apply Theorem A.11 to take into account the infinite variances, but everything will work out in the usual way. So this is how we can take into account a measurement of only part of the vector f .

2.2. APPROXIMATING VARIABLES WE HAVE NOT MEASURED

So far, we haven't really talked about *regression* yet. After all, regression is about making predictions of parameters we have never performed any measurements on. How that works is something we will look into in this section, where we will actually examine the approximation of a function $f(x)$.

We will first review our prior assumptions (Section 2.2.1), before we use these new assumptions to predict function values (Section 2.2.2). Then, after adjusting our notation a bit (Section 2.2.3) we also implement measurement noise (Section 2.2.4).

2.2.1. MAKING PRIOR ASSUMPTIONS ON FUNCTION VALUES

Suppose that we have some function $f(x)$ and we want to estimate or approximate the value of this function at various input points x_1 , x_2 and x_3 . (For instance, take $x_1 = 1$, $x_2 = 2$ and $x_3 = 3$.) So we want to know the function values $f(x_1)$, $f(x_2)$ and $f(x_3)$, which we shorten, for ease of writing, to f_1 , f_2 and f_3 . How do we do this?

Well, we again have three numbers that we want to know, so just like in the previous section we assume that they are random variables. We write them as \underline{f}_1 , \underline{f}_2 and \underline{f}_3 , and we assume that they each have some prior mean $m(x)$ and variance $\lambda_f(x)^2$. Note that these may now also depend on the function input x .

The problem now is that, if we now know something about \underline{f}_1 , we still cannot say anything about \underline{f}_2 or \underline{f}_3 . Without further assumptions, these function values are completely unrelated, and we cannot apply any regression. Mathematically, this is because

⁴In literature people often try to avoid working with infinity. To accomplish this, they work with the inverse of the covariance matrix, which is known as the *precision matrix* or the *information matrix*. An infinite covariance hence means zero precision/information, a zero covariance means infinitely precise information and (as is explained right after Theorem B.23) a negative covariance means negative information or disinformation. To keep things simple, we will stick with the above notation and only use the covariance matrix.

the matrix K in (2.11) is a diagonal matrix. So we have to assume some kind of link between these function values.

We could for instance assume that the original function $f(x)$ is smooth and doesn't vary too much with varying x . From a probabilistic point of view this means that, because x_1 is close to x_2 , \underline{f}_1 has a similar value as \underline{f}_2 , but because x_1 and x_3 are further apart, we can say less about \underline{f}_3 . The question now is 'How do we express this mathematically?'

We should keep in mind here that we have assumed that these function values $f(x)$ are random variables. And random variables can be correlated! In fact, we could assume that \underline{f}_1 and \underline{f}_2 are strongly correlated, while \underline{f}_1 and \underline{f}_3 are less strongly correlated. To do so, we define a *correlation function* $c(x, x')$ which defines the (a priori) correlation between the function values $f(x)$ and $f(x')$ for any input points x and x' . For instance, we could use a *Squared Exponential correlation function* (SE correlation function)

$$c(x, x') = \exp\left(-\frac{1}{2} \frac{(x - x')^2}{\lambda_x^2}\right). \quad (2.17)$$

Here⁵ λ_x is a length scale for the input x . The correlation function for $\lambda_x = 1$ is shown in Figure 2.3. It basically shows that the function values $\underline{f}(x)$ and $\underline{f}(x')$ of nearby input points x and x' are strongly correlated, while the function values for input points that are very far away from each other have a nearly zero correlation.

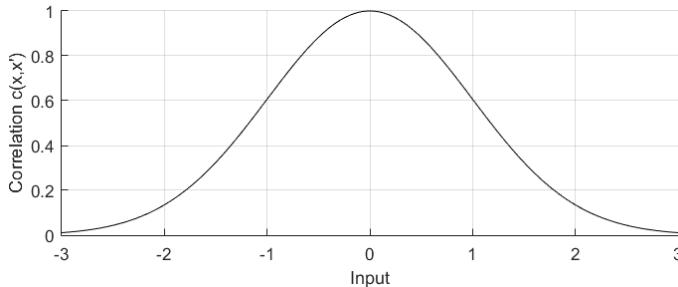


Figure 2.3: The correlation function (2.17) for $\lambda_x = 1$. For other values of λ_x , just scale the function horizontally by a factor λ_x .

In practice, we actually don't use a correlation function but we use something very similar. We use a *covariance function*⁶ $k(x, x')$, which follows as

$$k(x, x') = \lambda_f(x)\lambda_f(x')c(x, x'). \quad (2.18)$$

If we use a constant standard deviation $\lambda_f(x) = \lambda_f$, then we wind up with the well-known

⁵Other literature generally uses the notation λ without subscript here. But we use the subscript so as not to confuse it with the output scaling parameter λ_f .

⁶In literature the covariance function is sometimes also called the *kernel function*. This is because the covariance function performs a very similar role to the kernel function in kernel methods like support vector machines.

Squared Exponential covariance function (SE covariance function)

$$k(x, x') = \lambda_f^2 \exp\left(-\frac{1}{2} \frac{(x - x')^2}{\lambda_x^2}\right). \quad (2.19)$$

Note that the covariance of $\underline{f}(x)$ with itself, for any input point x , is now given by $k(x, x) = \lambda_f^2$, which is exactly what we had already assumed in Section 2.2.1.

Using our assumptions so far, we can set up our prior distribution. It is the (in this case) three-dimensional Gaussian distribution

$$\begin{aligned} \underline{\mathbf{f}} = \underline{\mathbf{f}}(X) &= \begin{bmatrix} \underline{f}(x_1) \\ \underline{f}(x_2) \\ \underline{f}(x_3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \mid \begin{bmatrix} m(x_1) \\ m(x_2) \\ m(x_3) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) \end{bmatrix}\right) \quad (2.20) \\ &= \mathcal{N}(\underline{\mathbf{f}} | m(X), k(X, X)) \\ &= \mathcal{N}(\underline{\mathbf{f}} | \mathbf{m}, K). \end{aligned}$$

It is important to note the difference with (2.11). Now the matrix K is not diagonal anymore, but there are covariances in it. In other words, the function values are linked. Knowing something about one will tell us more about the others, which is what regression is all about.

2.2.2. MAKING PREDICTIONS ABOUT OTHER FUNCTION VALUES

Next, suppose that we have measured (deterministically – without noise) that $\underline{f}_1 = \hat{f}_1$. What does this tell us about the distribution of \underline{f}_2 and \underline{f}_3 ?

One way to determine this would be to take the prior distribution (2.20) and subsequently set up the conditional distribution of \underline{f}_2 and \underline{f}_3 , given that $\underline{f}_1 = \hat{f}_1$. This follows from Theorem B.15 as

$$\begin{aligned} \underline{f}_2 | (\underline{f}_1 = \hat{f}_1) &\sim \mathcal{N}\left(f_2 | m(x_2) + k(x_2, x_1)k(x_1, x_1)^{-1}(\hat{f}_1 - m(x_1)), \right. \\ &\quad \left. k(x_2, x_2) - k(x_2, x_1)k(x_1, x_1)^{-1}k(x_1, x_2)\right). \end{aligned} \quad (2.21)$$

and identically for \underline{f}_3 . Although often we leave out the addition ‘ $|(\underline{f}_1 = \hat{f}_1)$ ’ and just assume it is clear we are talking about the posterior distribution.

With the above expression we can find the posterior distribution of \underline{f}_2 and \underline{f}_3 . This process of calculating the posterior distribution is what I call *predicting*, although the common term in literature is *conditioning*. If we would plot the predictions, they would look like Figure 2.4. Note that the error bars (the variances) have gotten smaller, especially for \underline{f}_2 , while we only did a measurement of \underline{f}_1 .

2.2.3. SPLITTING UP THE MEASUREMENT AND TRIAL POINTS

You may have noticed that the expressions of our distributions are already becoming quite long. And in addition, it is hard to keep track of for which input points x we have performed measurements and for which input points x we make predictions. So we could definitely use a better notation system.

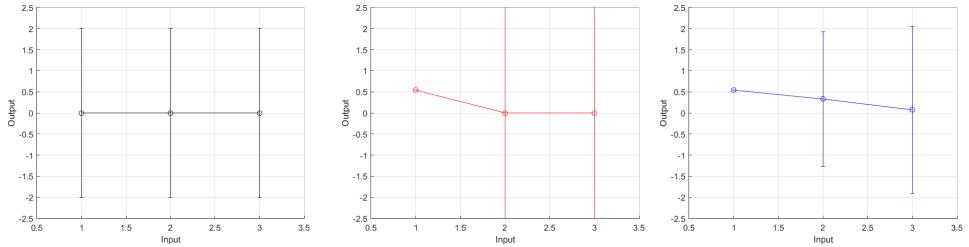


Figure 2.4: A first example of GP regression. The left figure shows the prior distribution $\mathcal{N}(0, 1^2)$ for three measurement points. The middle figure shows the measurement, where we measure the left point exactly (0 variance) but not the middle or right point (infinite variance). The right figure shows the posterior distribution. Note that the measurement of the first point has affected the predictions for the other points.

What we generally do, is write all the *measurement points* – so input points x for which we have performed measurements – as $x_{m_1}, x_{m_2}, \dots, x_{m_{n_m}}$, with n_m the number of measurements. We put all these points into the *measurement input set*⁷ X_m . We then write $\underline{f}(X_m) = \underline{f}_m$ (the *measured function values*), $m(X_m) = \mathbf{m}_m$ and $k(X_m, X_m) = K_{mm}$.

Similarly, all the points that we want to predict the function value $f(x)$ of are denoted by $x_{*_1}, x_{*_2}, \dots, x_{*_n_*}$, with n_* the number of trial points. We call these *trial points* and put them in the *trial input set* X_* . We then write $\underline{f}(X_*) = \underline{f}_*$ (the *trial function values*), the $m(X_*) = \mathbf{m}_*$ and $k(X_*, X_*) = K_{**}$. In addition, we write $k(X_*, X_m) = K_{*m} = K_{m*}^T = k(X_m, X_*)^T$.

The prior distribution of \underline{f}_m and \underline{f}_* is now given by

$$\begin{bmatrix} \underline{f}_m \\ \underline{f}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \underline{f}_m \\ \underline{f}_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_m \\ \mathbf{m}_* \end{bmatrix}, \begin{bmatrix} K_{mm} & K_{m*} \\ K_{*m} & K_{**} \end{bmatrix} \right). \quad (2.22)$$

According to the same Theorem B.15, the posterior distribution of \underline{f}_* , given that $\underline{f}_m = \hat{\mathbf{f}}_m$, now equals

$$\underline{f}_* \sim \mathcal{N} \left(\underline{f}_* \middle| \mathbf{m}_* + K_{*m} K_{mm}^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m), K_{**} - K_{*m} K_{mm}^{-1} K_{m*} \right). \quad (2.23)$$

The nice thing is that, with this expression, we can incorporate as many measurement points and as many trial points as we want. (Within the limits of our computer.) Some example results are shown in Figures 2.5.

2.2.4. IMPLEMENTING NOISY MEASUREMENTS

What happens when we do not measure $\underline{f}_{m_1}, \dots, \underline{f}_{m_{n_m}}$ precisely? What if there is measurement noise $\underline{v} \sim \mathcal{N}(\underline{v}|0, \hat{\sigma}_f^2)$ again, and we have

$$\hat{\underline{f}}_{m_1} = \underline{f}_{m_1} + \underline{v}_1, \dots, \hat{\underline{f}}_{m_{n_m}} = \underline{f}_{m_{n_m}} + \underline{v}_{n_m}. \quad (2.24)$$

⁷In literature, the subscript m is often omitted. So the measurement input set X_m is denoted as X . To prevent confusion between the various other sets we are still about to see, we will always include the subscript m , unless specifically indicated otherwise.

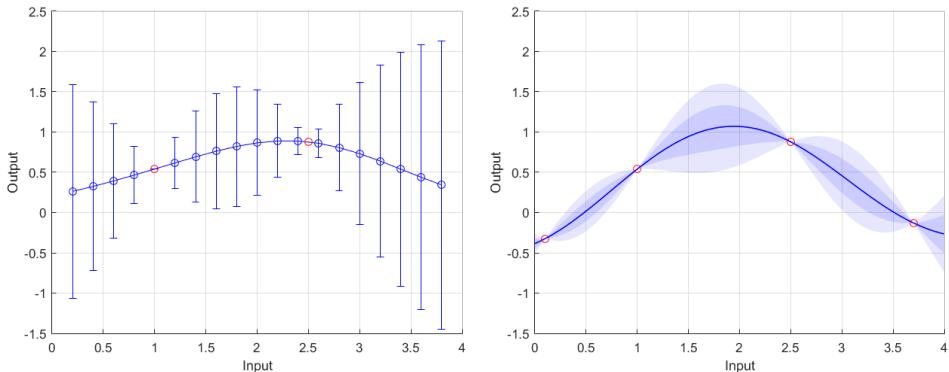


Figure 2.5: A second example of GP regression, with $\lambda_f = \lambda_x = 1$. For the left figure we have used $n_m = 2$ measurement points and $n_* = 20$ trial points. Error bars denote two times the standard deviation. For the right figure we have used $n_m = 4$ measurement points and $n_* = 400$ trial points. To prevent chaos, we have replaced the error bars by a colored area, where the inner (darker) area is one times the standard deviation and the outer (lighter) area is two times the standard deviation. This is something that we will continue to do from now on.

Or, in vector notation, $\hat{\underline{f}}_m = \underline{f}_m + \underline{v}$, where \underline{v} is distributed according to

$$\underline{v} = \begin{bmatrix} \underline{v}_1 \\ \vdots \\ \underline{v}_{n_m} \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} v_1 \\ \vdots \\ v_{n_m} \end{bmatrix} \mid \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} \hat{\sigma}_{f_{m1}}^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{\sigma}_{f_{mn_m}}^2 \end{bmatrix} \right) = \mathcal{N}(\underline{v} | \mathbf{0}, \hat{\Sigma}_{f_m}). \quad (2.25)$$

What do we do then?

In this case there are different ways to tackle the problem. In most textbooks, people generally look at the prior distribution of the noisy measurement vector $\hat{\underline{f}}_m = \underline{f}_m + \underline{v}$. Identically to (2.6), it equals

$$\hat{\underline{f}}_m \sim \mathcal{N}(\underline{m}_m + \mathbf{0}, K_{mm} + \hat{\Sigma}_{f_m}). \quad (2.26)$$

Now, again using Theorem B.15, we can find that the posterior distribution of \underline{f}_* , given that $\hat{\underline{f}}_m$ is some deterministic value $\hat{\underline{f}}_m$, equals

$$\underline{f}_* \sim \mathcal{N} \left(\underline{m}_* + K_{*m} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\underline{f}}_m - \underline{m}_m), K_{**} - K_{*m} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{m*} \right). \quad (2.27)$$

For the full proof, see Theorem B.16. Do note that $\hat{\underline{f}}_m$ denotes our actual measurement. So this is where we should put the numbers we read from our measurement equipment.

We will go for a different approach here though, which is (in my eyes) more intuitive and more powerful. It is the approach we also used in Section 2.1.2.

The idea is that we have two distributions. First there is the prior distribution (2.22). Additionally, our measurements have told us that

$$\underline{f}_m = \hat{\underline{f}}_m - \underline{v} \sim \mathcal{N}(\underline{f}_m | \hat{\underline{f}}_m, \hat{\Sigma}_{f_m}). \quad (2.28)$$

We want to merge these two distributions, but we can only merge random vectors of the same size. Keeping in mind that our measurements have not told us anything (directly) about the trial function values \underline{f}_* , we can extend the above to

$$\begin{bmatrix} \underline{f}_m \\ \underline{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \underline{f}_m \\ \underline{f}_* \end{bmatrix} \middle| \begin{bmatrix} \hat{\underline{f}}_m \\ * \end{bmatrix}, \begin{bmatrix} \hat{\Sigma}_{f_m} & * \\ * & \infty \end{bmatrix}\right). \quad (2.29)$$

Here, the term ∞ can be seen as a matrix in which all eigenvalues are infinitely large. So you may write it as ∞I as well, although we will just stick with the shorter notation ∞ . (See Section B.4.5 for a further background on this.)

If we subsequently merge the prior distribution (2.22) and the measured distribution (2.29) using (2.15), we wind up (see Theorem B.22) with the *GP regression equation*

$$\begin{aligned} \begin{bmatrix} \underline{f}_m \\ \underline{f}_* \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \underline{f}_m \\ \underline{f}_* \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \Sigma_{mm} & \Sigma_{m*} \\ \Sigma_{*m} & \Sigma_{**} \end{bmatrix}\right), \quad (2.30) \\ \begin{bmatrix} \Sigma_{mm} & \Sigma_{m*} \\ \Sigma_{*m} & \Sigma_{**} \end{bmatrix} &= \begin{bmatrix} K_{mm} - K_{mm}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mm} & K_{m*} - K_{mm}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{m*} \\ K_{*m} - K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mm} & K_{**} - K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{m*} \end{bmatrix} \\ &= \begin{bmatrix} K_{mm}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} \hat{\Sigma}_{f_m} & \hat{\Sigma}_{f_m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{m*} \\ K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} \hat{\Sigma}_{f_m} & K_{**} - K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{m*} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_* \end{bmatrix} &= \begin{bmatrix} \boldsymbol{m}_m + K_{mm}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\underline{f}}_m - \boldsymbol{m}_m) \\ \boldsymbol{m}_* + K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\underline{f}}_m - \boldsymbol{m}_m) \end{bmatrix} \\ &= \begin{bmatrix} \Sigma_{mm} (K_{mm}^{-1} \boldsymbol{m}_m + \hat{\Sigma}_{f_m}^{-1} \hat{\underline{f}}_m) \\ \boldsymbol{m}_* + K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\underline{f}}_m - \boldsymbol{m}_m) \end{bmatrix}. \end{aligned}$$

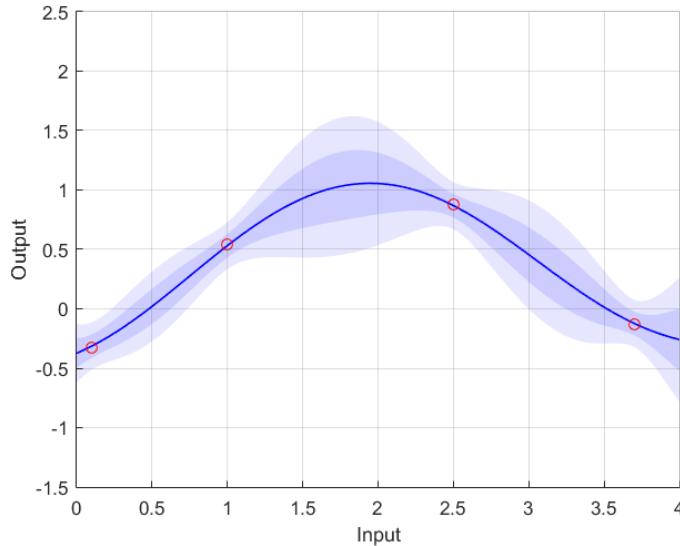
There are two ways of writing the above expression. Often the second is easier to apply, while the first is easier to remember. The powerful thing is that we now not only find the posterior distribution of the trial function values \underline{f}_* , but also that of the measured function values \underline{f}_m , in case we need them.

It is interesting to note that, when $\hat{\Sigma}_{f_m} \rightarrow 0$ and the measurement hence becomes infinitely precise, then $\boldsymbol{\mu}_m \rightarrow \hat{\underline{f}}_m$, $\Sigma_{mm} \rightarrow 0$ and the expression for \underline{f}_* reduces to (2.23).

So what is the effect of adding measurement noise to our predictions? Basically, the posterior uncertainties (variance) will be slightly bigger. To see how, compare the earlier Figure 2.5 (right) with Figure 2.6.

2.3. DIFFERENT VIEWS ON GAUSSIAN PROCESSES

Now we know how Gaussian process regression works. But we haven't really looked at what a Gaussian process actually is. That is what we will look at now. We start with the formal definition (Section 2.3.1), continue with a more intuitive view (Section 2.3.2) and end with a view on what GP regression mathematically comes down to (Section 2.3.3).



2

Figure 2.6: An adjusted version of Figure 2.5 (right) in which measurement noise with standard deviation $\hat{\sigma}_{fm} = 0.1$ has been implemented.

2.3.1. THE FORMAL DEFINITION OF A GAUSSIAN PROCESS

It is time to look at the formal definition of a Gaussian process. A Gaussian process formally is a collection of a (possibly infinite) number of random variables f_1, f_2, \dots , any finite number of which has a joint Gaussian distribution. Let's take a look at how exactly this works.

Generally, a Gaussian process has an *index variable* associated with it. For us, this will be the input x . It could also be the time t or a multi-dimensional input vector \mathbf{x} . (We'll discuss the latter case soon in Section 2.4.1.) For now, we will write the index variable as x though. Different values of the index variable x will result in different random variables $f(x)$.

You should note here that x can be (and often is) a continuous variable, and hence can take infinitely many different values. As such, our Gaussian process consists of just as many random variables $f(x)$. However, there is no such thing as an infinitely large joint Gaussian distribution, so we cannot work with this directly. But if we would take a finite number of points x_1, \dots, x_n , merge these into a set X and set up the distribution $\underline{f} = f(X)$, then we get a subset of all these random variables which does have a (finite) joint Gaussian distribution.

A (finite) vector $\underline{f} \sim \mathcal{N}(f|\mu, \Sigma)$ with a Gaussian distribution is fully defined when we know the mean vector μ and the covariance matrix Σ . To fully define a Gaussian process, for each possible set X that we may take, we need something similar. First of all, we need a *mean function* $m(x)$ and secondly a *covariance function* $k(x, x')$. If we know these two functions, our Gaussian process is fully defined.

At this point you've probably noticed that our prior distribution (2.20) satisfies this criterion. It is worthwhile to notice that also the posterior distribution of (2.23) or (2.30)

satisfies this criterion, for any set of trial points X_* we may take. In this case, the *posterior mean function* $m_{post}(x)$ and *posterior covariance function* $k_{post}(x, x')$, expressed in the prior mean function $m(x)$ and prior covariance function $k(x, x')$, equal

$$m_{post}(x_*) = m(x_*) + k(x_*, X_m) \left(k(X_m, X_m) + \hat{\Sigma}_{f_m} \right)^{-1} (\hat{f}_m - \mathbf{m}_m), \quad (2.31)$$

$$k_{post}(x_*, x'_*) = k(x_*, x'_*) - k(x_*, X_m) \left(k(X_m, X_m) + \hat{\Sigma}_{f_m} \right)^{-1} k(X_m, x_*). \quad (2.32)$$

2.3.2. THE INTUITIVE VIEW OF A GAUSSIAN PROCESS

There is also a more intuitive view of a Gaussian process. The way I see a Gaussian process is as a distribution over functions. Let me explain what that means.

Suppose that we have a Gaussian process with input points from the interval $[0, 4]$. And suppose that we take a hundred trial points evenly distributed over this interval, like we also did in earlier plots. We know the joint distribution $\mathcal{N}(\mu_*, \Sigma_{**})$ of these trial points, and as such we can take samples from it. A few examples of such samples is shown in Figure 2.7.

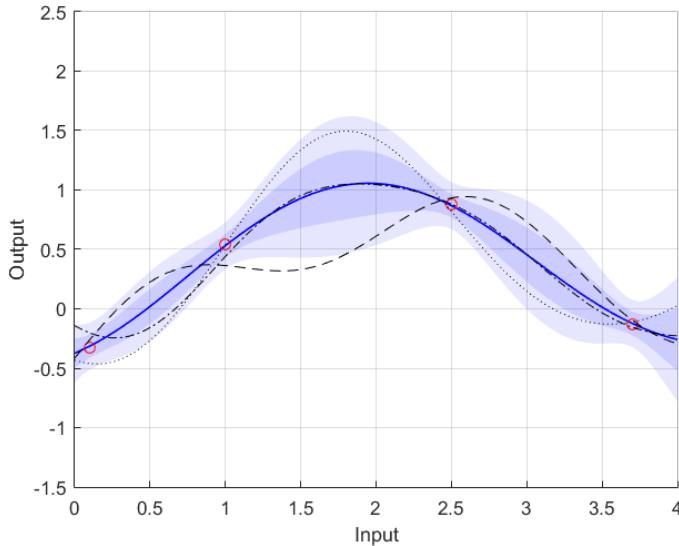


Figure 2.7: An example of three sample functions taken from a Gaussian process. Conditions are the same as in Figure 2.6. In theory, for 95% of the points, the samples should fall within the gray 95% region.

Each of the samples in Figure 2.7 now is a possible function $f(x)$ that could have (with the given measurement noise) generated the data that we have measured. As such, this is the more intuitive view on what a Gaussian process is.

2.3.3. THE MATHEMATICAL VIEW OF GAUSSIAN PROCESS REGRESSION

We can also look at what Gaussian process regression mathematically comes down to. In fact, let's consider the prediction equation for the mean μ_* for a single trial input point

x_* . From (2.30) we know that

$$\mu_* = m(x_*) + K_{*m} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{f}_m - \mathbf{m}_m). \quad (2.33)$$

Let's define the vector $\alpha = (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{f}_m - \mathbf{m}_m)$. This vector does not depend on x_* , so once all our measurements are known, we only have to compute it once. We can now rewrite the above expression to

$$\mu_* = m(x_*) + \sum_{i=1}^{n_m} \alpha^i k(x_{m_i}, x_*). \quad (2.34)$$

So what we see is that the posterior mean function – which happens to be our most likely function value – is a sum of n_m covariance functions. Mathematically, we can also see this as a sum of *basis functions*. And we use just as many basis functions as we have measurements. Although later on, in Section 4.1.3, we will also look at cases where we can vary the number of basis functions.

The interesting thing is that, when you look at other function approximation methods like neural networks and support vector machines, and what they mathematically come down to, then it is the same. Instead of covariance functions, support vector machines use kernels and neural networks use activation functions, but eventually all methods wind up with a sum of nonlinear basis functions. Although the philosophy behind these methods may be very different, mathematically they are very similar.

2.4. MULTI-DIMENSIONAL INPUTS AND OUTPUTS

Previously we have looked at a single-input single-output function $f(x)$. Now it is time to expand on that. First we will look at approximating functions with multiple inputs (Section 2.4.1). Then we also examine the case where we have multiple function outputs (Section 2.4.2), find a method to simplify this case (Section 2.4.3) and look at what the consequences of using multiple outputs are for the covariance functions (Section 2.4.4).

2.4.1. USING MULTI-DIMENSIONAL INPUT POINTS

So far we have looked at functions $f(x)$ with a single input parameter x . Now let's look at functions $f(x^1, x^2, \dots, x^{d_x})$ with multiple input parameters x^1, x^2, \dots, x^{d_x} . Although often we will write the multi-input function, using the vector notation, as $f(\mathbf{x})$, with \mathbf{x} the *input vector* of size d_x . Note that, because we are already using subscripts to distinguish different input points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, we use superscripts to distinguish different elements within the *same* input point \mathbf{x} .

The question now is, can we apply Gaussian process regression to this as well? The answer is yes, and we can actually use the exact same equations. The only thing that is different is the covariance function. (And possibly the mean function, but we will stick with $m(\mathbf{x}) = 0$ here.) The squared exponential covariance function is now given by

$$k(\mathbf{x}, \mathbf{x}') = \lambda_f^2 \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}')^T \Lambda_x^{-1} (\mathbf{x} - \mathbf{x}')\right). \quad (2.35)$$

The parameter λ_f is the same as before, but the matrix Λ_x is new. It is the *matrix of squared length scales* for the input. In practice, unless we are dealing with rather advanced applications, we assume it to be a diagonal matrix

$$\Lambda_x = \begin{bmatrix} \lambda_{x_1}^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{x_{d_x}}^2 \end{bmatrix}. \quad (2.36)$$

The quantities $\lambda_{x_1}, \dots, \lambda_{x_{d_x}}$ effectively determine how much the Gaussian process will/can vary when the corresponding input parameters x_1, \dots, x_{d_x} vary. We'll look more into the effects of these parameters in Section 3.1.

Using our new covariance function, we can again set up matrices K_{mm} , K_{m*} , K_{*m} and K_{**} and subsequently apply the regression equation (2.30) to predict the function values f_* . And now we might get a result that looks like Figure 2.8.

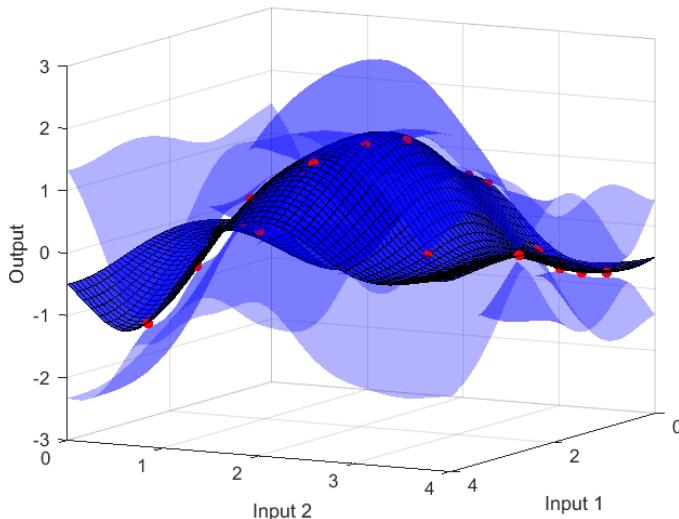


Figure 2.8: Gaussian process regression with multiple inputs. The length scales used were $\lambda_f = 1$, $\lambda_{x_1} = 1$, $\lambda_{x_2} = 0.5$ and $\hat{\sigma}_{f_m} = 0.02$. The solid plane is the mean, while the partly transparent planes are the two times standard deviation planes, together enclosing the 95% region.

2.4.2. USING MULTI-DIMENSIONAL OUTPUT POINTS

If it is so easy to use a multi-input function $f(\mathbf{x})$, with multiple input parameters x^1, \dots, x^{d_x} , is it also easy to use a multi-output function $\underline{f}(\mathbf{x})$, with multiple output parameters $f^1(\mathbf{x}), \dots, f^{d_f}(\mathbf{x})$? The answer here is ‘Mostly, yes. But there is more to it.’ Let’s take a look at what exactly the issues are.

Previously, before we would apply our GP regression equations, we had to set up the prior distribution. That is, we had to indicate how different function values $\underline{f}(\mathbf{x})$ and $\underline{f}(\mathbf{x}')$ were correlated, prior to doing any measurements. Now we need to do more.

We also need to indicate how the different function outputs $\underline{f}^1(\mathbf{x}), \dots, \underline{f}^{d_f}(\mathbf{x}')$ are correlated. And the way in which we can do this, is by replacing our scalar covariance function $k(\mathbf{x}, \mathbf{x}')$ by a *matrix covariance function*

$$k(\mathbf{x}, \mathbf{x}') = \begin{bmatrix} k_{11}(\mathbf{x}, \mathbf{x}') & \cdots & k_{1d_f}(\mathbf{x}, \mathbf{x}') \\ \vdots & \ddots & \vdots \\ k_{d_f 1}(\mathbf{x}, \mathbf{x}') & \cdots & k_{d_f d_f}(\mathbf{x}, \mathbf{x}') \end{bmatrix}. \quad (2.37)$$

By using this, we can apply GP regression in the usual way. That is, we can do measurements $\hat{\mathbf{f}}_{m_1}, \dots, \hat{\mathbf{f}}_{m_{n_m}}$, except now we get measurement vectors instead of measurement values. We can lump these together into a measured distribution \underline{f}_m like we did in (2.28). We can then set up the covariance matrices K_{mm} , K_{m*} , K_{*m} and K_{**} , although now they all consist of blocks of d_f by d_f sub-matrices. So K_{mm} will be a $d_f n_m \times d_f n_m$ matrix. And then we can once more apply the GP regression equation (2.30).

2.4.3. A SIMPLIFICATION WHEN USING MULTI-DIMENSIONAL OUTPUTS

Let's think for a second about what $k_{12}(\mathbf{x}, \mathbf{x}')$ actually means though. It is the prior covariance between the outputs $\underline{f}^1(\mathbf{x})$ and $\underline{f}^2(\mathbf{x}')$; so between two different function outputs. But often we don't know anything in advance about how these two functions outputs relate to each other. There is no reason why, if $\underline{f}^1(\mathbf{x})$ happens to be positive for some point \mathbf{x} , that also $\underline{f}^2(\mathbf{x}')$ should be positive (or negative for that matter). Because of that, we generally have $k_{12}(\mathbf{x}, \mathbf{x}') = 0$, and similarly for the other non-diagonal terms of $k(\mathbf{x}, \mathbf{x}')$. The covariance matrix hence becomes a diagonal matrix.

The interesting thing is that, when you now work out the GP regression equations, you will find that there is no link whatsoever between the different outputs $\underline{f}^1(\mathbf{x}), \underline{f}^2(\mathbf{x}), \dots$. A measurement of output $\underline{f}^1(\mathbf{x})$ will not have any effect at all on the prediction of \underline{f}^2 . (An exception to this occurs in Chapter 5 when we add input noise.) As a consequence, what we could also do is apply the Gaussian process regression equation (2.30) fully separately to each individual output $\underline{f}^1(\mathbf{x}), \underline{f}^2(\mathbf{x}), \dots$.

What is the advantage of this? Well, we used to have to invert K_{mm} , which was an $d_f n_m \times d_f n_m$ matrix. Now we have d_f separate GP regression algorithms, each with a matrix K_{mm} of size $n_m \times n_m$. Inverting an $n_m \times n_m$ times matrix a number of d_f times is a lot more computationally efficient than inverting a single huge $d_f n_m \times d_f n_m$ matrix. In fact, later on in Chapter 4 we will look more at the runtime of GP regression, and then we will see that it is about d_f^2 times faster. So this simplification will save us some time, especially for larger values of d_f .

2.4.4. THE COVARIANCE FUNCTIONS WITHIN THE COVARIANCE MATRIX

Finally, let's look at the remaining diagonal terms $k_{11}(\mathbf{x}, \mathbf{x}'), k_{22}(\mathbf{x}, \mathbf{x}'), \dots$. These covariance functions can all be different. (We will learn more about choosing covariance functions in Section 3.2.) For simplicity, in this thesis we will only use the squared exponential covariance function (2.19) though.

But the covariance functions do have parameters (length scales), and these length scales may be different for each covariance function. In fact, the output length scales

$\lambda_{f_1}, \dots, \lambda_{f_{d_f}}$ will most certainly be different for different outputs $f^1(\mathbf{x}), \dots, f^{d_f}(\mathbf{x})$. The squared input length scales $\Lambda_{x_1}, \dots, \Lambda_{x_{d_f}}$ (which determines how quickly the output varies for varying inputs) is often the same for different outputs. Although in some special cases it may be useful to choose a different input length scale $\Lambda_{x_1}, \dots, \Lambda_{x_{d_f}}$ for each output $f^1(\mathbf{x}), \dots, f^{d_f}(\mathbf{x})$, for instance when one output $f^1(\mathbf{x})$ varies strongly with one input x^1 but not so much with another input x^2 , while another output $f^2(\mathbf{x})$ varies more strongly with the other input x^2 but not so much with x^1 . How exactly we choose these parameters is something we will look at more closely in Section 3.1 though.

2.5. THE DERIVATIVE AND INTEGRAL OF A GAUSSIAN PROCESS

We have seen in Section 2.3.2 that a Gaussian process is basically a distribution over functions. And we can take the derivative of functions. So based on this, the derivative of a Gaussian process is again a distribution over functions. To be precise, it is again a Gaussian process. And naturally the same holds for the inverse derivative: the integral.

In Section 2.5.1 we will actually prove this. We will then find the corresponding mean and covariance functions in Section 2.5.2. In Section 2.5.3 we look at how we can actually make measurements of the derivative of a function and incorporate those. Finally, we examine integrals of Gaussian processes in Section 2.5.4.

2.5.1. THE DERIVATIVE OF A GAUSSIAN PROCESS

Consider a function $f(\mathbf{x})$. The derivative of this function is formally defined as

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \lim_{d\mathbf{x} \rightarrow \mathbf{0}} \frac{f(\mathbf{x} + d\mathbf{x}) - f(\mathbf{x})}{d\mathbf{x}}. \quad (2.38)$$

We can apply the same definition to a Gaussian process $\underline{f}(\mathbf{x})$. We now get

$$\frac{d\underline{f}(\mathbf{x})}{d\mathbf{x}} = \lim_{d\mathbf{x} \rightarrow \mathbf{0}} \frac{\underline{f}(\mathbf{x} + d\mathbf{x}) - \underline{f}(\mathbf{x})}{d\mathbf{x}}. \quad (2.39)$$

To find this, we need to subtract $\underline{f}(\mathbf{x})$, which is a Gaussian distribution, from $\underline{f}(\mathbf{x} + d\mathbf{x})$, which is another Gaussian distribution. And as we have learned (see for instance Theorem B.13), a linear combination of Gaussian parameters is again a Gaussian. This shows that $d\underline{f}(\mathbf{x})/d\mathbf{x}$ is a Gaussian distribution.

2.5.2. THE MEAN AND COVARIANCE OF THE DERIVATIVE

Let's write the derivative of a Gaussian process as

$$\frac{d\underline{f}(\mathbf{x})}{d\mathbf{x}} = \mathcal{N}(m_d(\mathbf{x}), k_d(\mathbf{x}, \mathbf{x}')). \quad (2.40)$$

So $m_d(\mathbf{x})$ is the *derivative mean function* and $k_d(\mathbf{x}, \mathbf{x}')$ is the *derivative covariance function*. Let's find what they are equal to.

For the derivative mean function we have

$$\begin{aligned} m_d(\mathbf{x}) &= \mathbb{E}\left[\frac{df(\mathbf{x})}{d\mathbf{x}}\right] = \mathbb{E}\left[\lim_{d\mathbf{x} \rightarrow 0} \frac{f(\mathbf{x} + d\mathbf{x}) - f(\mathbf{x})}{d\mathbf{x}}\right] \\ &= \lim_{d\mathbf{x} \rightarrow 0} \frac{\mathbb{E}[f(\mathbf{x} + d\mathbf{x})] - \mathbb{E}[f(\mathbf{x})]}{d\mathbf{x}} = \lim_{d\mathbf{x} \rightarrow 0} \frac{m(\mathbf{x} + d\mathbf{x}) - m(\mathbf{x})}{d\mathbf{x}} = \frac{dm(\mathbf{x})}{d\mathbf{x}}. \end{aligned} \quad (2.41)$$

2

In other words, to find the mean of the derivative of a Gaussian process, we can just take the derivative of the mean. Although actually we could have also found this in a much quicker way. If we realize that both the expectation and the derivative operators are linear operators, and we can hence change the order in which they are applied, we could have used

$$\mathbb{E}\left[\frac{df(\mathbf{x})}{d\mathbf{x}}\right] = \frac{d}{d\mathbf{x}}\mathbb{E}[f(\mathbf{x})] = \frac{dm(\mathbf{x})}{d\mathbf{x}}. \quad (2.42)$$

Now this is definitely easier than (2.41).

Next, what is the covariance function? We can use the same trick here, resulting in

$$\begin{aligned} k_d(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[\left(\frac{df(\mathbf{x})}{d\mathbf{x}} - \frac{dm(\mathbf{x})}{d\mathbf{x}}\right)\left(\frac{df(\mathbf{x}')}{d\mathbf{x}'} - \frac{dm(\mathbf{x}')}{d\mathbf{x}'}\right)\right] \\ &= \frac{d^2}{d\mathbf{x} d\mathbf{x}'} \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))\right] \\ &= \frac{d^2 k(\mathbf{x}, \mathbf{x}')}{d\mathbf{x} d\mathbf{x}'}. \end{aligned} \quad (2.43)$$

So to find the covariance function of $df(\mathbf{x})/d\mathbf{x}$, we have to take the derivative of $k(\mathbf{x}, \mathbf{x}')$ with respect to both \mathbf{x} and \mathbf{x}' . Do keep in mind that, when you have already incorporated measurements into your GP, then you will need to take the derivatives of the *posterior* mean and covariance functions. In other words, you will need to take the derivatives of (2.31) and (2.32). When we do, we can find a figure like Figure 2.9.

2.5.3. IMPLEMENTING DERIVATIVE MEASUREMENTS

Consider $df(\mathbf{x})/d\mathbf{x}$ and $f(\mathbf{x}')$ for certain points \mathbf{x} and \mathbf{x}' . These are both Gaussian random variables. So naturally, we can find their covariance. It will equal

$$\begin{aligned} \mathbb{V}\left[\frac{df(\mathbf{x})}{d\mathbf{x}}, f(\mathbf{x}')\right] &= \mathbb{E}\left[\left(\frac{df(\mathbf{x})}{d\mathbf{x}} - \frac{dm(\mathbf{x})}{d\mathbf{x}}\right)(f(\mathbf{x}') - m(\mathbf{x}'))\right] \\ &= \frac{d}{d\mathbf{x}}\mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))\right] \\ &= \frac{dk(\mathbf{x}, \mathbf{x}')}{d\mathbf{x}}. \end{aligned} \quad (2.44)$$

This expression is actually quite useful. Suppose that, next to measuring function values $f(\mathbf{x}_m)$ at input points \mathbf{x}_m , we also measure the derivatives $\frac{df}{d\mathbf{x}}(\mathbf{x}_d)$ at input points \mathbf{x}_d . (Here, the set X_m can be the same as X_d , but it can also be different.) With the above expression, we can incorporate this derivative data.

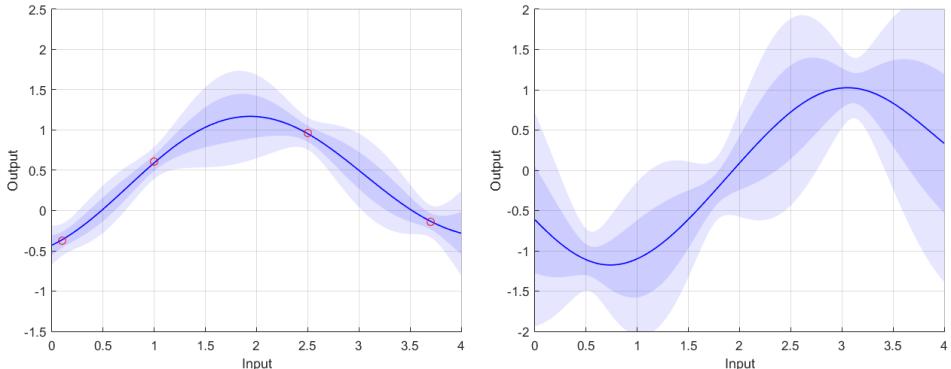


Figure 2.9: The derivative of a Gaussian process is also a Gaussian process. The left figure equals Figure 2.6. The right figure equals its derivative. It may be interesting to note that the variance of the derivative is generally not the smallest at the measurement locations, but somewhere in-between measurements.

How does it work? Well, the easiest way is to expand the measurement mean vector \mathbf{m}_m and covariance matrix K_{mm} to incorporate this derivative data. That is, we redefine \mathbf{m}_m and K_{mm} (and K_{m*} and K_{*m}) such that

$$\begin{bmatrix} \mathbf{m}_m \\ \mathbf{m}_* \end{bmatrix} = \begin{bmatrix} m(X_m) \\ \frac{dm}{dx}(X_d) \\ m(X_*) \end{bmatrix}, \quad (2.45)$$

$$\begin{bmatrix} K_{mm} & K_{m*} \\ K_{*m} & K_{**} \end{bmatrix} = \begin{bmatrix} k(X_m, X_m) & \frac{dk}{dx}(X_m, X_d) & k(X_m, X_*) \\ \frac{dk}{dx}(X_d, X_m) & \frac{d^2k}{dx^2}(X_d, X_d) & \frac{dk}{dx}(X_d, X_*) \\ k(X_*, X_m) & \frac{dk}{dx}(X_*, X_d) & k(X_*, X_*) \end{bmatrix}. \quad (2.46)$$

Note that, for the derivatives of the covariance function, we always take the derivative with respect to the parameter that we will plug the derivative points into.

Next, we should also incorporate the derivative measurements into $\hat{\mathbf{f}}_m$, and their corresponding noise in $\hat{\Sigma}_{f_m}$. But then the rest of our GP regression works exactly the same, resulting in a plot like Figure 2.10.

2.5.4. INTEGRALS OF GAUSSIAN PROCESSES

If the derivative of a Gaussian process is again a Gaussian process, can we then also integrate a Gaussian process? Not very surprisingly, the answer is yes. And there have already been some nice applications of this. (See for instance Wahlström (2015), Section 2.3.4.)

Let's consider single-input Gaussian processes $f(x)$ first. The first thing we could try is integrating over a fixed interval $[a, b]$. In this case, the outcome is not a Gaussian process though, but a single Gaussian random variable. To be precise, we would get

$$\int_a^b f(x) dx = \mathcal{N} \left(\int_a^b m(x) dx, \int_a^b \int_a^b k(x, x') dx' dx \right), \quad (2.47)$$

where the mean and variance do not depend on any parameter anymore. If we do want to get a Gaussian process as outcome, we have to let our integration interval depend on

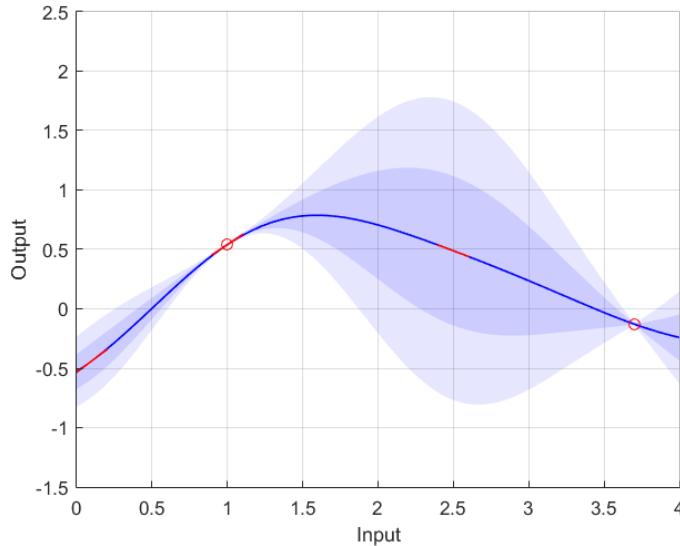


Figure 2.10: Gaussian process regression using derivative data. The length scales used were $\lambda_f = 1$, $\lambda_x = 1$ and $\sigma_{f_m} = 0.01$ (negligible) for both value and derivative measurements. For some points only derivative data is measured (stripes) and for other points only function values are measured (circles).

some parameter y . For instance, we can define $\underline{I}(y)$ as

$$\underline{I}(y) = \int_0^y f(x) dx. \quad (2.48)$$

In this case, $\underline{I}(y)$ is a Gaussian process with its mean function $m_I(y)$ and covariance function $k_I(y, y')$ satisfying

$$m_I(y) = \int_0^y m(x) dx, \quad (2.49)$$

$$k_I(y, y') = \int_0^y \int_0^{y'} k(x, x') dx' dx. \quad (2.50)$$

So that is how single-input Gaussian processes can be integrated. Note that, instead of having the integral bounds vary linearly with y , we could have also had them vary in a more complicated way with y . Or we could even have integrated from y_1 to y_2 , which would have given us a Gaussian process with multiple inputs $\mathbf{y} = [y_1 \ y_2]^T$.

For multi-input Gaussian processes $f(\mathbf{x})$ things work similarly. Again we could choose to integrate over a fixed multi-dimensional area, which would give us a single Gaussian random variable. Or we could choose to integrate over a multi-dimensional area, whose exact size and/or shape depends on one or more parameters \mathbf{y} .

It is interesting to note that the input vectors \mathbf{x} (of $f(\mathbf{x})$) and \mathbf{y} (of $\underline{I}(\mathbf{y})$) do not have to be of the same size. In fact, it could very well be possible to let \mathbf{x} have more elements than \mathbf{y} , or less. The crucial thing is the number of parameters which the space we are integrating over depends on.

2.6. IDENTIFYING THE DYNAMICS OF A PITCH-PLUNGE SYSTEM

We will now apply Gaussian process regression to a simple test system: a single airfoil in an airflow. We will look at the system in Section 2.6.1 and derive the equations of motion in Section 2.6.2. Deriving the equations of motion requires some basic knowledge of aerodynamics. If you do not have this, you could either read Jr. (2010) or skip this section altogether and just use the equations of motion.

Afterwards, we will identify the dynamics of the system. We first do this only for stationary initial states (Section 2.6.3) but then extend this to identify the dynamics from any state (Section 2.6.4). We then study the differences in accuracy of these approximations and how the regression algorithm deals with this.

2.6.1. THE PITCH-PLUNGE SYSTEM SET-UP

We will consider the *pitch-plunge system* shown in Figure 2.11. 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 our 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).

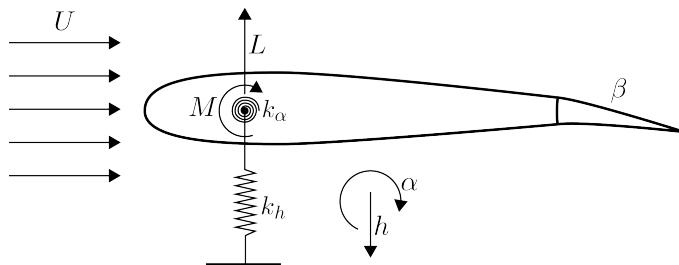


Figure 2.11: An airfoil with a trailing-edge flap. The exact system set-up is explained in the main text.

Let’s take a look at how the pitch-plunge system works. The system consists of an airfoil which is placed in an airflow with velocity U . The airfoil can *plunge* (move vertically, representing the flapping of a turbine blade) as well as *pitch* (rotate, representing the torsion of a turbine blade). 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 dynamics of the spring depend on various parameters. First, the airfoil will generate a lift force L and a moment M . These both depend on the airflow, so on the flow velocity U , the angle of attack α and the additional angle of attack caused by the vertical motion \dot{h} of the blade. (The dot represents the time-derivative $\frac{dh}{dt}$.) In addition, the airfoil also has a trailing edge flap, which is deflected by an angle β . We can control this flap ourselves, and naturally β also affects the lift and the moment.

The blade is constrained by two springs. There is a vertical spring with *spring constant* k_h and a rotational spring with spring constant $k_\alpha(\alpha)$. This latter spring constant

depends on α according to

$$k_\alpha(\alpha) = k_\alpha (1 + k_{\alpha_1} \alpha^1 + k_{\alpha_2} \alpha^2 + k_{\alpha_3} \alpha^3 + k_{\alpha_4} \alpha^4), \quad (2.51)$$

which causes the system to be nonlinear. In addition, both springs also have damping, expressed by the *damping coefficients* c_h and c_α . The value of these parameters (and all other parameters) can be found in Table 2.1.

Table 2.1: Numerical values of the parameters of the pitch-plunge system.

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

2.6.2. THE PITCH-PLUNGE SYSTEM EQUATIONS OF MOTION

Let's derive the equations of motion of the pitch-plunge system. You will encounter several parameters during this derivation. In our derivation we will use the same notation and sign convention as Ko et al. (1997, 1998). This sign convention is somewhat confusing though, so let's take a careful look at it first.

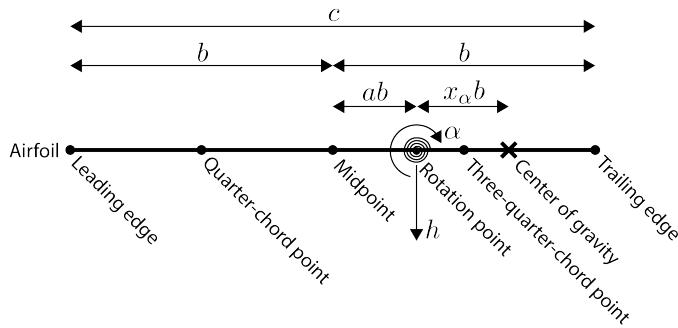


Figure 2.12: An overview of important points of the airfoil. Note that the depicted order of the points is not necessarily the true order in which these points occur. It has been set up to show the sign convention of a and x_α . To be precise, we will use $a = -0.6$ (see Table 2.1) which means that the rotation point is in reality quite close to the leading edge. Also note that the plunge h is defined as the vertical position of the rotation point, where downwards is defined as positive.

The main idea is that we have several different points on the airfoil, as shown in Figure 2.12. There is the quarter-chord position, the midpoint, the rotation point where the springs are attached, and the *center of gravity* (CG) of the airfoil. The height h of the airfoil (that is, its vertical position) is defined as the height at the *rotation point*, and we will also consider the lift L and the moment M with respect to this rotation point.

Keep in mind here that we are considering a two-dimensional airfoil. There is no 'depth'. When we do have to talk about the span of the airfoil, we assume we are working

with an airfoil of unit span. So all quantities, like the lift L , the mass m and so on, are quantities *per unit span*.

For now, suppose that there are no springs. In this case, what would the equations of motion be? Well, we can apply Newton's second law to the airfoil, but we are only allowed to do so with respect to the CG. If we denote the lift and moment with respect to the CG as L_{CG} and M_{CG} , respectively, then we have

$$\begin{bmatrix} m & 0 \\ 0 & I_{CG} \end{bmatrix} \begin{bmatrix} \ddot{h}_{CG} \\ \ddot{\alpha}_{CG} \end{bmatrix} = \begin{bmatrix} -L_{CG} \\ M_{CG} \end{bmatrix} = \begin{bmatrix} -L \\ M + Lx_\alpha b \end{bmatrix}, \quad (2.52)$$

with m the mass (per unit span) of the airfoil and I_{CG} the rotational inertia (per unit span) with respect to the CG. However, we want to have the equations of motion with respect to the rotation point. And although $\alpha = \alpha_{CG}$, we do have $h = h_{CG} - \alpha x_\alpha b$. If we implement this in the above equation and work out the results, we find that

$$\begin{bmatrix} m & mx_\alpha b \\ mx_\alpha b & I_\alpha \end{bmatrix} \begin{bmatrix} \ddot{h} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} -L \\ M \end{bmatrix}, \quad (2.53)$$

where we have defined $I_\alpha = I_{CG} + m(x_\alpha b)^2$ as the moment of inertia with respect to the rotation point.

Next, let's add the springs. These springs cause forces and moments directly in the rotation point, so we do not need to apply any transformations to them. If we add the respective forces to our equations, we wind up with

$$\begin{bmatrix} m & mx_\alpha b \\ mx_\alpha b & I_\alpha \end{bmatrix} \begin{bmatrix} \ddot{h} \\ \ddot{\alpha} \end{bmatrix} + \begin{bmatrix} c_h & 0 \\ 0 & c_\alpha \end{bmatrix} \begin{bmatrix} \dot{h} \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} k_h & 0 \\ 0 & k_\alpha(\alpha) \end{bmatrix} \begin{bmatrix} h \\ \alpha \end{bmatrix} = \begin{bmatrix} -L \\ M \end{bmatrix}. \quad (2.54)$$

The main question that still remains is 'How do we calculate the lift and the moment?'

When the airfoil is stationary, the lift is given directly by

$$L = \frac{1}{2} \rho U^2 S c_{L_\alpha} \alpha + \frac{1}{2} \rho U^2 S c_{L_\beta} \beta, \quad (2.55)$$

with ρ the *air density*, S the *surface area* of the wing, C_{L_α} the *lift coefficient per angle of attack* and C_{L_β} the *lift coefficient per control input*. You may note that no constant term involving C_{L_0} is present. This is because our airfoil is symmetric. That is, it has no camber.

When there is motion, we get extra terms though. First of all, a plunge motion \dot{h} will cause an extra angle of attack equal to \dot{h}/U . But next to that, a pitching motion $\dot{\alpha}$ will also result in a change in lift. This change in lift mainly depends on the downward velocity of the three-quarter point of the airfoil, which equals $\dot{\alpha} \left(\frac{1}{2} - \alpha \right) b$. Or at least, it depends on the ratio between this velocity and the flow velocity U . (See for instance Fung (1955).) So the extra lift that we get equals

$$\frac{1}{2} \rho U^2 S c_{L_\alpha} \frac{\dot{h}}{U} + \frac{1}{2} \rho U^2 S c_{L_\alpha} \left(\frac{1}{2} - \alpha \right) b \frac{\dot{\alpha}}{U}. \quad (2.56)$$

We can also simplify S . Because we are considering an airfoil of unit span, the surface S equals the chord length $S = c = 2b$. It follows that the lift equals

$$L = \rho U^2 b c_{L_\alpha} \left(\alpha + \frac{\dot{h}}{U} + \left(\frac{1}{2} - \alpha \right) b \frac{\dot{\alpha}}{U} \right) + \rho U^2 b c_{L_\beta} \beta. \quad (2.57)$$

For thin and symmetric airfoils, the aerodynamic moment with respect to the quarter-chord position is generally zero. As a result, we can approximate $c_{m_\alpha} = \left(\frac{1}{2} + a\right) c_{L_\alpha}$. This relationship of course does not hold for the control input coefficient c_{m_β} , whose exact value depends on the size and shape of the flap and has to be determined through measurements. It now does follow that

$$M = \rho U^2 b c_{m_\alpha} \left(\alpha + \frac{\dot{h}}{U} + \left(\frac{1}{2} - \alpha \right) b \frac{\dot{\alpha}}{U} \right) + \rho U^2 b c_{m_\beta} \beta. \quad (2.58)$$

If we subsequently insert the relations for L and M that we have found into (2.54) and work out the results, we wind up with the equations of motion

$$\begin{bmatrix} m & mx_\alpha b \\ mx_\alpha b & I_\alpha \end{bmatrix} \begin{bmatrix} \ddot{h} \\ \ddot{\alpha} \end{bmatrix} + \begin{bmatrix} c_h + \rho U b c_{L_\alpha} & \rho U b^2 c_{L_\alpha} \left(\frac{1}{2} - a \right) \\ -\rho U b^2 c_{m_\alpha} & c_\alpha - \rho U b^3 c_{m_\alpha} \left(\frac{1}{2} - a \right) \end{bmatrix} \begin{bmatrix} \dot{h} \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} k_h & \rho U^2 b c_{L_\alpha} \\ 0 & k_\alpha(\alpha) - \rho U^2 b^2 c_{m_\alpha} \end{bmatrix} \begin{bmatrix} h \\ \alpha \end{bmatrix} = \begin{bmatrix} -\rho U^2 b c_{L_\beta} \\ \rho U^2 b^2 c_{m_\beta} \end{bmatrix} \beta. \quad (2.59)$$

These are equations of motion we can use in a simulation. Although, somewhat similarly to [Lind and Baldelli \(2005\)](#), we will define the matrices

$$\begin{aligned} M &= \begin{bmatrix} m & mx_\alpha b \\ mx_\alpha b & I_\alpha \end{bmatrix}, & D &= \begin{bmatrix} 0 & \rho b c_{L_\alpha} \\ 0 & -\rho b^2 c_{m_\alpha} \end{bmatrix}, \\ C &= \begin{bmatrix} c_h & 0 \\ 0 & c_\alpha \end{bmatrix}, & E &= \begin{bmatrix} \rho b c_{L_\alpha} & \rho b^2 c_{L_\alpha} \left(\frac{1}{2} - a \right) \\ -\rho b^2 c_{m_\alpha} & -\rho b^3 c_{m_\alpha} \left(\frac{1}{2} - a \right) \end{bmatrix}, \\ K(\mathbf{x}) &= \begin{bmatrix} k_h & 0 \\ 0 & k_\alpha(\alpha) \end{bmatrix}, & F &= \begin{bmatrix} -\rho U^2 b c_{L_\beta} \\ \rho U^2 b^2 c_{m_\beta} \end{bmatrix}. \end{aligned} \quad (2.60)$$

With these matrices, and with the state vector $\mathbf{x} = [h \ \alpha]^T$, we can write

$$M \ddot{\mathbf{x}} + (C + UE) \dot{\mathbf{x}} + (K(\mathbf{x}) + U^2 D) \mathbf{x} = U^2 F \beta. \quad (2.61)$$

With these equations of motion we can set up a Simulink simulation of the system, allowing us to simulate it for various initial conditions, various control laws as well as various disturbances in the wind velocity U . An example of a system response is shown in Figure 2.13.

Note that, because of the term $k_\alpha(\alpha)$ within $K(\mathbf{x})$, the system is nonlinear. Although if we wanted to linearize it, we could simply replace $k_\alpha(\alpha)$ by the constant k_α . So we can choose to either use the nonlinear or the linear model of the pitch-plunge system.

2.6.3. A FIRST APPROXIMATION OF THE STATE TRANSITION FUNCTION

As a first challenge, we will approximate the discretized state transition function

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \dot{\mathbf{x}}_k, \beta_k), \quad (2.62)$$

for some time step $\Delta t = 0.1$ s. Note that this is a relatively large time step. In fact, it is more than a quarter of the period of the system's flutter behavior. (See Figure 2.13.)

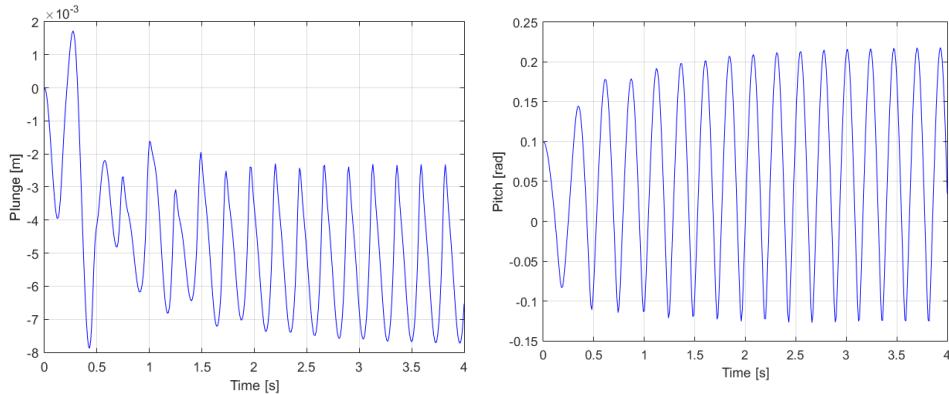


Figure 2.13: The response of the pitch-plunge system to the initial conditions of $\alpha = 0.1$ rad and $h = \dot{h} = \dot{\alpha} = 0$. The wind speed was relatively high ($U = 15$ m/s). This high wind speed, in combination with the nonlinear spring and the absence of control input, caused *flutter* behavior. That is, the system state converges to a high-frequency oscillation, known as the *limit cycle*.

For simplicity, we will first only look at the influence of \mathbf{x}_k on \mathbf{x}_{k+1} . In other words, we will vary \mathbf{x}_k , but set $\dot{\mathbf{x}}_k$ and β_k in (2.62) to zero. So we put the system in $n_m = 30$ different stationary initial positions \mathbf{x}_k , release it without applying any input and look where it winds up 0.1 seconds later.

The resulting problem is actually a two-dimensional problem. The state transition function \mathbf{f} now has only two (non-zero) inputs and two relevant outputs. As such, it is a relatively easy problem. The results of it are shown in Figure 2.14.

From Figure 2.14 we can see that GP regression can adequately predict the state transition function for this two-dimensional problem. Even with only thirty measurements, the predictions are accurate. The GP regression algorithm also knows it is accurate, because the variance is relatively small.

It is also worthwhile to note that a positive value of α_k generally causes the value of h_{k+1} to decrease. This means that the airfoil goes up (because downwards is defined as positive) which is what we would expect to happen.

By the way, you may be thinking, ‘Isn’t it very unrealistic to put the system in a different initial state every time? Shouldn’t we just run a full consecutive simulation of the system and use that as training data?’ The short answer to this is, ‘Yes, we should, and it would work, but we would not get pretty graphs.’

The reason here is that, during a full simulation, some states are more likely to be reached than others. In fact, there are many states that we will never reach at all, while other states may be visited multiple times. As a result, the algorithm will be very well capable of making predictions for some states, but highly incapable for other states. If we would then make a plot like Figure 2.15, which shows the predictions for *all* states, it would make it seem like the algorithm is worthless. There are states which it cannot make predictions for! As a result, the plot would look horrific, while in reality it only means that some states do not get visited, and hence we do not even have to make predictions for those states.

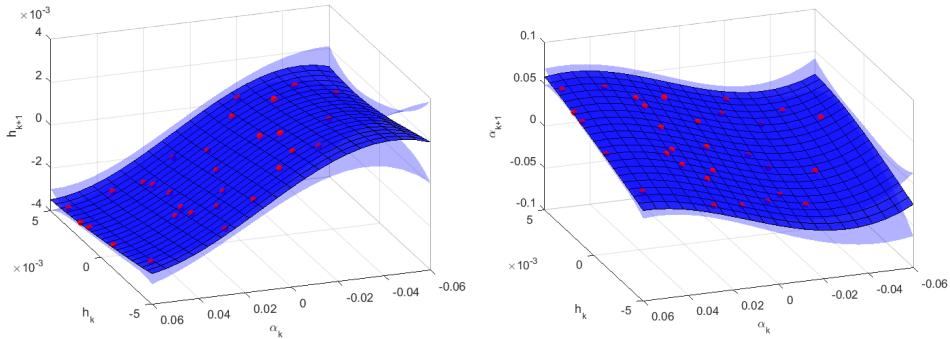


Figure 2.14: The prediction, as made by Gaussian process regression, of the next state of the pitch-plunge system, based on the current state. For each data point, the system was placed in a random position/orientation \mathbf{x}_k , kept stationary ($\dot{\mathbf{x}}_k = 0$) until it was released. After $\Delta t = 0.1$ s the next position \mathbf{x}_{k+1} was recorded. This was done $n_m = 30$ times at a constant wind speed of $U = 15$ m/s. The general shape of the function is almost identical to what we would get if we would have done many more measurements ($n_m = 300$; not shown here) although naturally the variance would be smaller then.

2.6.4. MAKING A HIGHER-DIMENSIONAL APPROXIMATION

Let's make the problem a bit more difficult. For every one of the n_m tests, we will now put the pitch-plunge system in a fully random initial state. So both \mathbf{x}_k and $\dot{\mathbf{x}}_k$ are randomly chosen. In addition, we also provide a random (constant) input β_k . The input points of the GP regression algorithm are hence five-dimensional. Even though the equations we need are exactly the same, this is computationally a quite more complex problem. When we apply the regression algorithm, we get the results shown in Figure 2.15.

From Figure 2.15 we can see that the predictions are a lot less accurate now. Though the GP regression algorithm detects the general shape of the function, it is very uncertain in its details. This uncertainty is also indicated by the algorithm itself through the larger uncertainty region.

Of course, when more experiments are performed, the certainty of the algorithm will increase. In fact, it takes about $n_m = 150$ experiments before the predictions of the five-dimensional problem will be as accurate as the predictions of the two-dimensional problem with $n_m = 30$ measurements. When we do use $n_m = 150$ measurements, the two plots look nearly identical.

To summarize, the GP regression seems very well capable of predicting the next state of a discrete-time system. In other words, GP regression can be used for nonlinear system identification. This does of course require the state to be fully and exactly known, so no partial state measurements or measurement noise is allowed. In Chapter 5 we will look into what to do when the state is subject to measurement noise as well. That is, when not just the output \mathbf{f}_m is noisy, but also the input \mathbf{x}_m .

2.7. LITERATURE – A SHORT HISTORY OF GP REGRESSION

Though few people realize it, Gaussian process regression has been applied for quite a long time. Its first significant applications were for time series analysis. The main theory here was developed by [Kolmogorov \(1941\)](#), [Wiener \(1949\)](#) in the 1940s. In the 1960s and

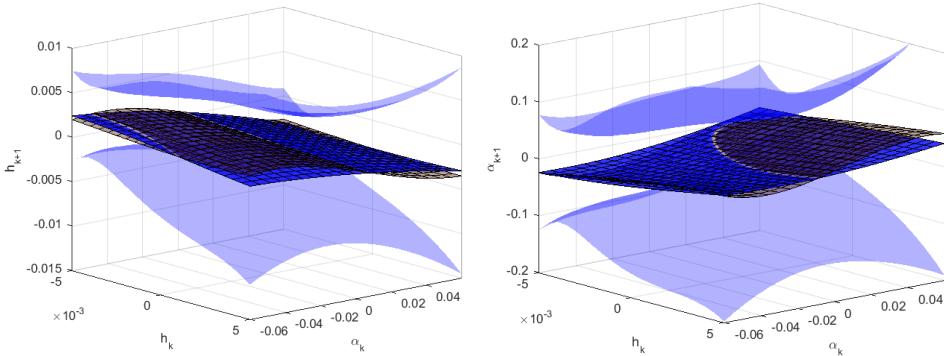


Figure 2.15: The prediction, as made by Gaussian process regression, of the next state of the pitch-plunge system, based on the current state. Also plotted, in an alternate color, is the mean function from Figure 2.14. The conditions are identical to Figure 2.14, except that now x_k , \dot{x}_k and β_k were all chosen at random while doing measurements. To make the above predictions, \dot{x}_k and β_k were still set to zero. Because of this difference, the measurement points could not be plotted anymore.

70s it was subsequently applied in meteorology by [Thompson \(1956\)](#), [Daley \(1991\)](#) and in geostatistics by [Matheron \(1973\)](#), [Journel and Huijbregts \(1978\)](#), although in the latter field it was commonly known as kriging.

Gradually, near the end of the 1970s, it became more well-understood that Gaussian process regression could be applied not only to time series analysis, but to regression problems in general. For instance, [O'Hagan and Kingman \(1978\)](#) present the general Gaussian process regression equations (2.30), though of course in a rather different form. But it was only in the 1990s when Gaussian process regression was finally applied to machine learning, mainly introduced by [Williams and Rasmussen \(1996\)](#), [Neal \(1996\)](#). It was at this time that the similarities with other machine learning techniques, like Support Vector Machines, Splines and Neural Networks, were investigated. Here it turned out that the statistical nature of Gaussian process regression gave it certain advantages, like a built-in regularization method.

The field converged with the publication of a single book by [Rasmussen and Williams \(2006\)](#), focusing on Gaussian processes for machine learning. The notation used in this book has become commonplace in the field, making it invaluable for the development of the Gaussian process community. This notation is also the one we will use in this thesis, albeit with a few extensions of our own.

Of course the field developed further after 2006, with various branches going into different directions. Several of these branches will be the subject of subsequent chapters.

3

DETAILS OF THE COVARIANCE FUNCTION

Summary—The parameters of the covariance function are known as hyperparameters θ . The true set of hyperparameters is generally unknown. Ideally we take into account all possible hyperparameters by integrating over them, but this is not analytically possible. Instead, we will choose a specific set of hyperparameters and simply assume they are the true ones. We can do this based on expert knowledge, or tune the hyperparameters by optimizing the a posteriori hyperparameter likelihood $p(\theta|X_m, \hat{f}_m)$. This tuning is often done in a gradient ascent manner.

Next to choosing hyperparameters, we also need to choose the covariance function itself. There are several possible covariance functions, including the squared exponential (smooth) covariance function, the piecewise smooth covariance function, the periodic covariance function, the linear covariance function and more. These functions can also be combined in various ways. Choosing which covariance function to use goes identically to tuning the hyperparameters. We either use expert knowledge, or optimize the likelihood.

Previously, we have always assumed that we measured the function values $f = f(X)$ itself and used those for our regression equations. If instead we measure linear relationships $Mf = c$ of function values, we can still apply Gaussian process regression. We just need to incorporate the matrix M into our regression equations. When we do, we find a generalization of the regular Gaussian process regression equations that offers a variety of extra possibilities.

All these techniques can be applied to practical applications. Specifically, they can be applied to identify the dynamics of a pitch-plunge system, or to approximate the value function of this system. In the latter case, the value function can then be optimized to tune the controller settings.

In this chapter we look at three different tricks that we can use to make GP regression more effective in certain situations. We start by looking at how we should choose parameters like λ_f , λ_x and $\hat{\sigma}_{f_m}$; that is, the hyperparameters of the covariance function (Section 3.1). Then we look into using different covariance functions $k(\mathbf{x}, \mathbf{x}')$ altogether (Section 3.2). The third trick we examine is how to apply Gaussian process regression when we only measure linear relations between function values (Section 3.3).

Afterwards we also apply the tricks we learned to a practical application, either identifying the system dynamics of the pitch-plunge system (Section 3.4) or approximating its value function (Section 3.5). As usual, at the back you can find an overview of literature and contributions (Section 3.6).

3.1. THE BASICS OF TUNING HYPERPARAMETERS

In Gaussian process regression, there are several parameters that need to be ‘chosen’. For instance, there are the length scales λ_x and λ_f , as well as the noise scale $\hat{\sigma}_{f_m}$. In case we choose a constant mean function $m(\mathbf{x}) = \bar{m}$, then this constant \bar{m} needs to be chosen as well.

All these parameters are called *hyperparameters*. They have a significant effect on the predictions that we make, as shown by Figure 3.1. The set of all hyperparameters is denoted by the vector $\boldsymbol{\theta}$. We can choose them ourselves, based on our expert knowledge of the function we are approximating, or we can tune them automatically. In this chapter we will look at how the latter works.

We start by looking at how to find the likelihood of a set of hyperparameters (Section 3.1.1). We then examine two methods to implement this, either by integrating over the possible hyperparameters (Section 3.1.2) or by taking the most likely hyperparameters (Section 3.1.3). Then we look at how we can implement the latter method in practice (Section 3.1.4) and finally how we can vary the main assumptions behind the method (Section 3.1.5).

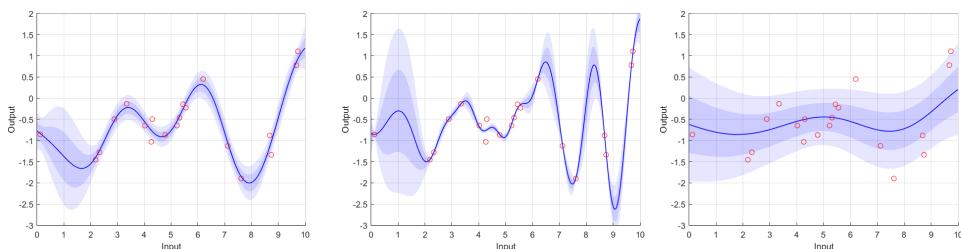


Figure 3.1: Gaussian process regression for different hyperparameters. Data was generated using a GP with $[\lambda_x, \lambda_f, \hat{\sigma}_{f_m}] = [1, 1, \frac{1}{5}]$. It was subsequently approximated using a GP with $[\lambda_x, \lambda_f, \hat{\sigma}_{f_m}]$ equaling the same $[1, 1, \frac{1}{5}]$ (left), $[\frac{1}{2}, 1, \frac{1}{25}]$ (middle) and $[2, 1, 1]$ (right). While the first approximation seems sensible, the second approximation explains everything through function variations and the third approximation explains everything through noise. Both results are not very good. This is also shown by the log-likelihoods (introduced in Section 3.1.3), which are -14.1 , -72.2 and -26.1 , respectively. Hyperparameter tuning (see Section 3.1.4) will, even when starting from completely incorrect hyperparameters, result in hyperparameters of $[0.925, 0.999, 0.201]$ and a corresponding log-likelihood of -13.2 .

3.1.1. PROBABILITIES AND LIKELIHOODS

The key realization we should first make, is that we do not know the set of hyperparameters $\boldsymbol{\theta}$, and hence we should treat it as a random variable $\underline{\boldsymbol{\theta}}$. Now we can look at the probability that a certain set of hyperparameters $\boldsymbol{\theta}$ is the correct one. We write this as $p(\underline{\boldsymbol{\theta}} = \boldsymbol{\theta})$ or short as $p(\boldsymbol{\theta})$.

We should keep in mind here that we also have data. To be precise, we know X_m and \hat{f}_m . So actually, we should find the probability that $\underline{\boldsymbol{\theta}} = \boldsymbol{\theta}$ given the values of X_m and \hat{f}_m . We write this as $p(\boldsymbol{\theta}|X_m, \hat{f}_m)$. *Bayes' theorem* (or alternatively the definition of the conditional distribution) tells us that, for certain events A, B and C , we have

$$p(A|B, C)p(B|C) = p(A, B|C) = p(B|A, C)p(A|C). \quad (3.1)$$

Using this, we can now find that

$$p(\boldsymbol{\theta}|\hat{f}_m, X_m) = \frac{p(\hat{f}_m|\boldsymbol{\theta}, X_m)p(\boldsymbol{\theta}|X_m)}{p(\hat{f}_m|X_m)}. \quad (3.2)$$

This is an important relation. It has four quantities which we will examine one by one.

The first quantity $p(\boldsymbol{\theta}|\hat{f}_m, X_m)$ is known as the *posterior hyperparameter distribution*. It is what we want to know. Or at least, we want to know which hyperparameters $\boldsymbol{\theta}$ have a high probability.

The second term $p(\hat{f}_m|\boldsymbol{\theta}, X_m)$ is called the *observation likelihood*, or short, just the *likelihood*. It is the probability that, given certain hyperparameters $\boldsymbol{\theta}$, we made the observations/measurements that we did. We already knew, before making any measurements, that $\underline{\hat{f}_m}$ was distributed according to (2.26). The probability that we obtained our measurements \hat{f}_m is hence equal to

$$p(\hat{f}_m|\boldsymbol{\theta}, X_m) = \mathcal{N}\left(\hat{f}_m|\mathbf{m}_m, K_{mm} + \hat{\Sigma}_{f_m}\right). \quad (3.3)$$

Note that K_{mm} , $\hat{\Sigma}_{f_m}$ and possibly even \mathbf{m}_m depend on the hyperparameters $\boldsymbol{\theta}$.

The third term $p(\boldsymbol{\theta}|X_m)$ is the *prior hyperparameter distribution*, or short, the *hyper-prior*. This probability actually does not depend on X_m . In fact, only knowing X_m does not tell us anything about $\boldsymbol{\theta}$. It hence equals $p(\boldsymbol{\theta})$.

We can use the hyper-prior to indicate which hyperparameters we roughly expect to get. In practice, we often don't really know much in advance about the hyperparameters. For simplicity, we hence assume that $p(\boldsymbol{\theta})$ is constant. Later on, in Section 3.1.5, we will look at other hyper-priors.

Finally, the fourth quantity is the denominator $p(\hat{f}_m|X_m)$. It is called the *marginal likelihood*, but since it does not depend on $\boldsymbol{\theta}$, it is a constant too.

Putting all this together, we find that the probability $p(\boldsymbol{\theta}|\hat{f}_m, X_m)$ is proportional to

$$p(\boldsymbol{\theta}|\hat{f}_m, X_m) \propto \mathcal{N}\left(\hat{f}_m|\mathbf{m}_m, K_{mm} + \hat{\Sigma}_{f_m}\right). \quad (3.4)$$

So how do we continue? There are two options now, which we will look at in the next two subsections.

3.1.2. INTEGRATING OVER HYPERPARAMETERS

Suppose that we know the true hyperparameters $\boldsymbol{\theta}$. In that case, we can use (2.30) to predict \underline{f}_* . In fact, this distribution tells us, given the hyperparameters $\boldsymbol{\theta}$ and the measurements $X_m, \hat{\mathbf{f}}_m$, the chance that \underline{f}_* equals a given value f_* . This means that another way to write (2.30) is as the probability

$$p(f_* | \boldsymbol{\theta}, \hat{\mathbf{f}}_m, X_m) = \mathcal{N}(f_* | \boldsymbol{\mu}_*, \Sigma_{**}). \quad (3.5)$$

3

Technically, using this notation is incorrect, since we are working with probability density functions and not with probabilities, but we will ignore that detail to keep the notation simple.

Now suppose that there are two possible sets of hyperparameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ that both explain our measurements pretty well. They are both correct with probability 1/2. In this case, we can take both of them into account by adding up the probabilities. That is,

$$p(f_* | \hat{\mathbf{f}}_m, X_m) = \frac{1}{2} p(f_* | \boldsymbol{\theta}_1, \hat{\mathbf{f}}_m, X_m) + \frac{1}{2} p(f_* | \boldsymbol{\theta}_2, \hat{\mathbf{f}}_m, X_m). \quad (3.6)$$

The extended version of this idea is known as the principle of marginalization. (See for instance Theorem B.1.) We can find the posterior distribution of \underline{f}_* , taking into account all possible hyperparameters, using

$$\begin{aligned} p(f_* | \hat{\mathbf{f}}_m, X_m) &= \int_{\Theta} p(f_*, \boldsymbol{\theta} | \hat{\mathbf{f}}_m, X_m) d\boldsymbol{\theta} \\ &= \int_{\Theta} p(f_* | \boldsymbol{\theta}, \hat{\mathbf{f}}_m, X_m) p(\boldsymbol{\theta} | \hat{\mathbf{f}}_m, X_m) d\boldsymbol{\theta}. \end{aligned} \quad (3.7)$$

We know both the probabilities in the final expression (at least up to a constant). They are given by (3.5) and (3.4), respectively. So in theory we can solve this.

The main problem here is that the result will not be Gaussian. In fact, the way in which these probabilities depend on the hyperparameters $\boldsymbol{\theta}$ is very complicated. Solving the above integral analytically will be impossible, and the result surely will not be a Gaussian process anymore.

One way to work around this would be to use numerical methods. That would be beyond the scope of this explanation, but for further reading, you can start with the work of Svensson et al. (2015) or Murray and Adams (2010). In most applications another method is used instead, which we will explore next.

3.1.3. THE MAXIMUM LIKELIHOOD METHOD

Instead of taking into account *all* possible hyperparameters $\boldsymbol{\theta}$, the idea now is to find the *most likely* hyperparameters and only use those.

There are two common ways to do this. The first is the *Maximum Likelihood method* (ML method). The idea here is to find the hyperparameters $\boldsymbol{\theta}$ that optimize the likelihood $p(\underline{f}_m | \boldsymbol{\theta}, X_m)$. That is, the hyperparameters that best explain the measurements.

A similar but slightly more ‘honest’ method is the *Maximum A Posteriori method* (MAP method). Here, we want to maximize the posterior hyperparameter distribution $p(\boldsymbol{\theta} | \hat{\mathbf{f}}_m, X_m)$ from (3.4). However, we often assume that $\boldsymbol{\theta}$ has a uniform distribution and

hence that the hyper-prior $p(\boldsymbol{\theta})$ is constant. In this case $p(\boldsymbol{\theta}|\hat{\mathbf{f}}_m, X_m)$ is only a constant multiple of $p(\underline{\mathbf{f}}_m|\boldsymbol{\theta}, X_m)$. So as long as $p(\boldsymbol{\theta})$ is constant, the ML method and the MAP method do exactly the same. Hence, we will simply apply the MAP method with $p(\boldsymbol{\theta})$ constant.

The key now is to find the maximum of the posterior hyperparameter distribution

$$p(\boldsymbol{\theta}|\hat{\mathbf{f}}_m, X_m) \propto \frac{1}{\sqrt{|2\pi(K_{mm} + \hat{\Sigma}_{f_m})|}} \exp\left(-\frac{1}{2} (\hat{\mathbf{f}}_m - \mathbf{m}_m)^T (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m)\right). \quad (3.8)$$

The exponent makes maximizing this somewhat difficult. To solve this issue, we take the logarithm of the above function. Because the logarithm is a strictly ascending function, this again does not affect the position of the maximum. When we take the logarithm and work out the result, we get the *log-likelihood*

$$\begin{aligned} \log(p) = & -\frac{n_m}{2} \log(2\pi) - \frac{1}{2} \log|K_{mm} + \hat{\Sigma}_{f_m}| \\ & - \frac{1}{2} (\hat{\mathbf{f}}_m - \mathbf{m}_m)^T (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m) + \log(c), \end{aligned} \quad (3.9)$$

where c is the (not important) proportionality constant from (3.8). The first term in the above expression is also a *normalization constant*. Since both these terms are constant, we can ignore them in our optimization process. The third term is called the *data fit*. It describes how well our measurements $\hat{\mathbf{f}}_m$ fit with our hyperparameters. If for instance $\hat{\mathbf{f}}_m$ is very close to \mathbf{m}_m , then the magnitude of the data fit term will be very small. But, because there is a minus sign in front of it, the data fit itself will be large, or at least not strongly negative.

Another way to obtain a good data fit, is to give the matrix $K_{mm} + \hat{\Sigma}_{f_m}$ huge values. This comes down to assuming there is so much noise, that any data fits within our model. Luckily, the second term in the log-likelihood expression prevents that problem. We call this term the *complexity penalty*. When we use a large matrix $K_{mm} + \hat{\Sigma}_{f_m}$, this term will become highly negative, reducing the log-likelihood.

The great thing about this complexity penalty is that, unlike other methods like neural networks, Gaussian process regression has a far smaller risk of *overfitting*¹. It has an automatic *regularization*² built into it through its foundation in Bayesian probability theory.

Although to be fair, it must be noted that overfitting is still possible when using the maximum likelihood method. Suppose that only very few measurements are available. In this case, we cannot really be sure yet which hyperparameters are the correct ones. In other words, the uncertainty within $\boldsymbol{\theta}$ is large. When we would integrate over all possible hyperparameters, we would take this into account, resulting in a large uncertainty for predictions $\underline{\mathbf{f}}_*$, and rightfully so. However, the maximum likelihood method only takes

¹Overfitting means that a regression algorithm is trained so much on a particular data set, that it not only manages to explain the measured data, but also ‘explain’ the noise that occurred for that particular data set. Naturally, when given a different data set, the noise will be different, so this is a bad thing.

²Regularization is a trick that introduces extra information (effectively, a prior distribution) to the system to prevent overfitting.

the hyperparameters θ that are the most likely, and then claims it is 100% certain that these are the correct hyperparameters. This unjustly results in a relatively small variance of our predictions. That is why, especially when few measurement data is available, the maximum likelihood method may result in some overfitting, claiming it is more certain of its predictions than it has a right to be.

3.1.4. OPTIMIZING THE LOG-LIKELIHOOD

The next question is: how do we find the maximum of the log-likelihood? There are many ways to do so. One option is to just plug the log-likelihood into an automatic optimization function of for instance Matlab. For a more efficient process, we could set up our own gradient ascent method. Though to do that, we would need the derivative of $\log(p)$ with respect to the hyperparameters.

First, let's define the shorthand $P = (K_{mm} + \hat{\Sigma}_{f_m})$. We now have

$$\log(p) = -\frac{n_m}{2} \log(2\pi) - \frac{1}{2} \log|P| - \frac{1}{2} (\hat{\mathbf{f}}_m - \mathbf{m}_m)^T P^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m) - \log(c). \quad (3.10)$$

For now, we will ignore hyperparameters effecting \mathbf{m}_m . So we only consider hyperparameters affecting P . Let's consider the derivative with respect to some general hyperparameter θ_i . (We will insert specific hyperparameters afterwards.) Using Theorem A.2 as well as relation (A.9), we can find that

$$\frac{\partial \log(p)}{\partial \theta_i} = -\frac{1}{2} \text{tr}\left(P^{-1} \frac{\partial P}{\partial \theta_i}\right) + \frac{1}{2} (\hat{\mathbf{f}}_m - \mathbf{m}_m)^T P^{-1} \frac{\partial P}{\partial \theta_i} P^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m). \quad (3.11)$$

Next, we will rewrite the above. We start by defining $\alpha = P^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m)$. Then we also take the trace of the rightmost term. This allows us to cycle the order of multiplication (see Theorem A.1) so we wind up with

$$\begin{aligned} \frac{\partial \log(p)}{\partial \theta_i} &= -\frac{1}{2} \text{tr}\left(P^{-1} \frac{\partial P}{\partial \theta_i}\right) + \frac{1}{2} \text{tr}\left(\alpha^T \frac{\partial P}{\partial \theta_i} \alpha\right) \\ &= \frac{1}{2} \text{tr}\left((\alpha \alpha^T - P^{-1}) \frac{\partial P}{\partial \theta_i}\right). \end{aligned} \quad (3.12)$$

Now we just have to find $\partial P / \partial \theta_i$ for various hyperparameters. Which hyperparameters we have does depend on the covariance function that we are using. (The above holds for any covariance function we might use.) For now, let's assume we are using the squared exponential covariance function (2.35). We will look at other covariance functions in Section 3.2.

We will start with the derivative with respect to $\hat{\sigma}_{f_m}^2$, where we assume that the measurement noise has the same strength $\hat{\sigma}_{f_{m1}}^2 = \dots = \hat{\sigma}_{f_{mn}}^2 = \hat{\sigma}_{f_m}^2$ for each measurement. Note that we take the derivative with respect to $\hat{\sigma}_{f_m}^2$ and not with respect to $\hat{\sigma}_{f_m}$. The reason is that this will result in slightly easier expressions, though it is also perfectly possible to take the derivative with respect to $\hat{\sigma}_{f_m}$. We now have

$$\frac{\partial P}{\partial \hat{\sigma}_{f_m}^2} = \frac{\partial (K_{mm} + \hat{\Sigma}_{f_m})}{\partial \hat{\sigma}_{f_m}^2} = \frac{\partial \hat{\Sigma}_{f_m}}{\partial \hat{\sigma}_{f_m}^2} = I. \quad (3.13)$$

Note that we have used $\hat{\Sigma}_{fm} = \hat{\sigma}_{fm}^2 I$. Inserting the above relation into (3.12) will now give us $\partial \log(p) / \partial \hat{\sigma}_{fm}^2$.

Next, we take the derivative with respect to λ_f^2 . Here we have

$$\frac{\partial P}{\partial \lambda_f^2} = \frac{\partial K_{mm}}{\partial \lambda_f^2} = \frac{\partial k(X_m, X_m)}{\partial \lambda_f^2} = \frac{K_{mm}}{\lambda_f^2}. \quad (3.14)$$

In the last part, we have used that $k(\mathbf{x}, \mathbf{x}')$ actually linearly depends on λ_f^2 . As a result, $\partial k(\mathbf{x}, \mathbf{x}') / \partial \lambda_f^2$ would equal $k(\mathbf{x}, \mathbf{x}') / \lambda_f^2$.

The next derivative is the hardest one. We will take the derivative with respect to $\lambda_{x_k}^2$, which is the squared length scale in the direction of input x_k . Note that we have one such derivative for each of the d_x input dimensions. We will find these derivatives element-wise. So,

$$\frac{\partial P_{ij}}{\partial \lambda_{x_k}^2} = \frac{\partial k(\mathbf{x}_{m_i}, \mathbf{x}_{m_j})}{\partial \lambda_{x_k}^2} = \frac{\partial}{\partial \lambda_{x_k}^2} \left(\lambda_f^2 \exp \left(-\frac{1}{2} (\mathbf{x}_{m_i} - \mathbf{x}_{m_j})^T \Lambda_x^{-1} (\mathbf{x}_{m_i} - \mathbf{x}_{m_j}) \right) \right). \quad (3.15)$$

Using the chain rule and Theorem A.2, we can find that the above equals

$$\begin{aligned} \frac{\partial P_{ij}}{\partial \lambda_{x_k}^2} &= k(\mathbf{x}_{m_i}, \mathbf{x}_{m_j}) \frac{\partial}{\partial \lambda_{x_k}^2} \left(-\frac{1}{2} (\mathbf{x}_{m_i} - \mathbf{x}_{m_j})^T \Lambda_x^{-1} (\mathbf{x}_{m_i} - \mathbf{x}_{m_j}) \right) \\ &= k(\mathbf{x}_{m_i}, \mathbf{x}_{m_j}) \left(\frac{1}{2} (\mathbf{x}_{m_i} - \mathbf{x}_{m_j})^T \Lambda_x^{-1} \frac{\partial \Lambda_x}{\partial \lambda_{x_k}^2} \Lambda_x^{-1} (\mathbf{x}_{m_i} - \mathbf{x}_{m_j}) \right). \end{aligned} \quad (3.16)$$

It is important to consider what the derivative $\partial \Lambda_x / \partial \lambda_{x_k}^2$ looks like. It is a matrix filled with zeros, except for a single one, which is on row k and in column k . Keeping this in mind, we can simplify the above to

$$\frac{\partial P_{ij}}{\partial \lambda_{x_k}^2} = \frac{1}{2} k(\mathbf{x}_{m_i}, \mathbf{x}_{m_j}) \left(\frac{x_{m_i}^k - x_{m_j}^k}{\lambda_{x_k}^2} \right)^2. \quad (3.17)$$

Note that the term $x_{m_i}^k$ is element k from the vector \mathbf{x}_{m_i} .

Using all these derivatives, we can apply our favorite gradient ascent algorithm to optimize the log-likelihood. This then gives us the most likely hyperparameters to represent our measurement data.

In literature, when we tune the hyperparameters to specifically find $\lambda_{x_1}, \lambda_{x_2}, \dots$, it is also called *Automatic Relevance Determination* (ARD). The idea here is that the tuned parameters $\lambda_{x_1}, \lambda_{x_2}, \dots$ tell us something about the relevance of certain input parameters. If λ_{x_1} is very small, then x^1 has a very strong affect on the predicted output. A small variation in x^1 can already cause a significant change. However, if λ_{x_2} is very big, then the value of x^2 is pretty much irrelevant. As such, the values of λ_x indicate the relevance of the corresponding input dimensions.

Finally, there is one more hyperparameter we can tune. Let's suppose that the mean function $m(\mathbf{x})$ equals some constant \bar{m} . What value of \bar{m} maximizes the log-likelihood?

We will have to start from (3.10) again. It is important to realize that P does not depend on \bar{m} , but \mathbf{m}_m does. In fact, we have $\mathbf{m}_m = \bar{m}\mathbf{1}$, where $\mathbf{1}$ is a vector filled with ones. So,

$$\frac{\partial \log(p)}{\partial \bar{m}} = \frac{\partial}{\partial \bar{m}} \left(-\frac{1}{2} \left(\hat{\mathbf{f}}_m - \mathbf{m}_m \right)^T P^{-1} \left(\hat{\mathbf{f}}_m - \mathbf{m}_m \right) \right) = \mathbf{1}^T P^{-1} \left(\hat{\mathbf{f}}_m - \mathbf{m}_m \right). \quad (3.18)$$

The special thing here is that we can analytically find the optimal \bar{m} . This optimum occurs when the above derivative equals zero. So by setting the above to zero, while substituting \mathbf{m}_m for $\bar{m}\mathbf{1}$ and solving for \bar{m} , we get

$$\bar{m} = \frac{\mathbf{1}^T P^{-1} \hat{\mathbf{f}}_m}{\mathbf{1}^T P^{-1} \mathbf{1}}. \quad (3.19)$$

As a result, it is useful to always set \bar{m} to the above value, every time the other hyperparameters are adjusted or more measurement data is added. It is even possible to take this new constant value into account in the expression for the log-likelihood, although that would go into too much detail. If you are interested, you can read more about tuning \bar{m} in the thesis of [Lizotte \(2008\)](#), Section 3.1.

An example of the development of the GP prediction, as hyperparameters are tuned, is shown in Figure 3.2.

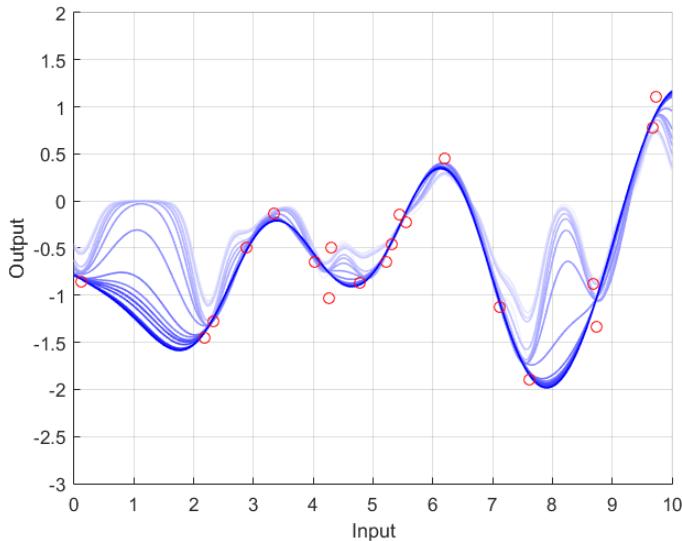


Figure 3.2: The mean of the prediction while tuning the hyperparameters. We use the same data as Figure 3.1, but start with hyperparameters $[\lambda_x, \lambda_f, \hat{\sigma}_{f_m}]$ equal to the blatantly incorrect values $[\frac{1}{5}, 5, 2]$. We then tune them using a gradient ascent algorithm. The first few steps are shown, although after ten steps the algorithm has pretty much found the correct values. The algorithm converged to values of $[0.925, 0.999, 0.201]$, with $\bar{m} = -0.57$. The corresponding log-likelihood was -13.2 .

3.1.5. USING DIFFERENT HYPER-PRIORS

So far we have assumed that the hyper-prior $p(\boldsymbol{\theta})$ is a constant. Or to be precise, we have assumed that the prior probability density function $f_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ equals a constant. So the hyperparameters can have every possible value with equal likelihood. But in reality that does not seem very realistic.

The first reason is that most hyperparameters are length scales, and these length scales are always positive. So we actually do not have to consider negative values at all. In addition, when we use our constant hyper-prior, we basically say that the probability that $0 < \underline{\theta}_i \leq 1$, for some hyperparameter $\underline{\theta}_i$, is just as large as the probability that $1 < \underline{\theta}_i \leq 2$. This is also doubtful.

It would be more sensible if $p(0.1 \leq \underline{\theta}_i \leq 1)$ would equal $p(1 \leq \underline{\theta}_i \leq 10)$, and identically for other similar intervals. This comes down to assuming that $\log(\underline{\theta}_i)$ has a constant distribution $f_{\log(\underline{\theta}_i)}(\log(\underline{\theta}_i)) = \gamma$, or equivalently (see Theorem B.8) that θ_i has the PDF

$$f_{\underline{\theta}_i}(\theta_i) = \frac{1}{\theta_i} f_{\log(\underline{\theta}_i)}(\log(\theta_i)) = \frac{1}{\theta_i} \gamma, \quad (3.20)$$

with γ an infinitesimally small constant. So the PDF of $\underline{\theta}_i$ is proportional to $1/\theta_i$. This is for a single hyperparameter. Combining this for all hyperparameters will result in the hyper-prior

$$p(\boldsymbol{\theta}) = \gamma^{n_\theta} \prod_{i=1}^{n_\theta} \frac{1}{\theta_i}, \quad (3.21)$$

where n_θ is the number of hyperparameters we need to choose.

To implement this hyper-prior, we should redo all the steps we did previously, calculating the log-likelihood $\log(p)$. Similarly to (3.10), we now get

$$\begin{aligned} \log(p) = & -\frac{n_m}{2} \log(2\pi) - \frac{1}{2} \log|P| - \frac{1}{2} \left(\hat{\mathbf{f}}_m - \mathbf{m}_m \right)^T P^{-1} \left(\hat{\mathbf{f}}_m - \mathbf{m}_m \right) \\ & + n_\theta \log(\gamma) - \sum_{i=1}^{n_\theta} \log(\theta_i) + \log(c). \end{aligned} \quad (3.22)$$

So the main thing our new hyper-prior does is discount high values of θ_i , while making low values of θ_i more likely. The derivative of the above is, identically to (3.12),

$$\frac{\partial \log(p)}{\partial \theta_i} = \frac{1}{2} \text{tr} \left((\boldsymbol{\alpha} \boldsymbol{\alpha}^T - P^{-1}) \frac{\partial P}{\partial \theta_i} \right) - \frac{1}{\theta_i}. \quad (3.23)$$

The only difference is the $-1/\theta_i$ term at the end. This term makes the derivative smaller, which again makes lower values of θ_i more likely than higher values.

In general, the changes are not spectacular. The hyperparameters will turn out to be somewhat smaller than earlier, but that's pretty much it. This hyper-prior does show, however, that there are different options when selecting the hyper-prior.

Next to this hyper-prior that we have examined, there are of course also more specific hyper-priors you can use, if you really have prior knowledge about $\boldsymbol{\theta}$. That is too detailed for this explanation though, so I will leave that for you to explore on your own.

3.2. OTHER COVARIANCE FUNCTIONS AND TUNING METHODS

So far we have only examined the squared exponential covariance function (2.35). This covariance function is useful for smooth functions in general. But what do we do when the function we want to approximate is not smooth and for example has jumps? Or when we know more about the structure of our function other than just that it is smooth? Then we can also use different covariance functions, and that's what we will look at in this section.

3

We will start this section with some basic covariance functions (Section 3.2.1). Then we continue by looking in-depth at the covariance function for linear functions (Section 3.2.2). Once we know a sufficient number of these basic covariance functions, we look at how we can make a trade-off between covariance functions (Section 3.2.3) and/or combine covariance functions to make a better covariance function (Section 3.2.4).

3.2.1. DIFFERENT KINDS OF COVARIANCE FUNCTIONS

Suppose that we have a piecewise smooth function $f(\mathbf{x})$. (See Figure 3.3 for examples.) To be precise, there are certain regions $\mathcal{D}_1, \mathcal{D}_2, \dots$ dividing the input space of \mathbf{x} . As long as the input \mathbf{x} stays within the same region \mathcal{D}_i , the function $f(\mathbf{x})$ is smooth, but when \mathbf{x} moves to a different region, a jump in the function value $f(\mathbf{x})$ can occur. How would we approximate such a function?

The key realization here is that, when two inputs \mathbf{x} and \mathbf{x}' are in the same region \mathcal{D}_i , then their values will be similar, especially if they are close together. But when \mathbf{x} and \mathbf{x}' are in different regions, then their values are not linked at all. Their covariance is zero. We can hence use the *piecewise smooth covariance function*

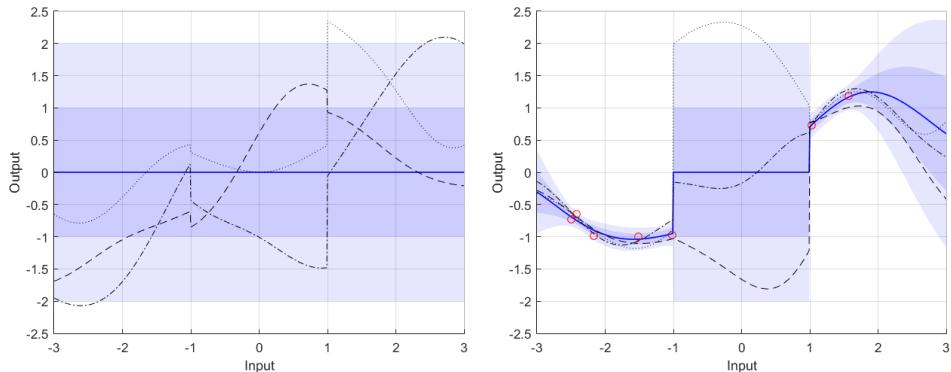
$$k_{\text{ps}}(\mathbf{x}, \mathbf{x}') = \begin{cases} \lambda_f^2 \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}')^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}')\right) & \text{if } \mathbf{x} \in \mathcal{D}_i \text{ and } \mathbf{x}' \in \mathcal{D}_i \text{ for some } \mathcal{D}_i, \\ 0 & \text{otherwise.} \end{cases} \quad (3.24)$$

Example functions that result from such a covariance function are shown in Figure 3.3. The boundaries between the regions here can either be known in advance, or tuned using the techniques from Section 3.1. Although if you want to do the latter, you will have to derive the hyperparameter derivatives yourself.

Next, let's consider a very different problem. Suppose that we want to approximate a smooth and *periodic* function $f(x)$, with known period p . This knowledge that the function is periodic is very useful. It can prevent us from having to do too many measurements. So how can we take it into account?

The obvious but incorrect way to do so, would be to shift the input points x_m of all measurements \hat{f}_m to the interval $[0, p]$ and then apply GP regression on this interval. The reason why this is incorrect, is because we do not take into account that $f(0) = f(p)$, nor that the function values $f(0.01)$ and $f(p-0.01)$ are also quite strongly linked (though not exactly equal).

The correct way is to set up a covariance function that not only links (correlates) $f(x)$ with its own nearby values, but also with nearby values of $f(x+p)$, $f(x+2p)$, and so



3

Figure 3.3: A Gaussian process with a piecewise smooth covariance function. Input points satisfying $x < -1$ are not correlated with points satisfying $-1 \leq x < 1$, which again are not correlated with points satisfying $1 \leq x$. On the left is the prior distribution, with three sample functions taken from it. These samples tell us which kind of functions the covariance function can approximate. On the right some measurement data has been added, and again three sample functions are shown. These samples all correspond to some degree with the given data.

forth. A covariance function that does this is

$$k_{\text{per}}(x, x') = \lambda_f^2 \exp\left(\frac{-2 \sin\left(\pi \frac{x-x'}{p}\right)^2}{\lambda_x^2}\right). \quad (3.25)$$

Of course the parameter p can also be tuned, like the other hyperparameters, although we should be careful not to find $2p$, $3p$ or $4p$ or so, instead of p itself. An example in which this covariance function is applied is shown in Figure 3.4.

There are many more covariance functions available in literature. Discussing all of them in detail is a whole subject of its own, which is not directly relevant to us now. For further information, you can consult the book of Rasmussen and Williams (2006) (Chapter 4). Although there is one covariance function that I still want to look at.

3.2.2. THE COVARIANCE FUNCTION FOR LINEAR FUNCTIONS

Let's suppose that we want to approximate a linear function $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ (or equivalently $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$). The *weight vector* \mathbf{w} is unknown, and hence we will treat it as a random variable $\underline{\mathbf{w}}$ with prior distribution $\underline{\mathbf{w}} \sim \mathcal{N}(\mathbf{0}, K_w)$. What kind of covariance function should we use now?

THE LINEAR MEAN AND COVARIANCE FUNCTION

In this special case, we can calculate the prior mean and covariance function. We have

$$m_{\text{lin}}(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] = \mathbb{E}[\underline{\mathbf{w}}^T \mathbf{x}] = \mathbb{E}[\underline{\mathbf{w}}^T] \mathbf{x} = \mathbf{0}^T \mathbf{x} = 0, \quad (3.26)$$

$$k_{\text{lin}}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})])(f(\mathbf{x}') - \mathbb{E}[f(\mathbf{x}')])] = \mathbb{E}[\mathbf{x}^T \underline{\mathbf{w}} \underline{\mathbf{w}}^T \mathbf{x}'] = \mathbf{x}^T K_w \mathbf{x}'. \quad (3.27)$$

These are the mean and covariance function for approximating linear functions. We often call this covariance function the *linear covariance function*. This does not mean

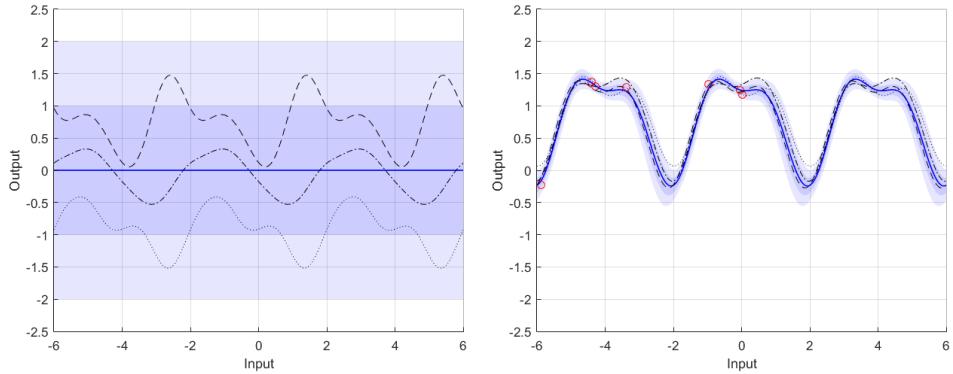


Figure 3.4: A Gaussian process with a smooth periodic covariance function. The period is set to $p = 4$. On the left is the prior distribution, with three samples. On the right is the posterior distribution after a few measurements have been performed. It is interesting to see that the rule ‘the further away you are from a measurement point, the bigger the uncertainty becomes’ now does not hold.

that the covariance function itself is linear (it is certainly not) but that it is the covariance function used for approximating linear functions.

If we would now have a measurement set X_m with corresponding noisy measurements \hat{f}_m , as well as a trial set X_* , then the posterior distribution of \underline{f}_* would become, according to (2.30),

$$\underline{f}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_{**}), \quad (3.28)$$

$$\begin{aligned} \Sigma_{**} &= K_{**} - K_{*m} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{m*} = X_*^T K_w X_* - X_*^T K_w X_m (X_m^T K_w X_m + \hat{\Sigma}_{f_m})^{-1} X_m^T K_w X_*, \\ \boldsymbol{\mu}_* &= K_{*m} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} \hat{f}_m = X_*^T K_w X_m (X_m^T K_w X_m + \hat{\Sigma}_{f_m})^{-1} \hat{f}_m. \end{aligned}$$

An example of such an approximation is shown in Figure 3.5 (left).

PREDICTING THE WEIGHTS w

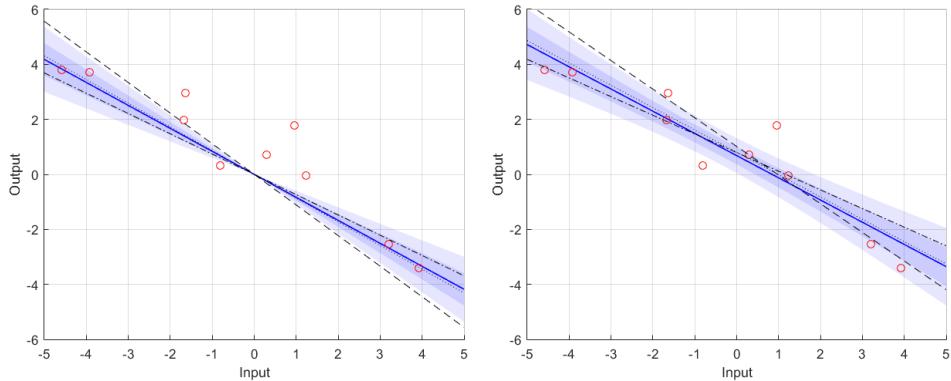
It may also be interesting to know what the posterior distribution of the weights \underline{w} is. This requires more mathematics, but according to Theorem B.27 the result will be

$$\underline{w} \sim \mathcal{N}(\boldsymbol{\mu}_w, \Sigma_w) = \mathcal{N}\left(\left(X_m \hat{\Sigma}_{f_m}^{-1} X_m^T + K_w^{-1}\right)^{-1} X_m \hat{\Sigma}_{f_m}^{-1} \hat{f}_m, \left(X_m \hat{\Sigma}_{f_m}^{-1} X_m^T + K_w^{-1}\right)^{-1}\right). \quad (3.29)$$

Knowing this, we can also find the posterior distribution of \underline{f}_* by using $\underline{f}_* = X_*^T \underline{w}$. It follows (applying Theorem B.3) that

$$\begin{aligned} \underline{f}_* &\sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_{**}), \\ \Sigma_{**} &= X_*^T \Sigma_w X_* = X_*^T \left(X_m \hat{\Sigma}_{f_m}^{-1} X_m^T + K_w^{-1}\right)^{-1} X_*, \\ \boldsymbol{\mu}_* &= X_*^T \boldsymbol{\mu}_w = X_*^T \left(X_m \hat{\Sigma}_{f_m}^{-1} X_m^T + K_w^{-1}\right)^{-1} X_m \hat{\Sigma}_{f_m}^{-1} \hat{f}_m. \end{aligned} \quad (3.30)$$

These expressions are actually the same expressions as when we would have used a weighted least-squares algorithm, with $\hat{\Sigma}_{f_m}^{-1}$ as the weight matrix (where more accurate



3

Figure 3.5: A Gaussian process with a linear covariance function. Three sample functions have been plotted from the posterior GP as well. To generate measurements, we used $w = 0.8$, $\sigma_{f_m} = 1$ and no offset b . To approximate this, we used $K_w = 1$. For the right figure, we added a possible offset to the algorithm as well (even though there isn't one) with $k_b = 4$. The posterior distribution of w was $w \sim \mathcal{N}(-0.84, 0.12^2)$ (left) and $w \sim \mathcal{N}(-0.81, 0.12^2)$ (right) while the posterior distribution of b was $b \sim \mathcal{N}(0.68, 0.32^2)$ (right). With more measurements, the means of these distributions will become closer to the true value, while the variances will decrease.

measurements result in higher weights) and K_w^{-1} as regularization term (where regularization comes down to taking into account prior knowledge). So by using GP regression with the linear covariance function, we are actually applying the least-squares method, except that we now intuitively understand what the weight matrix and the regularization matrix stand for.

At this point you may be comparing (3.30) with (3.28). They look like very different expressions, but mathematically they are equivalent. It is possible to rewrite one set into the other using the matrix inversion lemma (Theorem A.7). When there are more measurements than weights (in practice pretty much always) it is computationally more efficient to use (3.30) though, because the resulting matrix that needs to be inverted will be smaller.

ADDING AN OFFSET TO THE FUNCTION

There is one minor limitation to our method. The linear function $f(\mathbf{x}) = \underline{\mathbf{w}}^T \mathbf{x}$ is a function that always passes through the origin. To get any possible linear function, we can add an *offset* or *bias* $\underline{b} \sim \mathcal{N}(m_b, k_b^2)$, resulting in the function $f(\mathbf{x}) = \underline{\mathbf{w}}^T \mathbf{x} + \underline{b}$. Assuming that this offset is independent from the weights, we can now find that

$$m_{\text{lin}}(\mathbf{x}) = \mathbb{E}[\underline{\mathbf{w}}^T \mathbf{x} + \underline{b}] = m_b, \quad (3.31)$$

$$\begin{aligned} k_{\text{lin}}(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})])(f(\mathbf{x}') - \mathbb{E}[f(\mathbf{x}')])] \\ &= \mathbb{E}[(\underline{\mathbf{w}}^T \mathbf{x} + \underline{b} - m_b)(\underline{\mathbf{w}}^T \mathbf{x}' + \underline{b} - m_b)] \\ &= \mathbb{E}[\mathbf{x}^T \underline{\mathbf{w}} \underline{\mathbf{w}}^T \mathbf{x}' + (\underline{b} - m_b)^2] \\ &= \mathbf{x}^T K_w \mathbf{x}' + k_b^2. \end{aligned} \quad (3.32)$$

Although in practice it is often easier to consider the offset \underline{b} as an extra weight. We then replace the weight vector $\underline{\mathbf{w}}$ by $[\underline{\mathbf{w}}^T \underline{b}]^T$ and every single input vector \mathbf{x} by $[\mathbf{x}^T 1]^T$. Similarly, k_b^2 will be appended to the diagonal of K_w . When we apply this, we also get an estimate for the offset of the function we are approximating. For an example, see Figure 3.5 (right).

FUNCTIONS THAT ARE LINEAR IN FEATURES

3

Another way to extend this scheme, is by applying a so-called *feature function* $\phi(\mathbf{x})$. This is a known function that turns the d_x -dimensional input vector \mathbf{x} into a new d_ϕ -dimensional *feature vector* $\phi(\mathbf{x})$. The question now is ‘How can we approximate a function $f(\mathbf{x}) = \underline{\mathbf{w}}^T \phi(\mathbf{x}) + \underline{b}$ that is linear in the features?’

This actually goes in exactly the same way. The mean and covariance function that we now wind up with equal

$$m_{\text{lin}}(\mathbf{x}) = m_b, \quad (3.33)$$

$$k_{\text{lin}}(\mathbf{x}, \mathbf{x}') = \phi^T(\mathbf{x}) K_w \phi(\mathbf{x}') + k_b^2. \quad (3.34)$$

It does not matter here whether the feature function ϕ is linear or nonlinear. It only matters that $f(\mathbf{x})$ is linear in the features.

If we want to apply this in our regression equations, we cannot use X_m anymore though. Instead, just like we have replaced \mathbf{x} by $\phi(\mathbf{x})$ in the covariance function, we should also replace X_m by $\Phi_m \equiv \phi(X_m)$ and X_* by $\Phi_* \equiv \phi(X_*)$. If we do, then all equations still work as normal and we can apply GP regression with feature functions. An example of this at work is shown in Figure 3.6.

3.2.3. TRADING OFF COVARIANCE FUNCTIONS

By now we know of various different covariance functions. Which one should we pick for our data? We can choose that manually. For instance, if we know our data is periodic, we can take a periodic covariance function. But it can also be calculated which covariance function is most likely to represent the data.

To do this, we should see our choice of covariance function as just another (discrete) hyperparameter. So identically to (3.2), we now have

$$p(k, \boldsymbol{\theta} | \hat{\mathbf{f}}_m, X_m) = \frac{p(\hat{\mathbf{f}}_m | k, \boldsymbol{\theta}, X_m) p(k, \boldsymbol{\theta} | X_m)}{p(\hat{\mathbf{f}}_m | X_m)}, \quad (3.35)$$

where k denotes the covariance function that we will use. The likelihood $p(\hat{\mathbf{f}}_m | k, \boldsymbol{\theta}, X_m)$ is known and the marginal likelihood $p(\hat{\mathbf{f}}_m | X_m)$ is a constant with respect to k . The prior $p(k, \boldsymbol{\theta} | X_m)$ can still be split up further though. We can write this as

$$p(k, \boldsymbol{\theta} | X_m) = p(k, \boldsymbol{\theta}) = p(\boldsymbol{\theta} | k) p(k), \quad (3.36)$$

where $p(k)$ is the prior distribution of covariance functions. It is where we can indicate in advance which covariance function is more likely. Given this covariance function, $p(\boldsymbol{\theta} | k)$ is the prior distribution of the hyperparameters specific to that covariance function.

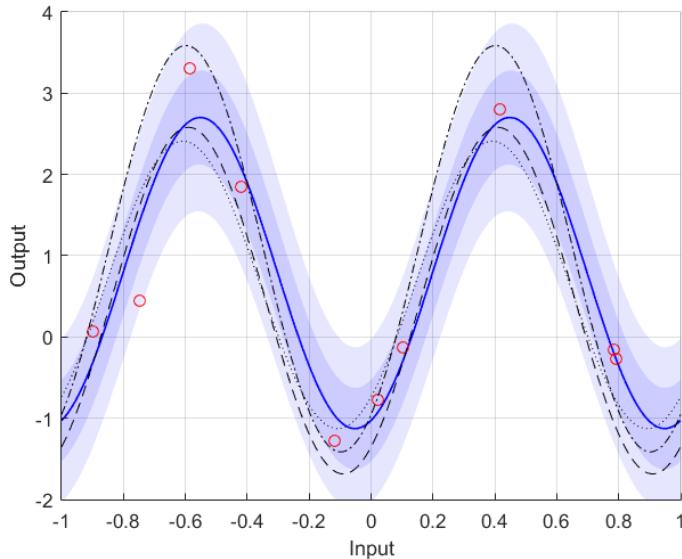


Figure 3.6: Feature GP regression applied to a sinusoid with a known period $p = 1$ but with an unknown amplitude, phase and mean. Measurement data was taken from $f(x) = 1 + 2 \sin\left(2\pi\left(x - \frac{1}{4}\right)\right)$ with noise of standard deviation $\hat{\sigma}_{f_m} = 1$. As feature functions, we used $\phi_1(x) = \sin(2\pi x)$, $\phi_2(x) = \cos(2\pi x)$ and $\phi_3(x) = 1$. By adding up linear combinations of these feature functions, any sinusoid with the same period can be obtained. (To see how this works, take a look at Theorem A.40.) After the regression, the weight posteriors were $w_1 \sim \mathcal{N}(0.61, 0.45^2)$ (for our function, ideally $\sqrt{2}$), $w_2 \sim \mathcal{N}(-1.81, 0.44^2)$ (ideally $-\sqrt{2}$) and $w_3 \sim \mathcal{N}(0.79, 0.32^2)$ (ideally 1). This corresponds to an amplitude of $A = 1.91$ (ideally 2) and a phase of -0.40π (ideally $-\frac{1}{2}\pi$).

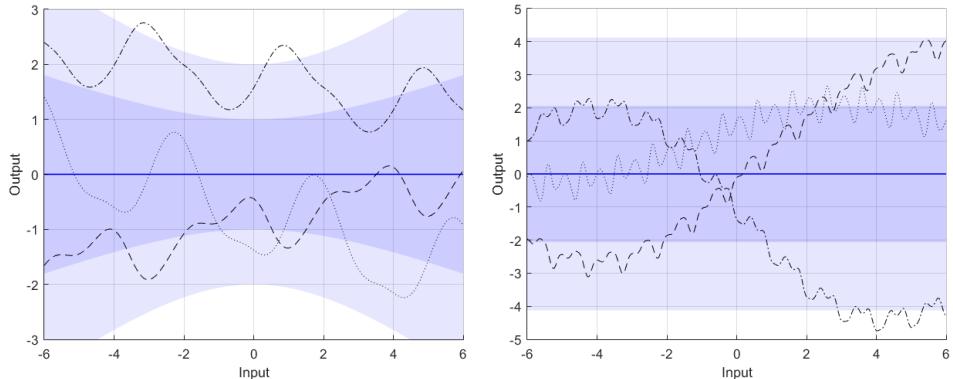


Figure 3.7: Prior distribution and samples of a Gaussian processes resulting from two added covariance functions. The left figure is a periodic covariance function ($\lambda_x = \lambda_f = 1$, $p = 4$) added to a linear covariance function ($\Sigma_w = \frac{1}{4^2}$). The right figure is a quickly varying periodic covariance function ($\lambda_x = \lambda_f = \frac{1}{2}$, $p = 1$) added to a slowly varying SE covariance function ($\lambda_x = 5$, $\lambda_f = 2$).

How do we proceed? Ideally, when making predictions, we take into account all possible covariance functions, just like in (3.7) we took into account all possible hyperparameters. So if we sum over all possible covariance functions k , we get

$$p(\underline{f}_* | \hat{\mathbf{f}}_m, X_m) = \sum_k \int_{\Theta} p(\underline{f}_* | k, \boldsymbol{\theta}, \hat{\mathbf{f}}_m, X_m) p(k, \boldsymbol{\theta} | \hat{\mathbf{f}}_m, X_m) d\boldsymbol{\theta}. \quad (3.37)$$

However, just like (3.7), this integral cannot be solved analytically. So we either have to resort to numerical methods again, or instead go for the maximum likelihood approach. That is, we find the combination of k and $\boldsymbol{\theta}$ that maximizes $p(k, \boldsymbol{\theta} | \hat{\mathbf{f}}_m, X_m)$ and only use those. In practice, because of the difficulty of setting up the numerical methods in a computationally efficient way, this last approach is the preferred method. Although, just like we noted at the end of Section 3.1.3, the maximum likelihood method has a small risk of overfitting.

3.2.4. COMBINING COVARIANCE FUNCTIONS

Suppose that we have a function $f(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$, where $f_1(\mathbf{x})$ is a linear function and $f_2(\mathbf{x})$ is a periodic function. We know which covariance functions to use for both $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$. So which covariance function should we take for $f(\mathbf{x})$?

The answer here is that we can also add up the covariance functions. To be precise, if $f_1(\mathbf{x})$ is a GP with mean $m_1(\mathbf{x})$ and covariance function $k_1(\mathbf{x}, \mathbf{x}')$ and similarly for $f_2(\mathbf{x})$, and if the two GPs are independent, then $f(\mathbf{x})$ is a GP with mean and covariance function

$$m(\mathbf{x}) = m_1(\mathbf{x}) + m_2(\mathbf{x}), \quad (3.38)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}'). \quad (3.39)$$

A few applications of this method are shown in Figure 3.7, where you can see the resulting GPs.

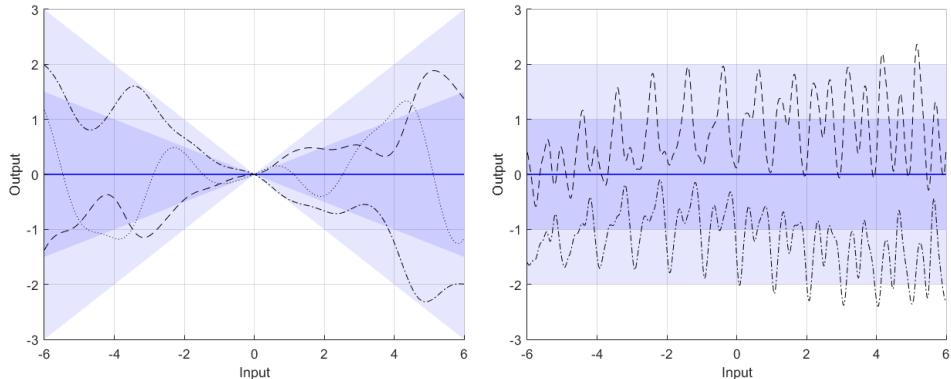


Figure 3.8: Prior distribution and samples of a Gaussian processes resulting from two multiplied covariance functions. The left figure is a periodic covariance function ($\lambda_x = \lambda_f = 1, p = 4$) multiplied by a linear covariance function ($\Sigma_w = \frac{1}{4^2}$). This results in a periodic function with linearly varying amplitude. The right figure is a quickly varying periodic covariance function ($\lambda_x = \lambda_f = \frac{1}{2}, p = 1$) multiplied by a slowly varying SE covariance function ($\lambda_x = 5, \lambda_f = 2$). This results in a periodic function where the exact variations during a period slowly change.

3

Can we apply the same trick for a function $f(\mathbf{x}) = f_1(\mathbf{x})f_2(\mathbf{x})$? Sadly, the answer here is no. The product of two Gaussian processes is generally not a Gaussian process. We can, however, approximate it as a Gaussian process. That is, pretend the outcome is a Gaussian process that just happens to have the same mean and covariance for each point. (This idea is called *moment matching*. For more details on it, see Section 5.1.2.) When we do, all the while assuming the Gaussian processes $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are independent, we find that the mean equals

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] = \mathbb{E}[f_1(\mathbf{x})f_2(\mathbf{x})] = \mathbb{E}[f_1(\mathbf{x})]\mathbb{E}[f_2(\mathbf{x})] = m_1(\mathbf{x})m_2(\mathbf{x}). \quad (3.40)$$

The covariance can be found with the same method, albeit with more bookkeeping. If we quickly walk through the derivation, occasionally skipping a step or two, we find that

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f_1(\mathbf{x})f_2(\mathbf{x}) - m_1(\mathbf{x})m_2(\mathbf{x})) (f_1(\mathbf{x}')f_2(\mathbf{x}') - m_1(\mathbf{x}')m_2(\mathbf{x}'))] \\ &= \mathbb{E}[f_1(\mathbf{x})f_1(\mathbf{x}')]\mathbb{E}[f_2(\mathbf{x})f_2(\mathbf{x}')] - m_1(\mathbf{x})m_1(\mathbf{x}')m_2(\mathbf{x})m_2(\mathbf{x}') \\ &= (k_1(\mathbf{x}, \mathbf{x}') + m_1(\mathbf{x})m_1(\mathbf{x}'))(k_2(\mathbf{x}, \mathbf{x}') + m_2(\mathbf{x})m_2(\mathbf{x}')) - m_1(\mathbf{x})m_1(\mathbf{x}')m_2(\mathbf{x})m_2(\mathbf{x}') \\ &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') + k_1(\mathbf{x}, \mathbf{x}')m_2(\mathbf{x})m_2(\mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')m_1(\mathbf{x})m_1(\mathbf{x}'). \end{aligned} \quad (3.41)$$

Often, when $m_1(\mathbf{x}) = m_2(\mathbf{x}) = 0$, we can hence just use $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$. An example of an application of this, for different combinations of covariance functions, is shown in Figure 3.8.

The nice thing is that, by combining various ‘basic’ covariance functions like the linear covariance function, the periodic covariance function and such, we can create all sorts of more complicated covariance functions. So if we know that the function $f(\mathbf{x})$ we are approximating has some kind of structure, we can take it into account.

The next question is: can we also automatically detect when our function $f(\mathbf{x})$ is a sum or product of these basic covariance functions? Interestingly enough this is possible, but it is not very easy. The key is to efficiently try combinations of basic covariance functions and then see if the new combined covariance function happens to be more likely than its predecessor. The exact search strategy is a bit too complicated to discuss here, but you can read more about it in the work of Duvenaud et al. (2013).

3.3. APPLYING GP REGRESSION TO LINEAR RELATIONS

3

So far we have assumed that, through our measurements, we directly measure certain function values $f(x_m)$. But what if, instead, we would measure linear relations between function values? For instance, what if we measure that $\frac{2}{3}f(x_{m_1}) + \frac{1}{3}f(x_{m_2})$ equals some value c ? That is the question we will answer in this section.

We start by looking at how we can take into account measurements of linear relations of function values (Section 3.3.1) and then look at how we can make predictions using these measurements (Section 3.3.2). We also check how we can still apply hyperparameter tuning (Section 3.3.3) and finally look at a possible practical use of the new technique (Section 3.3.4).

3.3.1. MEASURING LINEAR RELATIONS OF FUNCTION VALUES

Suppose that we have a set of measurement points X_m . The corresponding function values are denoted by $\underline{\mathbf{f}}_m = f(X_m)$, as usual, and have $\underline{\mathbf{f}}_m \sim \mathcal{N}(\mathbf{m}_m, K_{mm})$ as prior distribution.

Next, suppose that measurements have told us that $M\underline{\mathbf{f}}_m = \hat{\mathbf{c}}$ for some $n_c \times n_m$ matrix M . Here, the measurement $\hat{\mathbf{c}}$ can also be distorted by measurement noise, turning it into a random variable $\underline{\hat{\mathbf{c}}} \sim \mathcal{N}(\hat{\mathbf{\mu}}_c, \hat{\Sigma}_c)$. The question now is ‘What is the posterior distribution of $\underline{\mathbf{f}}_m$?’

The idea behind solving this problem is explained by Theorem B.24, while the actual solution is given by Theorem B.25. The result is

$$\begin{aligned}\underline{\mathbf{f}}_m &\sim \mathcal{N}(\mathbf{\mu}_m, \Sigma_m), \\ \Sigma_m &= K_{mm} - K_{mm}M^T(MK_{mm}M^T + \hat{\Sigma}_c)^{-1}MK_{mm}, \\ \mathbf{\mu}_m &= \mathbf{m}_m + K_{mm}M^T(MK_{mm}M^T + \hat{\Sigma}_c)^{-1}(\hat{\mathbf{\mu}}_c - M\mathbf{m}_m).\end{aligned}\tag{3.42}$$

This expansion of GP regression we are now examining is called *constrained Gaussian process regression*. The reason is that we do not measure $\underline{\mathbf{f}}_m$ fully, but our measurements only constrain it to be in a certain subspace of the full n_m -dimensional vector space. In fact, every row of M is called a *constraint*. As such, we have n_c constraints.

3.3.2. APPLYING CONSTRAINED GAUSSIAN PROCESS REGRESSION

The next question we should ask ourselves is ‘How can we make predictions?’ Mathematically, this comes down to joining $\underline{\mathbf{f}}_m$ together with the trial function values $\underline{\mathbf{f}}_*$ into a joint vector $\underline{\mathbf{f}}$. When we do this, we should also replace M by $[M \ 0]$. The actual mathematics are explained in Theorem B.26, but the outcome will be the *constrained GP*

regression equation

$$\begin{aligned} \begin{bmatrix} \underline{\mathbf{f}}_m \\ \underline{\mathbf{f}}_* \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \underline{\mathbf{f}}_m \\ \underline{\mathbf{f}}_* \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \Sigma_{mm} & \Sigma_{m*} \\ \Sigma_{*m} & \Sigma_{**} \end{bmatrix}\right), \quad (3.43) \\ \begin{bmatrix} \Sigma_{mm} & \Sigma_{m*} \\ \Sigma_{*m} & \Sigma_{**} \end{bmatrix} &= \begin{bmatrix} K_{mm} & K_{m*} \\ K_{*m} & K_{**} \end{bmatrix} - \begin{bmatrix} K_{mm} \\ K_{*m} \end{bmatrix} M^T (MK_{mm}M^T + \hat{\Sigma}_c)^{-1} M \begin{bmatrix} K_{mm} & K_{m*} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_* \end{bmatrix} &= \begin{bmatrix} \mathbf{m}_m \\ \mathbf{m}_* \end{bmatrix} + \begin{bmatrix} K_{mm} \\ K_{*m} \end{bmatrix} M^T (MK_{mm}M^T + \hat{\Sigma}_c)^{-1} (\hat{\boldsymbol{\mu}}_c - M\mathbf{m}_m). \end{aligned}$$

It is interesting to note that, when $M = I$ and $\hat{\boldsymbol{\mathcal{c}}} = \hat{\underline{\mathbf{f}}}_m$, then the above reduces (after a little bit of rewriting) back to the default GP regression equation (2.30). So this is actually a generalization of GP regression.

It is worthwhile to think about the runtime of this algorithm. We will look more into the runtime of algorithms in Section 4.1, but for now it is enough to know that the bottleneck is the size of the matrix $(MK_{mm}M^T + \hat{\Sigma}_c)$ we invert. Because M is an $n_c \times n_m$ matrix, the resulting matrix has size $n_c \times n_c$. Since often $n_c < n_m$, this is actually beneficial. The runtime does *not* significantly depend on how many measurement points are in X_m , but on how many constraints we have in M . The more constraints there are, the more data we add to our Gaussian process, but also the slower our regression algorithm becomes. Although if we do get $n_c > n_m$, then it is also possible to rewrite the regression equations through the matrix inversion lemma (Theorem A.7) to turn n_m into the bottleneck.

3.3.3. HYPERPARAMETER TUNING FOR CONSTRAINED GP REGRESSION

In this constrained GP regression problem, is it still possible to tune hyperparameter? The answer here is yes, although we need to adjust our equations a bit.

The key here is to realize that our measurements have told us that $M\underline{\mathbf{f}}_m = \hat{\boldsymbol{\mathcal{c}}}$, so we need to determine the likelihood that this happened. To do so, we define the adjusted distribution

$$\underline{\mathbf{f}}'_m = M\underline{\mathbf{f}}_m \sim \mathcal{N}(Mm(X_m), Mk(X_m, X_m)M^T). \quad (3.44)$$

When tuning the hyperparameters, we need to use the likelihood that $\underline{\mathbf{f}}'_m$ equals $\hat{\boldsymbol{\mathcal{c}}} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_c, \hat{\Sigma}_c)$. Identically to (3.9), we then wind up with

$$\begin{aligned} \log(p) &= -\frac{n_m}{2} \log(2\pi) - \frac{1}{2} \log |MK_{mm}M^T + \hat{\Sigma}_c| \\ &\quad - \frac{1}{2} (\hat{\boldsymbol{\mu}}_c - M\mathbf{m}_m)^T (MK_{mm}M^T + \hat{\Sigma}_c)^{-1} (\hat{\boldsymbol{\mu}}_c - M\mathbf{m}_m) + \log(c). \end{aligned} \quad (3.45)$$

If we now redefine $P = MK_{mm}M^T + \hat{\Sigma}_c$ and $\boldsymbol{\alpha} = P^{-1}(\hat{\boldsymbol{\mu}}_c - M\mathbf{m}_m)$, then the tuning of all the hyperparameters can be done nearly identically to what we did in Section 3.1.4. That is, we still have (3.12), which told us that

$$\frac{\partial \log(p)}{\partial \theta_i} = \frac{1}{2} \text{tr} \left((\boldsymbol{\alpha} \boldsymbol{\alpha}^T - P^{-1}) \frac{\partial P}{\partial \theta_i} \right). \quad (3.46)$$

We can use this to tune the hyperparameters of any covariance function. We will again focus on the SE covariance function though. The derivative of P with respect to $\hat{\sigma}_c^2$, which

is the noise variance of a single element of $\hat{\mathbf{c}}$, now still equals (see (3.13))

$$\frac{\partial P}{\partial \hat{\sigma}_c} = I. \quad (3.47)$$

The derivative of P with respect to λ_f^2 is slightly different from (3.14). It now equals

$$\frac{\partial P}{\partial \lambda_f^2} = \frac{MK_{mm}M^T}{\lambda_f^2}. \quad (3.48)$$

3

To find the derivative with respect to $\lambda_{x_k}^2$, we need to take two steps. First we should find the derivative $\partial K_{mm}/\partial \lambda_{x_k}^2$. We do this element-wise. Identically to (3.17), we get

$$\frac{\partial k(\mathbf{x}_{m_i}, \mathbf{x}_{m_j})}{\partial \lambda_{x_k}^2} = \frac{1}{2} k(\mathbf{x}_{m_i}, \mathbf{x}_{m_j}) \left(\frac{x_{m_i}^k - x_{m_j}^k}{\lambda_{x_k}^2} \right)^2. \quad (3.49)$$

It now directly follows that

$$\frac{\partial P}{\partial \lambda_{x_k}^2} = M \frac{\partial K_{mm}}{\lambda_{x_k}^2} M^T. \quad (3.50)$$

Finally, similarly to (3.19), we get

$$\bar{m} = \frac{\mathbf{1}^T M^T P^{-1} \hat{\mu}_c}{\mathbf{1}^T M^T P^{-1} M \mathbf{1}}. \quad (3.51)$$

This allows us to still tune the hyperparameters when using constrained GP regression.

3.3.4. A PRACTICAL USE OF CONSTRAINED GP REGRESSION

There are many practical uses for constrained GP regression. We will outline one of them here.

Consider we have a system subject to

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \quad (3.52)$$

with $\mathbf{x}(t)$ the state and $\mathbf{u}(t)$ the input. To control this system, we can use a *parameterized control law* $\mathbf{u}(t) = C(\mathbf{x}(t), \boldsymbol{\theta}_c)$, where $\boldsymbol{\theta}_c$ is a set of *controller parameters* that we need to choose. The question now is: which set of controller parameters is optimal?

To answer that question, we first need a quality criterion. We assume that the *instantaneous reward* of the system, for being in a state \mathbf{x} and applying an input \mathbf{u} , is given by the *reward function* $r(\mathbf{x}, \mathbf{u})$. We now want to maximize the sum of the rewards. In particular, we want to maximize the *value*

$$V = \frac{\int_0^\infty \gamma^t r(\mathbf{x}(t), \mathbf{u}(t)) dt}{\int_0^\infty \gamma^t dt}. \quad (3.53)$$

The parameter γ makes sure rewards in the future are weighed differently than rewards right now. In many reinforcement learning applications (see for instance Sutton and

Barto (1998), Bertsekas and Tsitsiklis (1996)) we have $0 < \gamma < 1$, and γ is then called the *discount factor*. For other applications, having $\gamma > 1$ could also be useful, although then it is often wise to cut the denominator from (3.53). (More on this can be found in Appendix C.2.4.) We now assume that $\gamma < 1$ and have the denominator in the above expression present for normalization purposes. It makes sure that the scale of V does not vary when we change γ . As a result, V can now be seen as a ‘weighted mean reward’ over the infinitely long time period.

Let’s take a closer look at (3.53). What does V depend on? Naturally, it depends on the initial state \mathbf{x}_0 , but it also depends on the control law that we use. Or to be precise, on the controller parameters $\boldsymbol{\theta}_c$. This means that the *value function* $V(\mathbf{x}_0, \boldsymbol{\theta}_c)$ satisfies

$$V(\mathbf{x}_0, \boldsymbol{\theta}_c) = -\log(\gamma) \int_0^\infty \gamma^t r(\mathbf{x}(t), C(\mathbf{x}(t), \boldsymbol{\theta}_c)) dt. \quad (3.54)$$

Note that we have also solved the integral in the denominator, causing the $\log(\gamma)$ factor. The next question is ‘How do we find or approximate this value function?’

If we do not know the system, we will have to do experiments. One thing that we can do is to put the system in a certain initial state \mathbf{x}_0 , apply controller parameters $\boldsymbol{\theta}_c$ and let the simulation run until $t \rightarrow \infty$, or at least until either γ^t or $\mathbf{x}(t)$ becomes negligibly small. This gives us a measurement of $V(\mathbf{x}_0, \boldsymbol{\theta}_c)$. If we do enough of such experiments, we can approximate the value function.

In reality running such long simulations is undesirable, especially when γ is close to 1 or even larger than 1. Instead, we could also run a shorter simulation, up to time T , and then note that

$$\begin{aligned} V(\mathbf{x}_0, \boldsymbol{\theta}_c) &= -\log(\gamma) \int_0^T \gamma^t r(\mathbf{x}(t), \mathbf{u}(t)) dt - \log(\gamma) \int_T^\infty \gamma^t r(\mathbf{x}(t), \mathbf{u}(t)) dt \\ &= -\log(\gamma) \int_0^T \gamma^t r(\mathbf{x}(t), \mathbf{u}(t)) dt - \log(\gamma) \gamma^T \int_0^\infty \gamma^t r(\mathbf{x}(t+T), \mathbf{u}(t+T)) dt \\ &= -\log(\gamma) \int_0^T \gamma^t r(\mathbf{x}(t), \mathbf{u}(t)) dt + \gamma^T V(\mathbf{x}(T), \boldsymbol{\theta}). \end{aligned} \quad (3.55)$$

Note that γ^T is not the transpose of the scalar γ , but actually denotes γ to the power of T . The above now gives us a recursive relation for the value function, expressing one value into another value. But what can we do with it?

During our experiment, we could of course keep track of the value of the integral. In fact, it is more convenient, for reasons we will see later, to instead keep track of the *weighted mean reward* \bar{r} during the time interval, defined as

$$\bar{r} \equiv \frac{\int_0^T \gamma^t r(\mathbf{x}(t), \mathbf{u}(t)) dt}{\int_0^T \gamma^t dt} = \frac{-\log(\gamma)}{1 - \gamma^T} \int_0^T \gamma^t r(\mathbf{x}(t), \mathbf{u}(t)) dt. \quad (3.56)$$

Using this quantity, we now have

$$V(\mathbf{x}_0, \boldsymbol{\theta}_c) = (1 - \gamma^T) \bar{r} + \gamma^T V(\mathbf{x}(T), \boldsymbol{\theta}_c). \quad (3.57)$$

You may have realized that this expression is very similar to the Bellman equation applied in reinforcement learning. We will use it when applying GP regression.

The idea is that we do n_m measurements. During the first measurement we start in an initial state \mathbf{x}_1^i , apply controller parameters $\boldsymbol{\theta}_{c_1}$ and end in a final state \mathbf{x}_1^f . During the experiment time T_1 , we have obtained a weighted mean reward \bar{r}_1 , and as a result

$$V(\mathbf{x}_1^i, \boldsymbol{\theta}_{c_1}) = (1 - \gamma^{T_1})\bar{r}_1 + \gamma^{T_1} V(\mathbf{x}_1^f, \boldsymbol{\theta}_{c_1}). \quad (3.58)$$

For each subsequent measurement we get a similar relationship. The result is that we get a set of equations

3

$$\begin{bmatrix} 1 & -\gamma^{T_1} & 0 & 0 & \cdots \\ 0 & 0 & 1 & -\gamma^{T_2} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} V(\mathbf{x}_1^i, \boldsymbol{\theta}_1) \\ V(\mathbf{x}_1^f, \boldsymbol{\theta}_1) \\ V(\mathbf{x}_2^i, \boldsymbol{\theta}_2) \\ V(\mathbf{x}_2^f, \boldsymbol{\theta}_2) \\ \vdots \end{bmatrix} = \begin{bmatrix} (1 - \gamma^{T_1})\bar{r}_1 \\ (1 - \gamma^{T_2})\bar{r}_2 \\ \vdots \end{bmatrix}. \quad (3.59)$$

This is something that we can put into our GP framework. In fact, we can write the above as

$$M\mathbf{f}_m = \hat{\mathbf{c}}, \quad (3.60)$$

with M and $\hat{\mathbf{c}}$ defined accordingly. At the same time, the first measurement input \mathbf{x}_{m_1} consists of \mathbf{x}_1^i and $\boldsymbol{\theta}_{c_1}$, the second measurement input \mathbf{x}_{m_2} consists of \mathbf{x}_1^f and $\boldsymbol{\theta}_{c_1}$, the third measurement input \mathbf{x}_{m_3} consists of \mathbf{x}_2^i and $\boldsymbol{\theta}_{c_2}$, and so on.

With this set of measurements, and with the constrained GP regression equation (3.43), we can subsequently make predictions of the value function. Later on in Section 3.5 we will see an application of this, proving that the method actually works.

3.4. LINEARIZED MODELING OF THE PITCH-PLUNGE SYSTEM

It is time to apply the things we have learned in this chapter. In this first application section we will revisit our experiments from Section 2.6, identifying a pitch-plunge system. We start by linearizing this system (Section 3.4.1) and subsequently we apply the linear covariance function to identify this system (Section 3.4.2). Finally we look at what happens when we apply the same methods to the nonlinear system (Section 3.4.3).

3.4.1. THE LINEARIZED DISCRETE EQUATIONS OF MOTION

Let's revisit the equations of motion of the pitch-plunge system derived in Section 2.6.2. If these equations were linear, we could approximate them using a linear covariance function. But are they?

To figure out the details behind this, we rewrite (2.61) to state-space form. This gives us

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \ddot{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -M^{-1}(K(\mathbf{x}) + U^2 D) & -M^{-1}(C + UE) \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}U^2 F \end{bmatrix} \beta. \quad (3.61)$$

We can write this as $\dot{\tilde{\mathbf{x}}} = \tilde{A}(\tilde{\mathbf{x}})\tilde{\mathbf{x}} + \tilde{B}\beta$. In the special case that our nonlinear spring $k_\alpha(\alpha)$ is a linear spring, then $\tilde{A}(\tilde{\mathbf{x}})$ equals a constant \tilde{A} and our system will indeed be linear. In

this case we can also easily turn it into a discrete-time system through

$$\tilde{\mathbf{x}}_{k+1} = e^{\tilde{A}\Delta t} \tilde{\mathbf{x}}_k + \int_0^{\Delta t} e^{\tilde{A}(\Delta t-s)} \tilde{B} \beta(s) ds. \quad (3.62)$$

If we additionally assume that $\beta(t)$ equals some constant β_k during the duration of the (small) time step Δt , this can be reduced to

$$\tilde{\mathbf{x}}_{k+1} = e^{\tilde{A}\Delta t} \tilde{\mathbf{x}}_k + \left(e^{\tilde{A}\Delta t} - I \right) \tilde{A}^{-1} \tilde{B} \beta_k = A_d \tilde{\mathbf{x}}_k + B_d \beta_k. \quad (3.63)$$

Here we have defined the discrete-time matrices A_d and B_d as shown above. In the limit of $\Delta t \rightarrow 0$ we could even reduce $(e^{\tilde{A}\Delta t} - I)$ to $\tilde{A}\Delta t$, causing B_d to equal $\tilde{B}\Delta t$, but usually our time step is not *that* small.

Still, the above tells us something very interesting. The expression for \mathbf{x}_{k+1} equals the top two rows of this four-dimensional matrix equation, which means that \mathbf{x}_{k+1} linearly depends on \mathbf{x}_k , $\dot{\mathbf{x}}_k$ and β_k . At least, as long as all matrices remain constant. So if we consider the linear system (keeping k_a constant) with a constant wind speed U , then the discretized state transition function (2.62) is a linear function! And that is something we can approximate using a linear covariance function.

3.4.2. APPLYING THE LINEAR COVARIANCE FUNCTION

The idea is that we will approximate A_d and B_d from (3.63) with an identical experiment as was done in Section 2.6.4. That is, we randomly choose \mathbf{x}_k , $\dot{\mathbf{x}}_k$ and β_k from a prespecified interval. Together, these parameters constitute the input for our regression algorithm. We then run a simulation for $\Delta t = 0.1$ s, measure the new values of \mathbf{x}_{k+1} and $\dot{\mathbf{x}}_{k+1}$, and use these as measured outputs for our regression algorithm.

The function we are approximating now has $d_x = 5$ inputs and $d_y = 4$ outputs. This means we need to apply the GP regression algorithm d_y separate times, once for each output. For each of these times, we need to set up the covariance function and properly choose the scaling parameters.

To do so, we first define length scales. Here we choose $\lambda_h = 0.005$ m, $\lambda_\alpha = 0.06$ rad, $\lambda_{\dot{h}} = 0.05$ m/s, $\lambda_{\dot{\alpha}} = 1$ rad/s and $\lambda_\beta = 0.5$ rad. Next, we will apply GP regression to predict h_{k+1} . We know that h_{k+1} can be written as

$$h_{k+1} = w_{h/h} h_k + w_{h/\alpha} \alpha_k + w_{h/\dot{h}} \dot{h}_k + w_{h/\dot{\alpha}} \dot{\alpha}_k + w_{h/\beta} \beta_k, \quad (3.64)$$

with the w -parameters being the weights within \mathbf{w} . Based on this, we could say that, a priori, the standard deviation of $w_{h/h}$ equals λ_h / λ_h , the standard deviation of $w_{h/\alpha}$ equals $\lambda_h / \lambda_\alpha$, and so on. As a result, we have as prior weight matrix, when predicting h ,

$$K_w^h = \bar{w}^2 \text{diag} \left(\frac{\lambda_h^2}{\lambda_h^2}, \frac{\lambda_h^2}{\lambda_\alpha^2}, \frac{\lambda_h^2}{\lambda_{\dot{h}}^2}, \frac{\lambda_h^2}{\lambda_{\dot{\alpha}}^2}, \frac{\lambda_h^2}{\lambda_\beta^2} \right), \quad (3.65)$$

where $\text{diag}(a, b, c, \dots)$ denotes the diagonal matrix with a, b, c, \dots along its diagonal. Also, \bar{w} is a scaling constant we can choose, depending on how much variation we expect. I often use $\bar{w} = 2$. If we now also appropriately pick a noise matrix $\hat{\Sigma}_{f_m}^h$, then we can

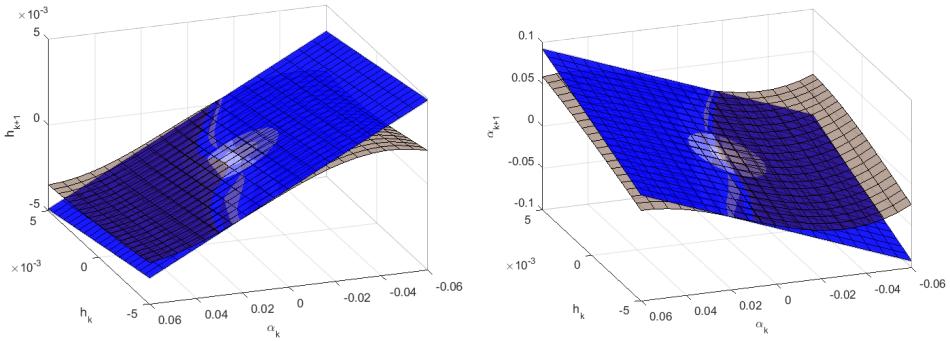


Figure 3.9: The prediction of the next state of the *linearized* pitch-plunge system, based on the current state. A linear covariance function was used. For the training data, all initial state and input parameters (h , α , \dot{h} , $\dot{\alpha}$ and β) were set to random values. For the tests resulting in the above plots, h , $\dot{\alpha}$ and β were set to zero. The time step used is $\Delta t = 0.1$ s and the number of measurements used is $n_m = 30$. The flat plane now is the prediction, where the 95% interval appears not to be shown because it is simply too small. The curved plane is the mean prediction of the *nonlinear* pitch-plunge system of Figure 2.14, present only as comparison. From the plots we can see that the linear system is indeed a linearized version of the nonlinear system, where the linearization was done around the origin. This confirms what we already knew.

apply (3.29) to estimate the top row of A_d and B_d . Doing the same for the other output parameters will also give us the other rows of A_d and B_d .

In its essence, this whole method comes down to applying least-squares regression. And when there is little measurement noise, this is of course a highly efficient and accurate algorithm. If we only use a few measurements, our predictions of A_d and B_d are already almost exactly equal to the analytical values of A_d and B_d . The resulting plots are shown in Figure 3.9.

3.4.3. SWITCHING TO THE NONLINEAR SYSTEM

We now know how to approximate a linear system. It gets more interesting when we apply the same methods to the nonlinear system. The algorithm now still tries to approximate the system as a linear system, which of course is not really possible anymore. The resulting outcome is shown in Figure 3.10.

When looking at Figure 3.10, we can get an idea. What if we approximate the difference between the measurements and the shown linear model by a Gaussian process with a squared exponential covariance function? Will that give a more accurate estimate?

This approach is a bit circuitous though. A more simple way to do nearly the exact same thing would be to use a combined covariance function: we add up the squared exponential covariance function to the linear covariance function we are already using. Then we apply GP regression using this covariance function. The outcome is now shown in Figure 3.11.

If you have a good memory, you may notice that this is almost exactly the same plot as Figure 2.15, where we only used the SE covariance function. So adding the linear covariance function here did not do much.

The reason for this is that the SE covariance function is already very well capable of approximating the function. In fact, the curved function that we are trying to approxi-

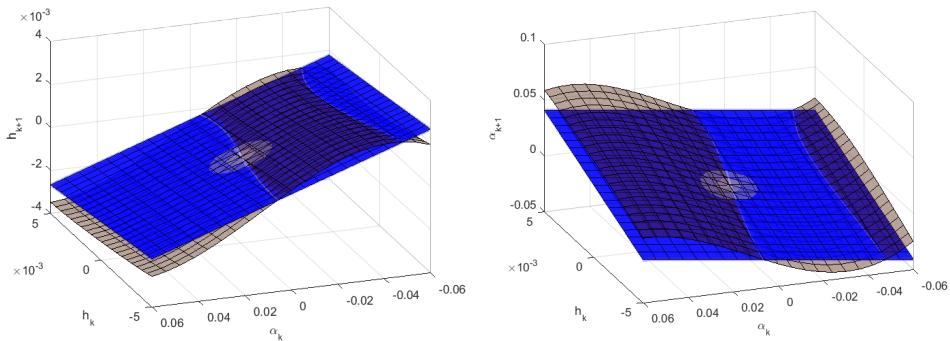


Figure 3.10: The prediction of the next state of the *nonlinear* pitch-plunge system, based on the current state. The set-up is exactly the same as that of Figure 3.9, except that now the nonlinear model is used. In this case the linear model resulting from the GP regression can be seen as an ‘average plane’ of the nonlinear model. In fact, because we use a linear covariance function, the GP regression algorithm ‘believes’ that the function it is approximating *must* be linear. As such, it explains any nonlinear discrepancy as noise, resulting in an overly certain estimate.

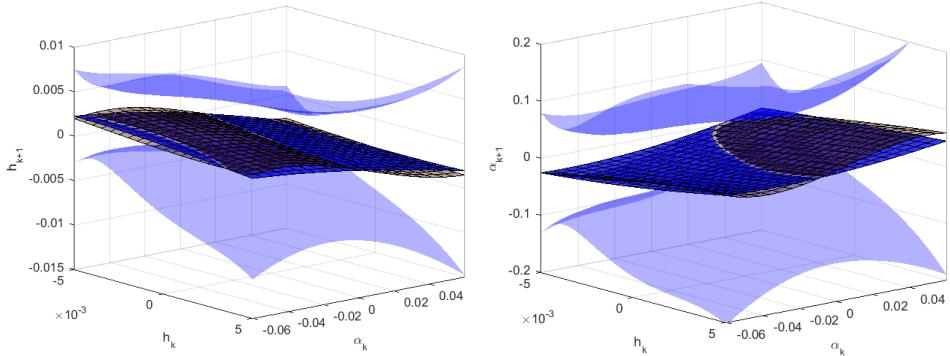


Figure 3.11: The prediction of the next state of the nonlinear pitch-plunge system, based on the current state. The set-up is exactly the same as that of Figure 3.10, except that now a linear plus squared exponential covariance function is used. The fact that this figure is nearly identical to Figure 2.15 shows that, for this particular problem, adding the linear covariance function is not really necessary.

mate looks like it could even have been generated by an SE covariance function. As such, adding the linear covariance function here is pointless. If we would have a function that is highly linear, with a few small deviations, then using a linear plus SE covariance function would be useful. But for our problem the SE covariance function suffices.

3.5. APPROXIMATING A QUADRATIC VALUE FUNCTION

The previous section identified the dynamics of the pitch-plunge system of Section 2.6. In this section we will look into adding a controller and tuning it by optimizing a quadratic value function.

3

We start by putting the pitch-plunge system into an LQG set-up (Section 3.5.1). Before adding a controller to this problem, we first approximate the value function of this problem, both without noise (Section 3.5.2) and with noise (Section 3.5.3). Afterwards, we add a controller and tune it by optimizing the value function (Section 3.5.4). Finally we look at a few extensions we could possibly still add to the scheme (Section 3.5.5).

3.5.1. THE LQG PROBLEM SET-UP

Let's consider the pitch-plunge system of Section 2.6. We want to add a controller to this system. To know which controllers work well and which ones do not, we need a criterion to optimize: a value function. The method that we will use is the one described in Section 3.3.4.

To be able to check our results, we want to be able to analytically calculate the outcome that we are supposed to get. As such we will start with a familiar set-up: we will use LQG control. The 'L' in LQG control stands for 'Linear'. That is, we will use the linearized system, as well as a control law that is linear in the state $\tilde{\mathbf{x}}$. So,

$$\beta(t) = C(\tilde{\mathbf{x}}(t), \boldsymbol{\theta}_c) = -\tilde{F}\tilde{\mathbf{x}} = -F_h h(t) - F_\alpha \alpha(t) - F_{\dot{h}} \dot{h}(t) - F_{\dot{\alpha}} \dot{\alpha}(t). \quad (3.66)$$

In this expression, $\boldsymbol{\theta}_c$ is the set of controller parameters that we want to tune. It hence consists of F_h , F_α , $F_{\dot{h}}$ and $F_{\dot{\alpha}}$. Together, these parameters make up the feedback matrix \tilde{F} . Also note that we write the full state as $\tilde{\mathbf{x}}(t)$, so as not to confuse it with the state $\mathbf{x}(t)$ from (2.61) that only contains h and α . By using the above control law, we can reduce the state-space form (3.61) of the system to

$$\dot{\tilde{\mathbf{x}}}(t) = (\tilde{A} - \tilde{B}\tilde{F})\tilde{\mathbf{x}}(t). \quad (3.67)$$

Another important part within the LQG control paradigm is the *quadratic cost function*, or inversely the *quadratic value function*. To obtain this, we will apply the reward function

$$\begin{aligned} r(\tilde{\mathbf{x}}(t), \beta(t)) &= -\tilde{\mathbf{x}}^T(t)Q\tilde{\mathbf{x}}(t) - \beta^T(t)R\beta(t) \\ &= -\tilde{\mathbf{x}}^T(t)(Q + \tilde{F}^T R \tilde{F})\tilde{\mathbf{x}}(t) \\ &= -\tilde{\mathbf{x}}^T(t)\tilde{Q}\tilde{\mathbf{x}}(t), \end{aligned} \quad (3.68)$$

where the matrix Q and the (in our case) scalar R are weights specifying what kind of behavior we want to penalize. Since we want to minimize the motion of the airfoil, irrespective of its position, we will penalize $\dot{h}(t)$ and $\dot{\alpha}(t)$ without penalizing $h(t)$ and $\alpha(t)$. Naturally, inputs are also penalized. In addition, we will use a value of $\gamma = \frac{1}{2}$.

Given these settings, what would the value function $V(\tilde{\mathbf{x}}(t), \boldsymbol{\theta}_c)$ defined in (3.54) look like, with respect to $\tilde{\mathbf{x}}(t)$ and $\boldsymbol{\theta}_c$?

3.5.2. APPROXIMATING THE VALUE FUNCTION FOR A SINGLE CONTROLLER

The nice part is that, for the LQG system, the value function (or equivalently the cost) can be calculated analytically. Though the value will not have a Gaussian distribution, we can find the mean using the theory from Appendix C.2 and the variance with Appendix C.4. When there is no noise, the value V is still deterministic and then (see Theorem C.9) it equals

$$V(\tilde{\mathbf{x}}_0, \boldsymbol{\theta}_c) = -\log(\gamma) \tilde{\mathbf{x}}_0^T \bar{X} \tilde{\mathbf{x}}_0. \quad (3.69)$$

The term \bar{X} is defined (see Appendix A.4 for the notation definitions) as the solution to the Lyapunov equation

$$\left(\tilde{A} + \frac{1}{2} \log(\gamma) I - \tilde{B} \tilde{F} \right)^T \bar{X} + \bar{X} \left(\tilde{A} + \frac{1}{2} \log(\gamma) I - \tilde{B} \tilde{F} \right) + (Q + \tilde{F}^T R \tilde{F}) = 0. \quad (3.70)$$

For now assume that the controller settings $\boldsymbol{\theta}_c$, or equivalently the feedback matrix \tilde{F} , is fixed. If we do not know the system matrices \tilde{A} and \tilde{B} , we can approximate the resulting value function $V(\tilde{\mathbf{x}}_0)$ using a Gaussian process. In that case the value V becomes linear in the elements of \bar{X} . To be precise, we can use Theorem A.5 to write it as

$$V(\tilde{\mathbf{x}}_0) = -\log(\gamma) \text{vec}(\bar{X})^T \text{vec}(\tilde{\mathbf{x}}_0 \tilde{\mathbf{x}}_0^T), \quad (3.71)$$

where $\text{vec}(\bar{X})$ is the vectorization (for the definition, see (A.10)) of the matrix \bar{X} . We can even take into account the knowledge that \bar{X} is symmetric, by lumping identical terms together.

We now define $\underline{\mathbf{w}} = -\log(\gamma) \text{vec}(\bar{X})$. After all, the vector $\underline{\mathbf{w}}$ can be seen as a vector of weights that we do not know but want to find. We assume that $\underline{\mathbf{w}} \sim \mathcal{N}(\mathbf{0}, K_w)$, where we choose the prior weight covariance K_w ourselves. The vector $\underline{\Phi}(\tilde{\mathbf{x}}_0) = \text{vec}(\tilde{\mathbf{x}}_0 \tilde{\mathbf{x}}_0^T)$ now contains our features, as discussed at the end of Section 3.2.2. We can merge all these feature vectors into a feature matrix Φ_m . In this case Theorem B.28 tells us that the posterior distribution of $\underline{\mathbf{w}}$ equals

$$\begin{aligned} \underline{\mathbf{w}} &\sim \mathcal{N}(\boldsymbol{\mu}_{\underline{\mathbf{w}}}, \Sigma_{\underline{\mathbf{w}}}), \\ \Sigma_{\underline{\mathbf{w}}} &= (\Phi_m M^T \hat{\Sigma}_c^{-1} M \Phi_m^T + K_w^{-1})^{-1}, \\ \boldsymbol{\mu}_{\underline{\mathbf{w}}} &= \Sigma_{\underline{\mathbf{w}}} \Phi_m M^T \hat{\Sigma}_c^{-1} \hat{\mathbf{c}}_m. \end{aligned} \quad (3.72)$$

Through this we can approximate $\text{vec}(\bar{X})$ and hence the value function $V(\mathbf{x}_0)$. The result, for manually chosen values K_w and $\hat{\Sigma}_c$, is shown in Figure 3.12.

From Figure 3.12 we can see that, for our simple system, it is apparently better to have a high h_0 and low α_0 than have both a high h_0 and high α_0 . This seems to make sense. When $h_0 > 0$, the airfoil is displaced downwards. This causes the spring k_h to push the airfoil back up. Similarly, when $\alpha_0 > 0$ the airfoil has a positive angle of attack, which means that the wind will push it upwards. Having both $h_0 > 0$ and $\alpha > 0$ at the same time will cause a large upward force, resulting in a fast motion of the airfoil. Since fast motions are penalized, this will result in a negative value.

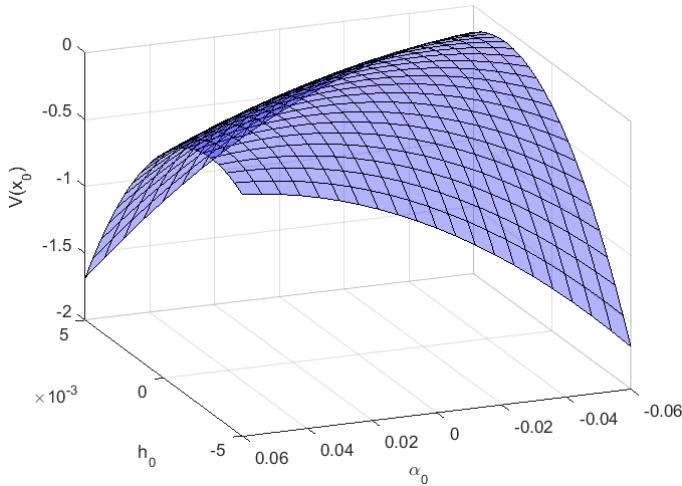


Figure 3.12: The value function of the linearized pitch-plunge system when no control is present. The plot was made for varying h_0 and α_0 , while \dot{h}_0 and $\dot{\alpha}_0$ were set to zero. To generate the data for this plot, $n_m = 50$ simulations of $T = 1$ s each were run. During these simulations, the initial parameters h_0 , α_0 , \dot{h}_0 and $\dot{\alpha}_0$ were set to random values. This initial state was recorded, as well as the final state and the weighted mean reward (3.56). This data was then used to approximate \tilde{X} . Because no noise was present, the approximation was highly accurate. It coincides with the true value function, barring minor numerical differences.

3.5.3. ADDING NOISE TO THE PROBLEM

The problem we just solved was a simple one: no noise at all was present. It is time to change that. We will assume that the incoming wind flow varies its direction, resulting in a change in the angle of attack $\underline{\alpha}_+$ that the system encounters.

Officially we should incorporate wind spectrums here. That is, we should use noise with approximately the same frequency distribution as regular wind has. However, to keep things simple (and linear) we define the extra angle of attack $\underline{\alpha}_+$, caused by the rotation of the wind, as Gaussian white noise with intensity σ_α^2 . This turns our system equation (3.67) into

$$\dot{\tilde{x}} = \tilde{A} \left(\tilde{x} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \underline{\alpha}_+ \right) - \tilde{B} \tilde{F} \tilde{x} = (\tilde{A} - \tilde{B} \tilde{F}) \tilde{x} + \tilde{A} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \underline{\alpha}_+ = (\tilde{A} - \tilde{B} \tilde{F}) \tilde{x} + \underline{w}, \quad (3.73)$$

where we have defined \underline{w} as Gaussian white noise with intensity

$$W = \tilde{A} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \sigma_\alpha^2 [0 \ 1 \ 0 \ 0] \tilde{A}^T. \quad (3.74)$$

Note that we use W here, because we already use V for the value function. Also note that such a quickly shifting wind field would be impossible in practice. It serves us well enough as a basic approximation of a turbulent wind field though.

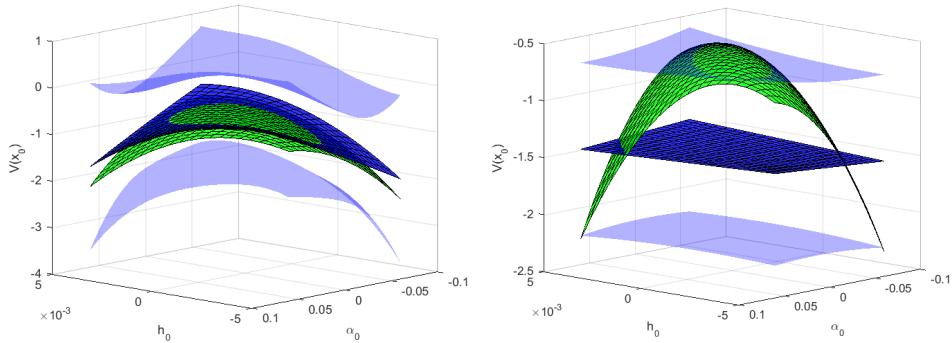


Figure 3.13: Approximation of the value function of the linear pitch-plunge system subject to process noise. Both the exact expected value $\mathbb{E}[V]$ and the GP approximation of it are shown. A number of $n_m = 50$ experiments was run, starting from a random initial state and lasting $T = 1$ s. The left figure tuned the output noise matrix $\hat{\Sigma}_c = \hat{\sigma}_c^2 I$. In addition, the right figure also tuned the initial weight covariance matrix K_w which, given the low number of measurements, was not an improvement. If a higher number of measurements was used (say, more than 200) then tuning K_w would have been possible.

For the current problem with noise, the value V has become a random variable. Using Theorem C.9 we can find that its mean equals

$$\mathbb{E}[V(\tilde{x}_0, \theta_c)] = -\log(\gamma) \text{tr} \left(\left(\tilde{x}_0 \tilde{x}_0^T - \frac{W}{\log(\gamma)} \right) \bar{X} \right) = \text{tr} \left((W - \log(\gamma) \tilde{x}_0 \tilde{x}_0^T) \bar{X} \right). \quad (3.75)$$

The value hence consists of two parts: one due to the initial state \tilde{x}_0 and one due to the noise w . So effectively we have added a constant term $\text{tr}(W\bar{X})$ to the expected value $\mathbb{E}[V]$. How can we take this into account in our regression algorithm?

The key here is to introduce a bias $b = \text{tr}(W\bar{X})$. Just like we discussed in Section 3.2.2, this means we add a 1 to $\text{vec}(\tilde{x}_0 \tilde{x}_0^T)$, while we add the unknown bias b to the unknown vector $w = \text{vec}(\bar{X})$. Although in reality the bias b is not Gaussian – given our reward function the value can only be negative – we can treat it as a Gaussian random variable. In this case, despite using an inaccurate prior distribution for b , our regression algorithm should work, although we might get a slightly smaller value of b then we would get with the correct prior distribution for b .

It is important to realize here that the weighted mean reward \bar{r} will be affected by noise at each experiment run. As a result, the measured vector \hat{e} will be noisy as well. This means that we need to choose the noise matrix $\hat{\Sigma}_c = \hat{\sigma}_c^2 I$. We can try tuning it by optimizing the log-likelihood (3.9) with respect to $\hat{\sigma}_c^2$. When we do, we get $\hat{\sigma}_c = 0.85$, resulting in the plot shown in Figure 3.13 (left).

From the plot, we see that the approximation is still reasonably good, despite all the noise. There is still quite some uncertainty, but this is to be expected, since we only used $n_m = 50$ measurements. Nevertheless, the found bias b is pretty close to what it should be, the error being roughly 10%. Of course everything will become more accurate when we use more measurements, showing that with Gaussian process regression we can take the effects of process noise w on the value function V into account.

Next, let's try to take this idea of hyperparameter tuning one step further. We have used it to tune $\hat{\sigma}_c$. We can also use it to tune K_w . Instead of tuning one parameter, we

would now be tuning more than ten parameters. Given that we have only fifty measurements, this is unlikely to be successful.

And indeed, Figure 3.13 (right) shows that the results are not very good. Many of the squared scale parameters in K_w decreased to very small values. Having a tiny K_w for the linear covariance function (or an overly large Λ_x for the SE covariance function) is a sign that the GP regression algorithm cannot find a structure in the data yet. As a result, the input X_m is pretty much ignored and all the output deviations are explained as noise. The result is a prediction that looks like a flat plane. So if you are tuning hyperparameters and get a flat plane, probably your data does not have enough information about its structure just yet.

Another interesting question we can ask ourselves is whether the tuned value of $\hat{\sigma}_c$ does make sense. That is, whether it corresponds with the actual noise (variance) that is present in our experiments.

To figure that out, we consider the vector \hat{c} before it is measured. In that case we write it as the random variable \underline{c} . We know from (3.59) that element i of the vector \underline{c} equals

$$\underline{c}_i = (1 - \gamma^T) \bar{t}_i = -\log(\gamma) \int_0^T \gamma^t (-\tilde{x}^T (Q + \tilde{F}^T R \tilde{F}) \tilde{x}) dt. \quad (3.76)$$

The integral (apart from the minus sign) equals the finite-time discounted cost \underline{J}_T , defined by (C.49). Its mean $\mathbb{E}[\underline{J}_T]$ can be found through Theorem C.8 and its variance $\mathbb{V}[\underline{J}_T]$ through Theorem C.23. Taking into account the factor $\log(\gamma)$ results in the mean and variance of \underline{c}_i . Hence, if we know the initial state \tilde{x}_0 , we can calculate $\mathbb{E}[\underline{c}_i(\tilde{x}_0)]$ and $\mathbb{V}[\underline{c}_i(\tilde{x}_0)]$.

The problem here is that both the mean and the variance depend on the initial state \tilde{x}_0 . We could let $\hat{\sigma}_c^2$ vary over the input space (a feature known as *heteroscedasticity*) or just assume it equals some average value. We will go for the latter option, and average over all initial states $\tilde{x}_0^1, \tilde{x}_0^2, \dots, \tilde{x}_0^{n_m}$ we encountered while setting up our measurement set X_m . When we do this, for the data that generated Figure 3.13, we find the mean cost variance as

$$\mathbb{V}[\underline{J}_T] = \frac{1}{n_m} \sum_{i=1}^{n_m} \mathbb{V}[\underline{J}_T(\tilde{x}_0^i)] = 0.11, \quad (3.77)$$

which results in a value of $\hat{\sigma}_c = \sqrt{0.11} = 0.34$. This is not exactly the value of $\hat{\sigma}_c$ found by the hyperparameter tuning, but it is the same order of magnitude. And indeed, using $\hat{\sigma}_c = 0.34$ would have resulted in a plot very similar to Figure 3.13 (left). This shows that using expert knowledge about the scale of system parameters is often a very effective way of choosing hyperparameters too.

3.5.4. VARYING THE CONTROLLER SETTINGS

So far we have not incorporated the controller settings θ_c . We will do so now.

The main issue to tackle is how we should adjust our covariance function. We know that the matrix $\tilde{X}(\theta_c)$ will vary with θ_c . Naturally, so will its elements $\underline{w}(\theta_c) = \text{vec}(\tilde{X}(\theta_c))$, and they will probably do so in a smooth way. As a result, we will use a squared exponential covariance function when approximating $\underline{w}(\theta_c)$. That is, we assume in advance

that

$$\mathbb{E} \left[(\underline{w}(\boldsymbol{\theta}_c) - \mathbb{E} [\underline{w}(\boldsymbol{\theta}_c)]) (\underline{w}(\boldsymbol{\theta}'_c) - \mathbb{E} [\underline{w}(\boldsymbol{\theta}'_c)])^T \right] = K_w \exp \left(-\frac{1}{2} (\boldsymbol{\theta}_c - \boldsymbol{\theta}'_c)^T \Lambda_{\theta}^{-1} (\boldsymbol{\theta}_c - \boldsymbol{\theta}'_c) \right). \quad (3.78)$$

Note that, for a single controller setting $\boldsymbol{\theta}_c = \boldsymbol{\theta}'_c$, the prior covariance of $\underline{w}(\boldsymbol{\theta}_c)$ equals K_w , just like it did previously. As $\boldsymbol{\theta}_c$ and $\boldsymbol{\theta}'_c$ drift further apart, the covariance reduces to zero, meaning that we assume there is almost no link between the value functions of two very differently controlled systems.

Keep in mind that the above is the covariance function of $\text{vec}(\bar{X})$. And since \underline{V} linearly depends on $\text{vec}(\bar{X})$, we get as covariance function of our value function $\underline{V}(\tilde{x}_0, \boldsymbol{\theta}_c)$

$$k(\tilde{x}_0, \boldsymbol{\theta}_c, \tilde{x}'_0, \boldsymbol{\theta}'_c) = \text{vec}(\tilde{x}_0 \tilde{x}'_0)^T K_w \exp \left(-\frac{1}{2} (\boldsymbol{\theta}_c - \boldsymbol{\theta}'_c)^T \Lambda_{\theta_c}^{-1} (\boldsymbol{\theta}_c - \boldsymbol{\theta}'_c) \right) \text{vec}(\tilde{x}'_0 \tilde{x}'_0^T). \quad (3.79)$$

With this covariance function, we can make predictions of $\underline{V}(\tilde{x}_0, \boldsymbol{\theta}_c)$ as a function of both the initial state \tilde{x}_0 and the controller settings $\boldsymbol{\theta}_c$.

We should note here that in our previous experiment the value function $\underline{V}(\tilde{x}_0)$ only had four input parameters. The function $\underline{V}(\tilde{x}_0, \boldsymbol{\theta}_c)$ we will approximate now has more. This means that we also need more measurements before we get accurate predictions. To keep the number of measurements somewhat limited, we set the noise W to zero, and still we will increase n_m to 500.

Because we are dealing with an LQG system, we can already calculate the optimal controller gains analytically. If we do, using Theorem C.15, we find the optimal feedback matrix $\tilde{F} = [-33.4 \quad 0.87 \quad -6.2 \quad -0.29]$. We should keep in mind here that these controller gains work well together, but separately they are not always effective. For instance, if we only set $C_h = -6.2$ or if we only set $C_{\alpha} = -0.29$, keeping the other gains zero, we actually wind up with an unstable system.

To make sure we only get stabilizing controllers during our experiments, we will tune only C_h and C_{α} , keeping both C_h and C_{α} at zero. Not very surprisingly, this causes the optimums of C_h and C_{α} to change. What is more surprising is that the optimal values of C_h and C_{α} now actually depend on the distribution of the initial state \tilde{x}_0 and on the noise W , while the optimal feedback matrix \tilde{F} did not depend on this in any way. Assuming that the initial state is distributed according to $\tilde{x}_0 \sim \mathcal{N}(\mathbf{0}, \Sigma_0)$ for an appropriately chosen Σ_0 , and keeping W at zero, we find the optimums $C_h = -10.6$ and $C_{\alpha} = 1.20$.

Of course we want to see if this also comes out of the GP regression algorithm. To check this, we want to plot the value function \underline{V} as a function of C_h and C_{α} . We can only plot this if we choose a value for the initial state \tilde{x}_0 . If we would set it to zero, we would (in the absence of noise) also get a zero value \underline{V} everywhere. So instead, we again use a distribution $\tilde{x}_0 \sim \mathcal{N}(\mathbf{0}, \Sigma_0)$ and integrate over this. For our prediction of the mean of \underline{V} , this turns out to be equivalent to replacing $\text{vec}(\tilde{x}_0 \tilde{x}_0)$ by $\text{vec}(\Sigma_0)$. When we do, we wind up with the results shown in Figure 3.14.

The first thing we can notice is that the tuned hyperparameters (the right plot) do not give us the prettiest graph. This does not mean that this prediction is incorrect. In fact, it is more correct than the plot with the manually chosen hyperparameters (left). The plot with the tuned hyperparameters basically says, 'I think the graph of the value function is somewhat curved, but I do not have enough data yet to be fully certain, because there is

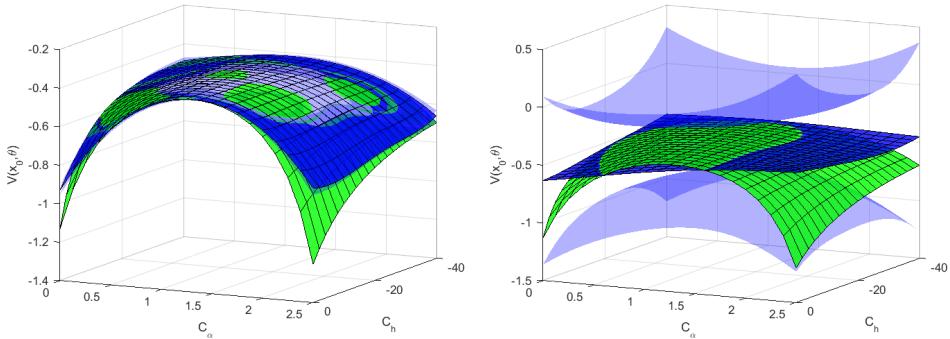


Figure 3.14: Approximation of the value function of the linear pitch-plunge system subject to process noise. Both the exact expected value $\mathbb{E}[V]$ and the GP approximation of it are shown. A number of $n_m = 500$ experiments was run, starting from a random initial state, using a random controller setting and lasting $T = 1$ s. The left figure used a chosen value of $\hat{\sigma}_c = 10^{-4}$, while the right figure tuned $\hat{\sigma}_c$ to 0.20. The plot was subsequently made by averaging over $\tilde{x}_0 \sim \mathcal{N}(\mathbf{0}, \Sigma_0)$ and letting C_h and C_α vary. Both $C_{\dot{h}}$ and $C_{\dot{\alpha}}$ have been kept at zero.

too much noise.' In the meantime the plot with the manually chosen hyperparameters more or less claims, 'The graph is definitely curved. I'm 100% certain about this. What noise are you talking about?' It is overly certain of its estimates, even though a somewhat conservative prediction would be more appropriate here. Especially when $C_h = 0$, its estimates are off by several times the uncertainty bounds, which should not be the case for a proper prediction. So while its estimates may be more accurate, its *integrity* – how honest it is about the uncertainties of its estimates – is severely lacking.

A second thing we notice is that the value function mostly depends on C_α and less on C_h . As long as $C_\alpha = 1.2$, the value of C_h is mostly irrelevant. The predicted optimum by the GP (taken at the maximum of the mean of the prediction) equals $C_h = -10.5$ and $C_\alpha = 1.19$ for the left plot, while it is $C_h = -19.6$ and $C_\alpha = 1.22$ for the right plot. Though the latter is further away from the optimal controller gains, these controller gains would still result in a value that is close to optimal, meaning that the provided controller gains are still of good quality.

From our experiments we can hence conclude that it is possible to tune a controller based on estimates of the value function. But another possibly more important conclusion is that the whole process of choosing a covariance function and tuning the hyperparameters is very complicated. Sometimes hyperparameter tuning works well. Sometimes it gives the wrong hyperparameters because there is not enough data. And sometimes it gives the right hyperparameters but an ugly plot, while the wrong hyperparameters give a more pretty plot. As always, having an intuitive understand of what every single parameter means really helps while applying Gaussian process regression.

3.5.5. FURTHER EXTENSIONS: UNSTABLE AND NONLINEAR SYSTEMS

After our successful estimations so far, we would like to extend the problem further. There are multiple ways to do so. We can include all controller parameters, switch from the linearized to the nonlinear system or even take into account a varying wind speed. All these methods require a significant overhaul of our set-up though.

INCLUDING UNSTABLE SYSTEMS

Suppose that we also include controller settings θ_c that may cause the system to go unstable. In this case, our value function $\underline{V}(\tilde{x}_0, \theta)$ will not be smoothly varying anymore. The reason is that, as we get close to instability, \underline{V} will go towards $-\infty$. But as we wind up with an unstable system, the value \underline{V} resulting from our equations suddenly becomes positive, starting at $+\infty$ and reducing to a smaller positive value for more unstable systems.

To see why this happens, consider the simple system $\dot{x} = ax + bu$, with reward $r(x, u) = -qx^2 - ru^2$, where $a = b = q = r = 1$. The Lyapunov equation of the system subject to $u = -fx$ now becomes $2(a - bf)\bar{X} + q + f^2r = 0$. Imagine what happens with the solution $\bar{X} = -\frac{q+f^2r}{2(a-bf)}$, and hence the value $V = \bar{X}x^2$, as $(a - bf)$ goes from stable (negative) to unstable (positive). It goes from being negative, to $-\infty$, to ∞ and subsequently being positive.

Though it is nice that we can see from the value \underline{V} when the system is unstable, and how unstable it is, it does mean that \underline{V} is not smoothly varying with respect to the controller settings θ_c anymore. As such, we need to either adjust the covariance function we use, or adjust the function we are approximating.

One idea would be to approximate the quantity $-\frac{1}{\underline{V}}$ instead of \underline{V} itself. This quantity is always positive for stable systems, and the larger it is, the better the value. In addition, the transition from stable to unstable systems becomes smooth. However, contrary to (3.57), we now do not get a relation that is linear in this new value-like quantity, which brings us some extra challenges. I will leave it as a subject for future research.

SWITCHING TO THE NONLINEAR SYSTEM

A completely different extension would be to switch from the linearized pitch-plunge system to the nonlinear variant. In theory we could do this transition without any extra adjustments, except for one fundamental problem.

In the set-up of our covariance function, we have assumed that the value \underline{V} varies quadratically with the initial state \tilde{x}_0 . Though this holds for linear systems, it naturally does not have to be the case for nonlinear systems. In particular, the flutter behavior present in the pitch-plunge system is known as a phenomenon that does occur from certain initial states but not from others. As such, the way in which the value \underline{V} depends on the initial state \tilde{x}_0 is certainly not quadratic. To be precise, if we are near the edge of a ‘flutter region’ in the state space, the value function \underline{V} is unlikely to even be smoothly varying.

As such, coming up with a suitable covariance function for this problem is a completely new problem. I would recommend starting with the squared exponential covariance function, as was done by [Bijl et al. \(2014\)](#) for a different problem. Although trying other covariance functions like the Matérn covariance function (see [Cornford et al. \(2002\)](#), [Rasmussen and Williams \(2006\)](#) for more information on this covariance function) could also be worthwhile.

INCLUDING THE WIND SPEED

So far we have kept the wind speed constant at $U = 10\text{m/s}$. In reality this wind speed will vary too. And though the wind speed is known, we cannot control it. We can of

course take it into account in our approximations. To do so, we include it as an additional parameter $\underline{V}(\tilde{x}_0, \theta, U)$ for our value function.

It is important to keep in mind here that the dynamics of the system change for higher wind speeds. In particular, for wind speeds larger than $U = 11.6\text{ m/s}$, the linear system will be unstable. The nonlinear system, though not diverging, will either flutter or find a new equilibrium point. Finding ways to take this into account in the covariance function will be crucial.

3

In general, the above schemes require a new covariance function and a larger input space for the approximation of \underline{V} . As a result, a lot more measurements will be needed before accurate predictions can be made. Taking into account large amounts of measurements will result in a whole new problem of its own though: the computational time required by GP regression will increase to the point where it is no longer practically feasible. Getting around that problem will be the subject of the next chapter.

3.6. OVERVIEW OF LITERATURE

We discussed three different subjects in this chapter: hyperparameter tuning, covariance function selection and constrained GP regression. We will look at the state of the literature of these three topics separately.

3.6.1. LITERATURE ON HYPERPARAMETER TUNING

Estimation of hyperparameters has been an issue in many fields for quite a while now. For instance in spatial statistics, where the habit generally was to optimize the likelihood to estimate things like length scales and noise strength (see [Mardia and Marshall \(1984\)](#), [Kaufman and Shaby \(2013\)](#)). Or within neural networks, where regularization parameters needed to be used to prevent overfitting (see [MacKay \(1999\)](#), [Bergstra et al. \(2011\)](#)). Or for support vector machines, where the parameters of the kernel needed to be chosen (see [Chapelle et al. \(2002\)](#), [Hsu et al. \(2003\)](#)). Different optimization methods have been used, ranging from manual search, grid search [Larochelle et al. \(2007\)](#) and random search [Bergstra and Bengio \(2012\)](#) up to gradient ascent methods [Blum and Riedmiller \(2013\)](#) and Bayesian methods [Snoek et al. \(2012\)](#).

To tune hyperparameters for Gaussian processes, the same methods are generally used. In fact, because the general framework for tuning hyperparameters also works well for GP regression, relatively little work has been done specifically related to GP hyperparameter tuning. Early work was done by [Seeger \(2000\)](#), [Sundararajan and Keerthi \(2000\)](#), although the most influential work here is the book of [Rasmussen and Williams \(2006\)](#) (Chapter 5), which summarized the methods quite well. Its main focus lies on gradient ascent methods to tune hyperparameters, which is also the method we apply in this thesis. In fact, most of the theory discussed in Section 3.1 comes from [Rasmussen and Williams \(2006\)](#).

3.6.2. LITERATURE ON COVARIANCE FUNCTIONS

A lot of work has been done on covariance functions. An early overview of possible covariance functions was given by [Abrahamsen \(1997\)](#). It already mentioned, among others, the (squared) exponential and various Bessel covariance functions.

There were various contributions by others as well. It was argued by Stein (1999) that the squared exponential covariance function resulted in a degree of smoothness that in practice never really occurs. He instead argued for the Matérn covariance function, which was then successfully applied by Cornford et al. (2002). At the same time the periodic covariance function was studied by MacKay (1998), Schölkopf and Smola (2002) and the neural network covariance function by Neal (1996), while Gibbs (1997) looked into anisotropic covariance functions, in which the length scale can vary over the input space.

A proper overview of all this work was eventually given in the book by Rasmussen and Williams (2006) (Chapter 4). Next to looking at various examples of covariance functions, this book also discusses general properties that covariance functions can or must have, making it quite a lot more in-depth than the basic introduction to covariance functions given in Section 3.2.

After Rasmussen and Williams (2006), the main contributions did not so much concern new covariance functions, but instead were about how to find the right covariance function, or combine different covariance functions together to get even better covariance functions. Interesting work here was done by Bach (2009), Hinton and Salakhutdinov (2008), Duvenaud et al. (2013).

3.6.3. LITERATURE ON CONSTRAINED GP REGRESSION

On constrained GP regression, there is very few literature whatsoever. It is a technique I devised myself and first published about in Bijl et al. (2014). I did find a technique very similar to what was discussed in Section 3.3, developed by Engel et al. (2003, 2005). Also slightly related is the work by Salzmann and Urtasun (2010) who constrain the different outputs of a multi-output function $f(\mathbf{x})$ for the same input point. So the idea of constrained GP regression is not entirely new. But the way in which it has been generalized like in Section 3.3 is quite novel.

A related and promising approach was explored quite recently by Jidling et al. (2017). Here, the idea was to add linear constraints on not just the function $f(\mathbf{x})$ we are approximating, but also on the derivatives. For instance, we can require that $\partial f / \partial x_1 - \partial f / \partial x_2 = 0$. This constraint is then incorporated into the covariance function, such that the GP only considers functions satisfying this constraint in the first place, even before any measurement is incorporated.

4

SPARSE AND ONLINE GAUSSIAN PROCESS REGRESSION

Summary — When the number of measurements n_m becomes big, Gaussian process regression runs into computational problems. The reason is the required runtime, which is cubic in the number of measurements.

To solve this, we can separate the regression algorithm into two parts. First we predict the distribution of the so-called inducing function values \underline{f}_u corresponding to inducing input points X_u . Then we use this distribution to make the actual predictions of the trial function values \underline{f}_ . Alternatively, we can use measurements one by one or in groups to find the distribution of \underline{f}_u . This results in the FITC and the PITC algorithm, respectively, whose runtime is linear in the number of measurements.*

When measurements come in one by one, both these algorithms can be applied in an online fashion, where we incorporate measurements one by one into the distribution of \underline{f}_u . The runtimes of these online algorithms are the same as those of the offline algorithms, but because we do not have to remember all measurements anymore, the memory requirements are lower.

The positions of the inducing input points can be chosen based on expert knowledge, or tuned automatically. Automatic tuning can be done either by optimizing the log-likelihood of the measurements (evidence maximization) or the posterior variance of the trial points. In addition, when extra accuracy is required, additional inducing input points can also be added online.

The derived algorithms have been proven to work by applying them to various example problems. On lower-dimensional problems, all algorithms have a comparable accuracy, given the same number of measurements. On higher-dimensional problems the assumptions behind the FITC and PITC algorithms do cause a reduced accuracy, but this may be worthwhile given the significantly reduced runtime of these algorithms.

This chapter is all about applying Gaussian process regression to big and expanding data sets. Regular Gaussian process regression as we have learned it so far is not capable of dealing with this, for various reasons, but there are ways to work around this.

We will start to look at methods to reduce the computational requirements of Gaussian process regression (Section 4.1). Next, we consider ways of adding new measurements in an online way, without having to redo all our calculations (Section 4.2). Fundamental to both these methods are the inducing input points and we will also examine ways to choose/tune them (Section 4.3). We then apply the methods we derived (Section 4.4) and in the end take a look at the available literature on this subject (Section 4.5).

4.1. SPARSE GAUSSIAN PROCESS REGRESSION

4

When using Gaussian process regression on big data sets, the *computational requirements* start to become important. Especially when the number of measurements n_m becomes larger than a thousand, you will notice that the algorithm starts to become very slow. In addition, your computer may also run out of allocated memory.

Sparse Gaussian process regression is all about reducing these computational requirements, both on runtime and on memory. The idea is to take advantage of the structure in our matrices, or to actually create a structure in our matrices, such that our equations can be calculated more efficiently.

We first look into what exactly we mean with computational requirements and how we can compare them (Section 4.1.1). Then we analyze the computational requirements of the regular GP regression algorithm (Section 4.1.2). We separate this algorithm into two steps: training and prediction. We then first reduce the runtime of the prediction step by introducing inducing input points (Section 4.1.3), and then we reduce the requirements of the training step by using measurements individually (Section 4.1.4). Finally we generalize this new method, allowing us to use measurements in groups (Section 4.1.5).

4.1.1. A NOTATION FOR DISCUSSING COMPUTATIONAL REQUIREMENTS

Let's consider the Gaussian process regression equation (2.30). If we only want to predict the posterior distribution of the trial function values \underline{f}_* , and not that of the measurement function values \underline{f}_m , then it becomes

$$\begin{aligned}\underline{f}_* &\sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_{**}) \\ \Sigma_{**} &= K_{**} - K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1}K_{m*}, \\ \boldsymbol{\mu}_* &= \mathbf{m}_* + K_{*m}(K_{mm} + \hat{\Sigma}_{f_m})^{-1}(\hat{\mathbf{f}}_m - \mathbf{m}_m).\end{aligned}\tag{4.1}$$

Note that we use \mathbf{m} and K to indicate the properties of prior distributions, while we use $\boldsymbol{\mu}$ and Σ for posterior distributions.

A question we could ask ourselves is ‘How long does it take to evaluate the above equation? And how much memory is required for this?’ However, these questions are rather vague. Their answers will of course depend on the number of measurement points n_m and the number of trial points n_* , but also on the speed of our computer, the way in which it implements matrix algebra, and so on.

Let's make this more specific. For calculations we only look at the number of basic calculations that we need to do, while for memory we only look at the number of parameters we need to store. For instance, setting up the $n_m \times n_m$ matrix K_{mm} costs n_m^2 runtime and n_m^2 memory.

At this point you may be thinking, ' K_{mm} is symmetric! Isn't the memory required equal to $\frac{1}{2}n_m(n_m + 1) = \frac{1}{2}n_m + \frac{1}{2}n_m^2$ then?' Technically you are correct here. However, when comparing computational requirements, any multiplying constants (like $\frac{1}{2}$) are not very important, because we can solve them by taking a little bit faster computer, possibly with a few extra cores. Also, any lower-order terms like n_m are not important when there are higher-order terms like n_m^2 around. After all, when n_m becomes big enough, any term $c_1 n_m$ will be negligible compared to $c_2 n_m^2$, even when c_1 is larger than c_2 .

Keeping this in mind, we can now say that the *memory requirement* for setting up the matrix K_{mm} is of the order n_m^2 . We write this using the *big O notation* as $\mathcal{O}(n_m^2)$. Similarly, the *runtime requirement* of calculating K_{mm} is $\mathcal{O}(n_m^2)$. The memory requirement and runtime requirement together are the most important *computational requirements* we need to take into account when calculating with matrices.

4.1.2. ANALYZING THE COMPUTATIONAL REQUIREMENTS

Let's consider (4.1) once more. What is the runtime to calculate this? And how much memory is required? To analyze this, we can apply the following rules.

- Storing an $m \times n$ matrix requires $\mathcal{O}(mn)$ memory.
- Storing a diagonal $n \times n$ matrix requires $\mathcal{O}(n)$ memory.
- Adding/subtracting two $m \times n$ matrices takes $\mathcal{O}(mn)$ time.
- Multiplying an $l \times m$ matrix by an $m \times n$ matrix takes $\mathcal{O}(lmn)$ time.
- Multiplying an $m \times n$ matrix by a vector of size n takes $\mathcal{O}(mn)$ time.
- Multiplying an $m \times n$ matrix by a diagonal $n \times n$ matrix takes $\mathcal{O}(mn)$ time.
- Inverting an $n \times n$ matrix takes $\mathcal{O}(n^3)$ time¹.
- Inverting a diagonal $n \times n$ matrix takes $\mathcal{O}(n)$ time.

Using these results, we can see that inverting $K_{mm} + \hat{\Sigma}_{f_m}$ takes $\mathcal{O}(n_m^3)$ time, while left-multiplying this by K_{*m} takes $\mathcal{O}(n_* n_m^2)$ time. Calculating μ_* hence takes $\mathcal{O}(n_* n_m^2 + n_m^3)$ time, while calculating Σ_* takes $\mathcal{O}(n_*^2 + n_* n_m^2 + n_m^3)$ time.

In practice it often happens that we first obtain our measurement data X_m and \hat{f}_m , but only later on receive the trial input points X_* for which we want to predict the corresponding function values f_* . In this case, when we just obtained X_m and \hat{f}_m , we can already prepare ourselves for making predictions. This is called the *training step* (or sometimes the *preparation step*) of the algorithm. For regular GP regression this comes down to calculating $(K_{mm} + \hat{\Sigma}_{f_m})^{-1}$. When we then obtain the trial input points X_* , we can more easily do the *prediction step*: calculating μ_* and Σ_* .

¹This is the runtime using *Gauss-Jordan elimination*. There are other algorithms available, like optimized versions of the *Coppersmith-Winograd algorithm*, that can invert an $n \times n$ matrix or multiply two $n \times n$ matrices in $\mathcal{O}(n^{2.373})$ runtime. For more information on this, see the work by [Davie and Stothers \(2013\)](#), [Gall \(2014\)](#). To keep the discussion intuitive and easy to read, we will ignore these possible computational gains.

We can analyze the runtime of both the training and the prediction steps of the regular GP regression algorithm, as well as the memory that is required altogether. When we do, we get the results shown in Table 4.1. Here we also find the computational requirements of the other algorithms we will look at later on, both in Section 4.1 (offline) and Section 4.2 (online).

Table 4.1: Computational requirements of the various algorithms, sorted by number of simplifying assumptions. (\mathcal{O} is not written out explicitly.) The runtime is the total runtime of adding the first n_m measurement points. n_u is the number of inducing input points and n_* the number of trial points, where we assume that $n_u < n_* < n_m$. For the PITC algorithm we assume that the subsets X_{m_i} do not grow larger than n_u data points.

Algorithm	Section		Training runtime		Prediction runtime		Memory	
	Offline	Online	Offline	Online	Offline	Online	Offline	Online
Regular GP regression	4.1.2	4.2.1	n_m^3	n_m^3	$n_m^2 n_*$	$n_m^2 n_*$	n_m^2	n_m^2
Sparse GP regression	4.1.3	4.2.2	n_m^3	n_m^3	$n_u n_*^2$	$n_u n_*^2$	n_m^2	n_m^2
PITC regression	4.1.5	4.2.4	$n_m n_u^2$	$n_m n_u^2$	$n_u n_*^2$	$n_u n_*^2$	$n_m n_u$	n_u^2
FITC regression	4.1.4	4.2.3	$n_m n_u^2$	$n_m n_u^2$	$n_u n_*^2$	$n_u n_*^2$	$n_m n_u$	n_u^2

4.1.3. FASTER PREDICTION: USING INDUCING INPUT POINTS

We will first suppose that we have ample time for training, but really need to make fast predictions. Calculating μ_* currently takes $\mathcal{O}(n_m n_*)$ time, because we need to add up n_m basis functions for every trial point. Calculating Σ_* even takes $\mathcal{O}(n_m^2 n_*)$ time. When n_m is big, this is unacceptable.

We can solve this problem by choosing a number n_u of *inducing input points*². We write these as $\mathbf{x}_{u_1}, \dots, \mathbf{x}_{u_{n_u}}$ and together they form the *inducing input set* X_u . During our training time, we then predict the posterior distribution of the *inducing function values* $\underline{\mathbf{f}}_u$. Afterwards, we throw away all our measurement data and only use this posterior distribution of $\underline{\mathbf{f}}_u$ to predict the trial function values $\underline{\mathbf{f}}_*$.

Mathematically this method comes down to assuming that $\underline{\mathbf{f}}_m$ and $\underline{\mathbf{f}}_*$ are conditionally independent, given the value of $\underline{\mathbf{f}}_u$. That is,

$$p(\underline{\mathbf{f}}_m, \underline{\mathbf{f}}_* | \underline{\mathbf{f}}_u) = p(\underline{\mathbf{f}}_m | \underline{\mathbf{f}}_u) p(\underline{\mathbf{f}}_* | \underline{\mathbf{f}}_u). \quad (4.2)$$

This assumption is known as the *inducing input assumption*. It implies that the covariance K_{m*} between the measurement function values and the trial function values has to equal $K_{mu} K_{uu}^{-1} K_{u*}$ (see Theorem B.29). And from the assumption we can derive our new regression equations (see Theorem B.30). However, there is also a more intuitive view on this (see Theorem B.31) which we will look at now.

Earlier we saw that the GP regression equation (2.30) can be found by merging the prior distribution (2.22) with the measured distribution (2.29). We will now do something similar. We take our prior distribution

$$\begin{bmatrix} \underline{\mathbf{f}}_m \\ \underline{\mathbf{f}}_u \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \underline{\mathbf{f}}_m \\ \underline{\mathbf{f}}_u \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_m \\ \mathbf{m}_u \end{bmatrix}, \begin{bmatrix} K_{mm} & K_{mu} \\ K_{um} & K_{uu} \end{bmatrix}\right), \quad (4.3)$$

²This terminology was introduced by [Candela and Rasmussen \(2005\)](#) and we will stick with it.

and we merge this together with our measured distribution

$$\begin{bmatrix} \underline{\mathbf{f}}_m \\ \underline{\mathbf{f}}_u \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{f}_m \\ \mathbf{f}_u \end{bmatrix} \middle| \begin{bmatrix} \hat{\mathbf{f}}_m \\ * \end{bmatrix}, \begin{bmatrix} \hat{\Sigma}_{f_m} & * \\ * & \infty \end{bmatrix} \right). \quad (4.4)$$

When we do, we wind up with an equation identical to the GP regression equation (2.30), being the *sparse GP training equation*

$$\begin{aligned} \begin{bmatrix} \underline{\mathbf{f}}_m \\ \underline{\mathbf{f}}_u \end{bmatrix} &\sim \mathcal{N} \left(\begin{bmatrix} \mathbf{f}_m \\ \mathbf{f}_u \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_u \end{bmatrix}, \begin{bmatrix} \Sigma_{mm} & \Sigma_{mu} \\ \Sigma_{um} & \Sigma_{uu} \end{bmatrix} \right), \quad (4.5) \\ \begin{bmatrix} \Sigma_{mm} & \Sigma_{mu} \\ \Sigma_{um} & \Sigma_{uu} \end{bmatrix} &= \begin{bmatrix} K_{mm} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} \hat{\Sigma}_{f_m} & \hat{\Sigma}_{f_m} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \\ K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} \hat{\Sigma}_{f_m} & K_{uu} - K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_u \end{bmatrix} &= \begin{bmatrix} \Sigma_{mm} (K_{mm}^{-1} \mathbf{m}_m + \hat{\Sigma}_{f_m}^{-1} \hat{\mathbf{f}}_m) \\ \mathbf{m}_u + K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m) \end{bmatrix}. \end{aligned}$$

4

Through this we can find the posterior distribution of the inducing function values $\underline{\mathbf{f}}_u$ during our training step. We would then of course only calculate $\boldsymbol{\mu}_u$ and Σ_{uu} and not the other terms.

Next, we switch to the prediction step where we know X_* . Instead of using the measurement data, we now use the prior distribution of $\underline{\mathbf{f}}_u$ and $\underline{\mathbf{f}}_*$. This equals

$$\begin{bmatrix} \underline{\mathbf{f}}_u \\ \underline{\mathbf{f}}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{f}_u \\ \mathbf{f}_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_u \\ \mathbf{m}_* \end{bmatrix}, \begin{bmatrix} K_{uu} & K_{u*} \\ K_{*u} & K_{**} \end{bmatrix} \right). \quad (4.6)$$

We then merge this prior distribution together with the inducing function value distribution that we just found. That is,

$$\begin{bmatrix} \underline{\mathbf{f}}_u \\ \underline{\mathbf{f}}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{f}_u \\ \mathbf{f}_* \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_u \\ * \end{bmatrix}, \begin{bmatrix} \Sigma_{uu} & * \\ * & \infty \end{bmatrix} \right). \quad (4.7)$$

This does *not* directly give us our final result for $\underline{\mathbf{f}}_*$ though. The reason is that our prior knowledge of $\underline{\mathbf{f}}_u$, being $\underline{\mathbf{f}}_u \sim \mathcal{N}(\mathbf{m}_u, K_{uu})$, has been used in both of the above distributions. If we would merge both distributions together, we would use our prior knowledge twice, which is not allowed. To make sure that we do not, we need to *unmerge* this prior distribution

$$\begin{bmatrix} \underline{\mathbf{f}}_u \\ \underline{\mathbf{f}}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{f}_u \\ \mathbf{f}_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_u \\ * \end{bmatrix}, \begin{bmatrix} K_{uu} & * \\ * & \infty \end{bmatrix} \right). \quad (4.8)$$

This idea of unmerging distributions is the inverse of merging distributions together. (See Theorem B.23 for how it works with Gaussian distributions.) When we apply this

(see Theorem B.31) we wind up with the *sparse GP prediction equation*

$$\begin{aligned} \begin{bmatrix} \underline{\mathbf{f}}_u \\ \underline{\mathbf{f}}_* \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_u \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \Sigma_{uu} & \Sigma_{u*} \\ \Sigma_{*u} & \Sigma_{**} \end{bmatrix}\right), \\ \begin{bmatrix} \Sigma_{uu} & \Sigma_{u*} \\ \Sigma_{*u} & \Sigma_{**} \end{bmatrix} &= \begin{bmatrix} \Sigma_{uu} & \Sigma_{uu} K_{uu}^{-1} K_{u*} \\ K_{*u} K_{uu}^{-1} \Sigma_{uu} & K_{**} - K_{*u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}) K_{uu}^{-1} K_{u*} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_u \\ \boldsymbol{\mu}_* \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu}_u \\ \mathbf{m}_* + K_{*u} K_{uu}^{-1} (\boldsymbol{\mu}_u - \mathbf{m}_u) \end{bmatrix}. \end{aligned} \quad (4.9)$$

With this equation, and assuming we have calculated K_{uu}^{-1} in advance, we can calculate the posterior distribution of $\underline{\mathbf{f}}_*$ in $\mathcal{O}(n_u n_*^2)$ time. The whole process is visualized in Figure 4.1. Since n_u is generally much smaller than n_* , which in turn is smaller than n_m , this is a significant improvement.

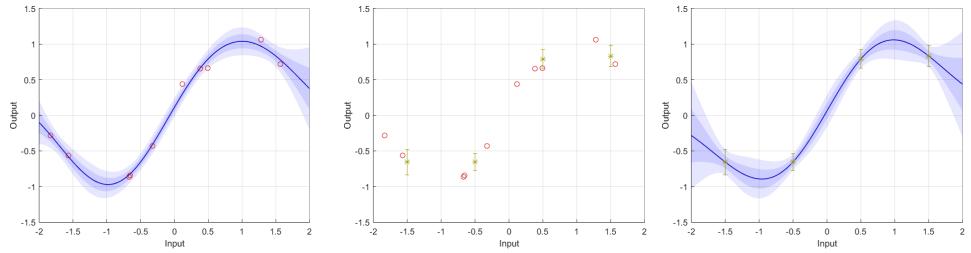


Figure 4.1: The comparison between regular and sparse GP regression. We used $n_m = 10$ measurements of $f(x) = \sin(2\pi \frac{x}{4})$ with noise strength $\hat{\sigma}_{fm} = 0.1$. The left plot shows regular GP regression: we use our measurements to directly predict $n_* = 100$ trial function values $\underline{\mathbf{f}}_*$. The middle and right plot shows the two steps of sparse GP regression. In the training phase we use the measurements to predict $n_u = 4$ inducing function values $\underline{\mathbf{f}}_u$. In the prediction phase we then ignore our measurements and only use the inducing function values $\underline{\mathbf{f}}_u$ to predict the trial function values $\underline{\mathbf{f}}_*$. If you look carefully, you can notice a slight accuracy decrease compared to regular GP regression for trial points that are far away from any inducing input points.

We will call the algorithm that we have just set up the *sparse GP regression algorithm*. This name is ambiguous, because ‘sparse GP regression’ is generally used to refer to any algorithm that tries to use structure in matrices to reduce the computational complexity of GP regression. So technically this name also incorporates the next few algorithms we will consider. However, as was also shown by [Candela and Rasmussen \(2005\)](#), nearly all sparse GP regression methods use inducing input points, and while other algorithms still use additional assumptions, and as a result have their own specific names, the above regression algorithm contains only the fundamental idea of sparse GP regression.

The question remains how to choose the inducing input points. We will look deeper into this in Section 4.3. For now we only note that if we would set X_u equal to X_m , then the sparse GP regression algorithm reduces to the regular GP regression algorithm. So effectively, we can see this algorithm as a generalization of the regular GP regression algorithm.

4.1.4. FASTER TRAINING: USING MEASUREMENTS INDIVIDUALLY

With the sparse GP regression algorithm we have made the prediction step faster, but not the training step. For large n_m this still takes a significant amount of time. The reason is that, when predicting \underline{f}_u , we combine all n_m measurements. This results in an $\mathcal{O}(n_m^3)$ runtime.

Instead, we could also use each measurement individually. So we first use measurement $(\mathbf{x}_{m_1}, \hat{f}_{m_1})$ to predict the distribution of \underline{f}_u . We write this as $\mathcal{N}(\boldsymbol{\mu}_u^1, \Sigma_{uu}^1)$, and its value can be found through (4.5). Then we use $(\mathbf{x}_{m_2}, \hat{f}_{m_2})$ to find $\mathcal{N}(\boldsymbol{\mu}_u^2, \Sigma_{uu}^2)$, and so on. This gives us n_m separate distributions for \underline{f}_u .

Next, we merge all these n_m distributions together. There is one caveat here. Each of these distributions of \underline{f}_u contains the information from the prior distribution $\underline{f}_u \sim \mathcal{N}(\mathbf{m}_u, K_{uu})$, but we are still only allowed to use this information once. To prevent us from using it n_m times, we need to unmerge the prior distribution $n_m - 1$ times. As a result, we get

$$\begin{aligned} \mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu}) &= \mathcal{N}(\mathbf{m}_u, K_{uu}) \oplus (\mathcal{N}(\boldsymbol{\mu}_u^1, \Sigma_{uu}^1) \ominus \mathcal{N}(\mathbf{m}_u, K_{uu})) \oplus \dots \\ &\quad \oplus (\mathcal{N}(\boldsymbol{\mu}_u^{n_m}, \Sigma_{uu}^{n_m}) \ominus \mathcal{N}(\mathbf{m}_u, K_{uu})) \\ &= \mathcal{N}(\boldsymbol{\mu}_u^1, \Sigma_{uu}^1) \oplus \dots \oplus \mathcal{N}(\boldsymbol{\mu}_u^{n_m}, \Sigma_{uu}^{n_m}) \underbrace{\ominus \mathcal{N}(\mathbf{m}_u, K_{uu})}_{(n_m - 1) \text{ times}}. \end{aligned} \quad (4.10)$$

This is a very intuitive expression, but actually calculating it takes some work. We can use this idea to derive a more simple and powerful expression though. If we define

$$\Lambda_{mm} = \text{diag}(K_{mm} - K_{mu} K_{uu}^{-1} K_{um}), \quad (4.11)$$

$$\Delta_{uu} = K_{uu} + K_{um} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu}, \quad (4.12)$$

then the resulting training equation (see Theorems B.32 through B.34) becomes

$$\begin{aligned} \begin{bmatrix} \mathbf{f}_m \\ \mathbf{f}_u \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_u \end{bmatrix}, \begin{bmatrix} \Sigma_{mm} & \Sigma_{mu} \\ \Sigma_{um} & \Sigma_{uu} \end{bmatrix}\right), \\ \Sigma_{mm} &= \left(\Lambda_{mm}^{-1} + \hat{\Sigma}_{f_m}^{-1}\right)^{-1} + \hat{\Sigma}_{f_m} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \Delta_{uu}^{-1} K_{um} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} \hat{\Sigma}_{f_m}, \\ \Sigma_{mu} &= \hat{\Sigma}_{f_m} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \Delta_{uu}^{-1} K_{uu}, \\ \Sigma_{um} &= K_{uu} \Delta_{uu}^{-1} K_{um} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} \hat{\Sigma}_{f_m}, \\ \Sigma_{uu} &= K_{uu} \Delta_{uu}^{-1} K_{uu}, \\ \begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_u \end{bmatrix} &= \begin{bmatrix} \mathbf{m}_m + \Sigma_{mm} \hat{\Sigma}_{f_m}^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m) \\ \mathbf{m}_u + \Sigma_{um} \hat{\Sigma}_{f_m}^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m) \end{bmatrix}. \end{aligned} \quad (4.13)$$

Through this equation, the training step only takes $\mathcal{O}(n_m n_u^2)$ time. And after we have found the distribution of \underline{f}_u in this way, we can still use the sparse GP prediction equation (4.9) to efficiently make predictions in $\mathcal{O}(n_u n_*^2)$ time. This is a significant runtime reduction, especially for large n_m . In addition, we do not have to calculate and store K_{mm} anymore, but only the diagonal matrix Λ_{mm} and the $n_m \times n_u$ matrix K_{mu} . As a

result, the memory required to run the regression algorithm is reduced from $\mathcal{O}(n_m^2)$ to $\mathcal{O}(n_m n_u)$. The entire process is visualized in Figure 4.2.

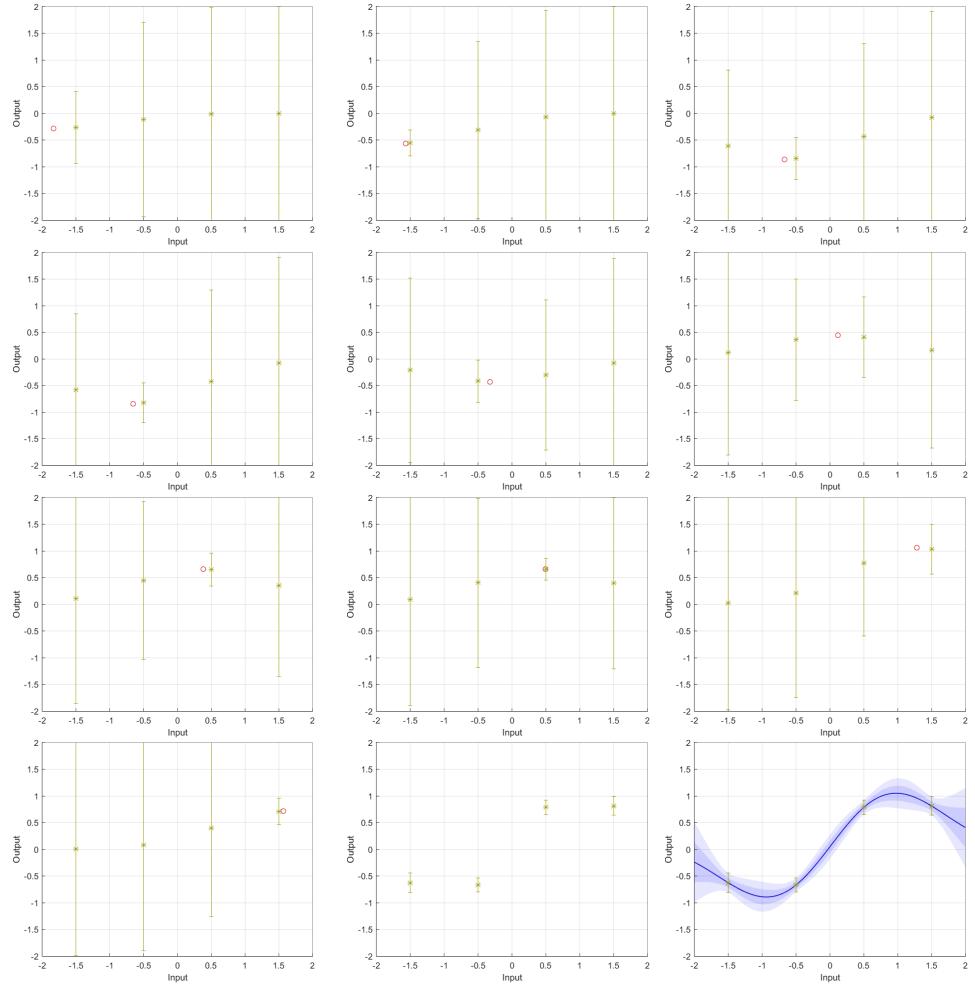


Figure 4.2: A visualization of the steps in the FITC regression algorithm. We use the same data as in Figure 4.1. We first use each measurement individually (the first ten plots) to predict the distribution of the inducing function values f_u . We subsequently merge all these individual distributions together into one single posterior distribution of f_u (bottom middle) which completes the training step. Then, just as in Figure 4.1, we use this distribution of the inducing function values f_u to predict the trial function values f_* (bottom right). The difference with the results from Figure 4.1 are small enough to be invisible.

What assumption are we making behind the scenes here though? Mathematically, we assume that, given the inducing function values f_u , each measurement function value $f_{m_1}, \dots, f_{m_{n_m}}$ is fully independent. That is,

$$p(f_{m_1}, f_{m_2}, \dots, f_{m_{n_m}} | f_u) = p(f_{m_1} | f_u) p(f_{m_2} | f_u) \dots p(f_{m_{n_m}} | f_u). \quad (4.14)$$

This assumption is called the *Fully Independent Training Conditional assumption* (FITC assumption) and the resulting algorithm is therefore known as the *FITC algorithm*. Because of this (4.13) is also known as the *FITC training equation*.

Previously, the inducing input assumption enforced K_{m*} to equal $K_{mu}K_{uu}^{-1}K_{u*}$. Similarly, the FITC assumption enforces $K_{m_i m_j}$ (for $i \neq j$) to equal $K_{m_i u}K_{uu}^{-1}K_{um_j}$, or equivalently it enforces K_{mm} to equal

$$\begin{aligned} K_{mm} &\leftarrow \text{diag}(K_{mm} - K_{mu}K_{uu}^{-1}K_{um}) + K_{mu}K_{uu}^{-1}K_{um} \\ &= \Lambda_{mm} + K_{mu}K_{uu}^{-1}K_{um}. \end{aligned} \quad (4.15)$$

We could of course also use this value of K_{mm} in the regular regression equations we used to, instead of using (4.13). However, using (4.13) is computationally much more efficient, saving us a lot of time.

4

4.1.5. MORE FLEXIBILITY: USING MEASUREMENTS IN SUBGROUPS

We can extend the idea we just explored even further. Instead of using each measurement $(\mathbf{x}_{m_i}, \hat{f}_{m_i})$ individually, we can also use them in small subgroups of the full measurement set X_m .

Let's define X_{m_1} to contain the first n_{m_1} measurement input points, X_{m_2} to contain the next n_{m_2} measurement input points, and so on. Similarly, $\hat{\mathbf{f}}_{m_1}$ contains the first n_{m_1} elements of $\hat{\mathbf{f}}_m$, $\hat{\mathbf{f}}_{m_2}$ the next n_{m_2} elements, and so on. After having set up these subgroups, we predict the distribution of $\underline{\mathbf{f}}_u$ based on the first subgroup of measurements, do the same for the second subgroup of measurements, and so on. Finally, we merge all the results together and unmerge the prior distribution of $\underline{\mathbf{f}}_u$ the right amount of times to get the posterior distribution of $\underline{\mathbf{f}}_u$.

The resulting expression is actually exactly the same as the FITC training equation (4.13) (see Theorem B.35) except for one small difference. We should redefine the matrix Λ_{mm} of (4.11) to

$$\Lambda_{mm} = \text{blkdiag}(K_{mm} - K_{mu}K_{uu}^{-1}K_{um}). \quad (4.16)$$

Earlier the `diag` function sets all non-diagonal terms to zero, turning the given matrix into a diagonal matrix. Similarly, the `blkdiag` function here turns the given matrix into a block-diagonal matrix, setting all other elements to zero. This block-diagonal matrix is set up such that the first diagonal block is $n_{m_1} \times n_{m_1}$, the second block is $n_{m_2} \times n_{m_2}$, and so on. So every block corresponds to a subgroup of measurements.

After having calculated the distribution of $\underline{\mathbf{f}}_u$ in this way, we can of course still use the Sparse GP prediction equation (4.9) to predict $\underline{\mathbf{f}}_*$. The resulting algorithm is called the *Partially Independent Training Conditional algorithm* (PITC algorithm). Its process is visualized in Figure 4.3.

The assumption behind the PITC algorithm is known as the *PITC assumption*. It assumes that the measurement function values $\mathbf{f}_{m_1}, \mathbf{f}_{m_2}, \dots$ of each subgroup are conditionally independent given $\underline{\mathbf{f}}_u$. That is,

$$p(\mathbf{f}_{m_1}, \mathbf{f}_{m_2}, \dots | \underline{\mathbf{f}}_u) = p(\mathbf{f}_{m_1} | \underline{\mathbf{f}}_u)p(\mathbf{f}_{m_2} | \underline{\mathbf{f}}_u)\dots \quad (4.17)$$

It is very interesting to realize here that if we only take 'subgroups' of size 1, and hence have $n_{m_1} = n_{m_2} = \dots = 1$, then the PITC algorithm reduces back to the FITC algorithm.

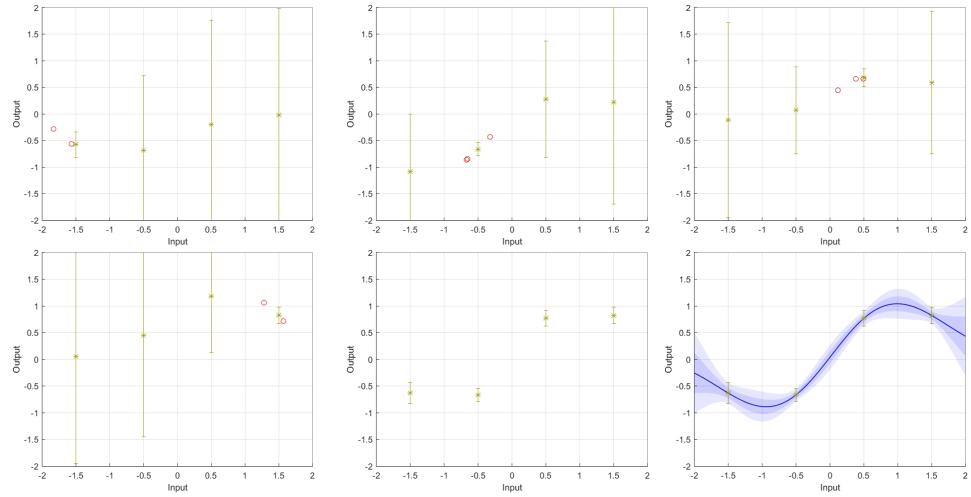


Figure 4.3: A visualization of the steps in the PITC regression algorithm. The set-up is the same as in Figure 4.2, except now we use measurements in small subgroups.

However, if we use one enormous ‘subgroup’ containing all n_m measurement points, then the PITC algorithm becomes the sparse GP algorithm. This makes the PITC algorithm a generalization of the two algorithms, where we can blend between them based on which subgroups of measurements we use.

What subgroups should we use though? It is important to realize here that, if you put measurement points \mathbf{x}_m together in a subgroup that do not have any covariance amongst each other, then you do not gain any additional accuracy in your predictions. You might as well have kept them in separate subgroups. So in practice it is best to make subgroups of measurement points that are as strongly correlated amongst each other as possible. This gives you the smallest accuracy loss with respect to the sparse GP regression algorithm.

Finally, let’s look at the runtime of the training phase of the PITC algorithm. Naturally, this depends on the size of the subgroups. Assuming that none of the subgroups is larger than n_u measurement points, the training runtime is the same as that of the FITC algorithm, being $\mathcal{O}(n_m n_u^2)$. Of course, if we only use one subgroup containing all measurements, we have the same runtime as the sparse GP algorithm, which is $\mathcal{O}(n_m^3)$.

4.1.6. AN INCORRECT ALTERNATIVE VIEW ON SPARSE GP REGRESSION

In GP regression we are predicting the distribution of $f_{\mathbf{x}^*} = f(\mathbf{x}_*)$. The mean prediction is given by μ_* . When we use sparse GP regression, then μ_* always takes the form of

$$\mu_* = m_* + K_{*u} K_{uu}^{-1} (\boldsymbol{\mu}_u - \mathbf{m}_u). \quad (4.18)$$

Let's define the weights $\mathbf{w} = K_{uu}^{-1}(\boldsymbol{\mu}_u - \mathbf{m}_u)$ and the feature vectors $\phi_i(\mathbf{x}) = k(\mathbf{x}_{ui}, \mathbf{x})$. We can now also write our posterior prediction as

$$\mu_* = m(\mathbf{x}_*) + \sum_{i=1}^{n_u} w_i \phi_i(\mathbf{x}_*) = m(\mathbf{x}_*) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_*), \quad (4.19)$$

where we have $\boldsymbol{\phi}(\mathbf{x}_*) = k(X_u, \mathbf{x}_*)$. This means that μ_* effectively is a function that is linear in given feature functions! Can we then also use the theory from Section 3.2.2 to find \mathbf{w} and subsequently approximate μ_* ?

The answer is 'Yes we can, but then we are solving a rather different problem.' Before we look into why, let's just try this solution method and see what we wind up with.

We can find the posterior distribution of the weights through (3.29). In this expression we do have to replace X_m by the feature matrix Φ_m , which in turn equals $\boldsymbol{\phi}(X_m) = k(X_u, X_m) = K_{um}$. As such, we have

$$\begin{aligned} \underline{\mathbf{w}} &\sim \mathcal{N}(\boldsymbol{\mu}_w, \Sigma_w), \\ \Sigma_w &= \left(K_{um} \hat{\Sigma}_{fm}^{-1} K_{mu} + K_w^{-1} \right)^{-1}, \\ \boldsymbol{\mu}_w &= \Sigma_w K_{um} \hat{\Sigma}_{fm}^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m). \end{aligned} \quad (4.20)$$

As prior weight covariance matrix K_w , we can best pick

$$K_w = \mathbb{E}[\underline{\mathbf{w}} \underline{\mathbf{w}}^T] = K_{uu}^{-1} \mathbb{E}\left[(\underline{\mathbf{m}}_u - \mathbf{m}_u)(\underline{\mathbf{m}}_u - \mathbf{m}_u)^T \right] K_{uu}^{-1} = K_{uu}^{-1}. \quad (4.21)$$

Note that, before we obtain any measurements, the posterior mean $\underline{\boldsymbol{\mu}}_u$ of the inducing function values is not yet specified, and hence is a random variable too. Using this, the posterior distribution of $\underline{\mathbf{f}}_*$ can now be found through

$$\begin{aligned} \underline{\mathbf{f}}_* &\sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_{**}), \\ \Sigma_{**} &= K_{*u} \Sigma_w K_{u*}, \\ \boldsymbol{\mu}_* &= K_{*u} \boldsymbol{\mu}_w. \end{aligned} \quad (4.22)$$

These equations have a runtime of $\mathcal{O}(n_m n_u^2)$, which makes them just as efficient as FITC regression and more efficient than sparse GP regression without the FITC assumption. When we apply them, we get the prediction shown in Figure 4.4.

In this figure we notice that the prediction we wind up with is mostly correct. However, the uncertainty is a lot smaller than what it used to be. This is actually a downside, and to see why, we need to examine what assumptions we have silently made.

As we have seen in Section 2.3.2, a Gaussian process is a distribution over functions. The covariance function specifies which functions are more likely and which are less likely. When we use the squared exponential covariance function, we say that smoothly varying functions are more likely and strongly fluctuating functions are less likely, but in theory all functions are still possible.

However, when setting up Figure 4.4, we have actually used the linear covariance function. This means that we have assumed that *only* functions $f(\mathbf{x})$ that are linear in

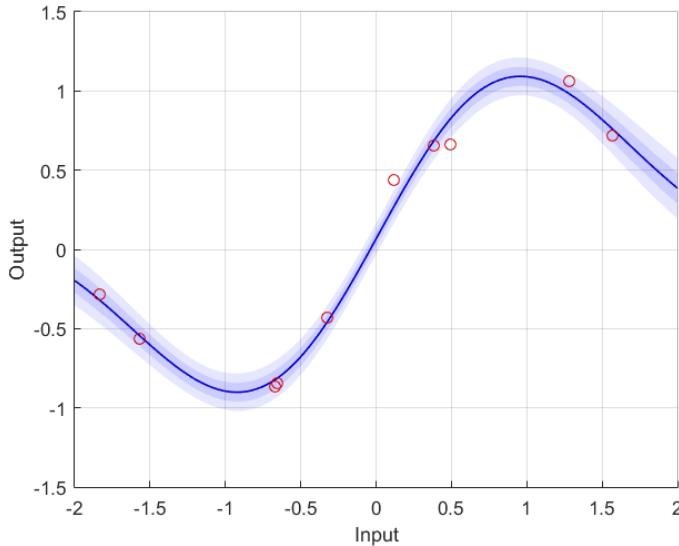


Figure 4.4: Using feature functions to approximate the function. K_w has been picked large enough for K_w^{-1} to be negligible. The result is a similar prediction as before, but with a much smaller covariance.

the features $\phi(\mathbf{x})$ are possible. We have assumed that other functions cannot take place at all. As such, we have severely restricted the possible functions that we can get.

Because there are now less possible functions we need to take into account, we have significantly simplified the problem. This explains both the reduced runtime and the higher certainty about our outcomes. However, if this assumption is not correct – if our true function $f(\mathbf{x})$ cannot be written as a weighted sum of feature functions $\phi(\mathbf{x})$ – then we will never find the true function. In our previous set-up this was not a problem: the algorithm would just add uncertainty to compensate for this, taking into account other potential functions. This new algorithm does not.

This is also confirmed by comparing the new algorithm we have just set up with the FITC algorithm. These algorithms have the exact same expressions, except for two changes: K_{uu} has turned into K_w^{-1} and Λ_{mm} has disappeared (turned into zero). The first difference was predicted by (4.21), but the second one is surprising. Λ_{mm} is representative of the part of the distribution of \underline{f}_m that cannot be explained using knowledge about \underline{f}_u . (See the intuitive view on Λ_{mm} just prior to Theorem B.32.) So by assuming that $\Lambda_{mm} = 0$, we basically assume (on top of the FITC assumption) that \underline{f}_m can be fully predicted if \underline{f}_u is given. Naturally, this assumption is generally false.

The conclusion is that this new set-up makes too many simplifying assumptions and as such provides predictions with a higher certainty than it has a right to. We would better steer clear of it.

4.2. ONLINE GAUSSIAN PROCESS REGRESSION

By now we know how to efficiently train a GP regression algorithm with n_m measurements. Now let's suppose that we already have done this training, but all of a sudden get another measurement. We denote this extra measurement by $(\mathbf{x}_+, \hat{f}_+)$ and the corresponding noise variance by $\hat{\sigma}_{f_+}^2$. We could of course add \mathbf{x}_+ to X_m , \hat{f}_+ to $\hat{\mathbf{f}}_m$ and $\hat{\sigma}_{f_+}^2$ to $\hat{\Sigma}_{f_m}$ and then redo all our calculations, but surely there must be some way to take into account our earlier training? How exactly can we do that?

That is what we will figure out in this chapter, starting with regular GP regression (Section 4.2.1), continuing with sparse GP regression (Section 4.2.2) and then extending these ideas to FITC (Section 4.2.3) and PITC (Section 4.2.4). Finally we still make a brief note on the numerical stability of these methods (Section 4.2.5).

4.2.1. REGULAR ONLINE GAUSSIAN PROCESS REGRESSION

In regular GP regression, the 'training phase' consists of calculating $(K_{mm} + \hat{\Sigma}_{f_m})^{-1}$. This is a large matrix inverse, so being able to recycle it could be useful.

Using the GP regression equation (2.30) (or equivalently (4.1)), we can find that

$$\begin{aligned}\underline{\mathbf{f}}_* &\sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_{**}) \\ \Sigma_{**}^+ &= K_{**} - [K_{*m} \quad K_{*+}] \left(\begin{bmatrix} K_{mm} & K_{m+} \\ K_{+m} & K_{++} \end{bmatrix} + \begin{bmatrix} \hat{\Sigma}_{f_m} & 0 \\ 0 & \hat{\sigma}_{f_+}^2 \end{bmatrix} \right)^{-1} \begin{bmatrix} K_{m*} \\ K_{+*} \end{bmatrix}, \\ \boldsymbol{\mu}_*^+ &= \mathbf{m}_* + [K_{*m} \quad K_{*+}] \left(\begin{bmatrix} K_{mm} & K_{m+} \\ K_{+m} & K_{++} \end{bmatrix} + \begin{bmatrix} \hat{\Sigma}_{f_m} & 0 \\ 0 & \hat{\sigma}_{f_+}^2 \end{bmatrix} \right)^{-1} \left(\begin{bmatrix} \hat{\mathbf{f}}_m \\ \hat{f}_+ \end{bmatrix} - \begin{bmatrix} \mathbf{m}_m \\ m_+ \end{bmatrix} \right).\end{aligned}\tag{4.23}$$

We use the superscript $^+$ to indicate that this is the posterior distribution of $\underline{\mathbf{f}}_*$ taking into account the new measurement $(\mathbf{x}_+, \hat{f}_+)$.

The problem with the above expression is that calculating the matrix inverse will take $\mathcal{O}(n_m^3)$. This means that every single added measurement will take $\mathcal{O}(n_m^3)$ to incorporate, which is unacceptable.

Luckily, we can do better. In the above expression, we already know $(K_{mm} + \hat{\Sigma}_{f_m})^{-1}$ from our previous calculations, and we can use that knowledge when calculating the new matrix inverse. To keep our notation short, let's write $\hat{K}_{mm} \equiv K_{mm} + \hat{\Sigma}_{f_m}$ and similarly $\hat{K}_{++} \equiv K_{++} + \hat{\sigma}_{f_+}^2$. Theorem A.6 now tells us that we have

$$\begin{aligned}\begin{bmatrix} \hat{K}_{mm} & K_{m+} \\ K_{+m} & \hat{K}_{++} \end{bmatrix}^{-1} &= \begin{bmatrix} \hat{K}_{mm}^{-1} + \hat{K}_{mm}^{-1} K_{m+} \Delta_{++}^{-1} K_{+m} \hat{K}_{mm}^{-1} & -\hat{K}_{mm}^{-1} K_{m+} \Delta_{++}^{-1} \\ -\Delta_{++}^{-1} K_{+m} \hat{K}_{mm}^{-1} & \Delta_{++}^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \hat{K}_{mm}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -\hat{K}_{mm}^{-1} K_{m+} \\ 1 \end{bmatrix} \Delta_{++}^{-1} \begin{bmatrix} -K_{+m} \hat{K}_{mm}^{-1} & 1 \end{bmatrix},\end{aligned}\tag{4.24}$$

where we have defined $\Delta_{++} = \hat{K}_{++} - K_{+m} \hat{K}_{mm}^{-1} K_{m+}$. Note that, because Δ_{++} is a scalar, it is very easy to invert it.

Let's analyze the runtime of this update law. The hardest part is calculating Δ_{++} . This still takes $\mathcal{O}(n_m^2)$ time. Inverting Δ_{++} subsequently takes $\mathcal{O}(1)$ time, while expanding the

matrix \hat{K}_{mm} to the above matrix takes $\mathcal{O}(n_m)$ time. Altogether, the update law hence takes $\mathcal{O}(n_m^2)$ time for every single added measurement. Adding n_m measurements will therefore take $\mathcal{O}(n_m^3)$, as is also noted in Table 4.1.

4.2.2. SPARSE ONLINE GAUSSIAN PROCESS REGRESSION

Can we set up a similar online version of the sparse GP algorithm of Section 4.1.3? The key in this algorithm is to properly keep track of the distribution of the inducing function values \underline{f}_u . That is, we only need to update Σ_{uu} and μ_u from (4.5). This is done, identically to (4.23), according to

$$\underline{f}_u \sim \mathcal{N}(\mu_u^+, \Sigma_{uu}^+), \quad (4.25)$$

$$\begin{aligned} \Sigma_{uu}^+ &= K_{uu} - [K_{um} \quad K_{u+}] \left(\begin{bmatrix} K_{mm} & K_{m+} \\ K_{+m} & K_{++} \end{bmatrix} + \begin{bmatrix} \hat{\Sigma}_{f_m} & 0 \\ 0 & \hat{\sigma}_{f_+}^2 \end{bmatrix} \right)^{-1} \begin{bmatrix} K_{mu} \\ K_{+u} \end{bmatrix}, \\ \mu_u^+ &= \mathbf{m}_u + [K_{um} \quad K_{u+}] \left(\begin{bmatrix} K_{mm} & K_{m+} \\ K_{+m} & K_{++} \end{bmatrix} + \begin{bmatrix} \hat{\Sigma}_{f_m} & 0 \\ 0 & \hat{\sigma}_{f_+}^2 \end{bmatrix} \right)^{-1} \left(\begin{bmatrix} \hat{\mathbf{f}}_m \\ \hat{f}_+ \end{bmatrix} - \begin{bmatrix} \mathbf{m}_m \\ m_+ \end{bmatrix} \right). \end{aligned}$$

We can still rewrite these update laws, to express them in the previous distribution of \underline{f}_u . For this, we should use the old relations

$$\Sigma_{uu} = K_{uu} - K_{um} \hat{K}_{mm}^{-1} K_{mu}, \quad (4.26)$$

$$\mu_u = \mathbf{m}_u + K_{um} \hat{K}_{mm}^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m), \quad (4.27)$$

as well as the matrix inverse (4.24). This would then give us

$$\begin{aligned} \Sigma_{uu}^+ &= \Sigma_{uu} - [K_{um} \quad K_{u+}] \begin{bmatrix} -\hat{K}_{mm}^{-1} K_{m+} \\ 1 \end{bmatrix} \Delta_{++}^{-1} [-K_{+m} \hat{K}_{mm}^{-1} \quad 1] \begin{bmatrix} K_{mu} \\ K_{+u} \end{bmatrix} \\ &= \Sigma_{uu} - (K_{u+} - K_{um} \hat{K}_{mm}^{-1} K_{m+}) \Delta_{++}^{-1} (K_{+u} - K_{+m} \hat{K}_{mm}^{-1} K_{mu}), \end{aligned} \quad (4.28)$$

$$\begin{aligned} \mu_u^+ &= \mu_u + [K_{um} \quad K_{u+}] \begin{bmatrix} -\hat{K}_{mm}^{-1} K_{m+} \\ 1 \end{bmatrix} \Delta_{++}^{-1} [-K_{+m} \hat{K}_{mm}^{-1} \quad 1] \left(\begin{bmatrix} \hat{\mathbf{f}}_m \\ \hat{f}_+ \end{bmatrix} - \begin{bmatrix} \mathbf{m}_m \\ m_+ \end{bmatrix} \right) \\ &= \mu_u + (K_{u+} - K_{um} \hat{K}_{mm}^{-1} K_{m+}) \Delta_{++}^{-1} ((\hat{f}_+ - m_+) - K_{+m} \hat{K}_{mm}^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m)). \end{aligned} \quad (4.29)$$

Here we see that, to update the distribution of \underline{f}_u , we still need to calculate Δ_{++} . In other words, the online sparse GP regression algorithm is just as slow as the regular online GP regression algorithm. Luckily, the online FITC algorithm offers better results.

4.2.3. ONLINE FITC REGRESSION

Earlier (Section 4.1.4) we discussed the intuitive interpretation of the FITC algorithm. It comes down to using each measurement $(\mathbf{x}_{m_i}, \hat{f}_{m_i})$ individually to predict the posterior distribution of \underline{f}_u and then merging all these distributions together in the proper way. This means that, to update the distribution of \underline{f}_u within the FITC algorithm, we can first

predict the distribution of \underline{f}_u using only the new measurement. We write this as

$$\begin{aligned}\underline{f}_u &\sim \mathcal{N}(\mu'_u, \Sigma'_{uu}), \\ \Sigma'_{uu} &= K_{uu} - K_{u+} \hat{K}_{++}^{-1} K_{+u}, \\ \mu'_u &= \mathbf{m}_u + K_{u+} \hat{K}_{++}^{-1} (\hat{f}_+ - m_+).\end{aligned}\quad (4.30)$$

We then merge this together with the previous posterior distribution of \underline{f}_u and unmerge the prior distribution of \underline{f}_u to prevent ourselves from using it twice. This gives us the update law

$$\mathcal{N}(\mu_u^+, \Sigma_{uu}^+) = \mathcal{N}(\mu'_u, \Sigma'_{uu}) \oplus \mathcal{N}(\mu_u, \Sigma_{uu}) \ominus \mathcal{N}(\mathbf{m}_u, K_{uu}). \quad (4.31)$$

Calculating μ_u^+ and Σ_{uu}^+ in this way would work, but it is not the most efficient way to calculate them. To be precise, one update would cost $\mathcal{O}(n_u^3)$ time. We can do better, and we will do so through a somewhat different solution method.

Suppose that we already know \mathbf{x}_+ , but we have not obtained the measurement \hat{f}_+ yet. What can we then say about the distribution of \underline{f}_+ ? In this case, according to (4.9), it is given by

$$\begin{aligned}\begin{bmatrix} \underline{f}_+ \\ \underline{f}_u \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \mu_+ \\ \mu_u \end{bmatrix}, \begin{bmatrix} \Sigma_{++} & \Sigma_{+u} \\ \Sigma_{u+} & \Sigma_{uu} \end{bmatrix}\right), \\ \begin{bmatrix} \Sigma_{++} & \Sigma_{+u} \\ \Sigma_{u+} & \Sigma_{uu} \end{bmatrix} &= \begin{bmatrix} K_{++} - K_{u+} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}) K_{uu}^{-1} K_{u+} & K_{u+} K_{uu}^{-1} \Sigma_{uu} \\ \Sigma_{uu} K_{uu}^{-1} K_{u+} & \Sigma_{uu} \end{bmatrix}, \\ \begin{bmatrix} \mu_+ \\ \mu_u \end{bmatrix} &= \begin{bmatrix} m_+ + K_{u+} K_{uu}^{-1} (\mu_u - \mathbf{m}_u) \\ \mathbf{m}_u \end{bmatrix}.\end{aligned}\quad (4.32)$$

The key insight is that this is the *prior* distribution of \underline{f}_+ and \underline{f}_u , and our measurement \hat{f}_+ is a *measurement* of \underline{f}_+ . And we already know how to incorporate measurements into distributions. We just use the default GP regression equation (2.30). The result (which is also confirmed by Theorem B.36) will be

$$\begin{aligned}\begin{bmatrix} \underline{f}_+ \\ \underline{f}_u \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \mu_+^+ \\ \mu_u^+ \end{bmatrix}, \begin{bmatrix} \Sigma_{++}^+ & \Sigma_{+u}^+ \\ \Sigma_{u+}^+ & \Sigma_{uu}^+ \end{bmatrix}\right), \\ \begin{bmatrix} \Sigma_{++}^+ & \Sigma_{+u}^+ \\ \Sigma_{u+}^+ & \Sigma_{uu}^+ \end{bmatrix} &= \begin{bmatrix} \Sigma_{++} \left(\Sigma_{++} + \hat{\sigma}_{f_+}^2 \right)^{-1} \hat{\sigma}_{f_+}^2 & \hat{\sigma}_{f_+}^2 \left(\Sigma_{++} + \hat{\sigma}_{f_+}^2 \right)^{-1} \Sigma_{+u} \\ \Sigma_{u+} \left(\Sigma_{++} + \hat{\sigma}_{f_+}^2 \right)^{-1} \hat{\sigma}_{f_+}^2 & \Sigma_{uu} - \Sigma_{u+} \left(\Sigma_{++} + \hat{\sigma}_{f_+}^2 \right)^{-1} \Sigma_{+u} \end{bmatrix}, \\ \begin{bmatrix} \mu_+^+ \\ \mu_u^+ \end{bmatrix} &= \begin{bmatrix} \left(\Sigma_{++}^+ \right)^{-1} \left(\Sigma_{++}^{-1} \mu_+ + \hat{\sigma}_{f_+}^{-2} \hat{f}_+ \right) \\ \mathbf{m}_u + \Sigma_{u+} \left(\Sigma_{++} + \hat{\sigma}_{f_+}^2 \right)^{-1} (\hat{f}_+ - \mu_+) \end{bmatrix}.\end{aligned}\quad (4.33)$$

Note the distinction between the old parameters before we obtained our new measurement (without superscript $^+$) and the new parameters, updated using our new measurement (with superscript $^+$).

The most important part of the above distribution is of course the new distribution $\mathcal{N}(\boldsymbol{\mu}_u^+, \Sigma_{uu}^+)$ of the inducing function values \underline{f}_u . We can use the above update law to incorporate single measurements, one by one. We can even do so right from the start, where we start with the prior distribution $\mathcal{N}(\boldsymbol{m}_u, K_{uu})$. And at every point in time, if we want to make predictions \underline{f}_+ , we can just plug the values of $\boldsymbol{\mu}_u$ and Σ_{uu} into (4.9).

So how long does it take to update the distribution $\mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu})$? Assuming that we already know K_{uu}^{-1} (which does not change), we can calculate Σ_{++} in $\mathcal{O}(n_u^2)$ time, and similarly update Σ_{++}^+ in $\mathcal{O}(n_u^2)$ time. A single update hence takes $\mathcal{O}(n_u^2)$ time, while incorporating a set of n_m measurements would take $\mathcal{O}(n_m n_u^2)$ time. Given that n_u is generally a lot smaller than n_m , this is a significant improvement compared to using regular online or sparse online GP regression.

4

There is still another advantage to this algorithm, and it concerns the memory requirements. In the online FITC algorithm, we do not need to keep track of all measurements. In fact, as soon as we have incorporated a measurement into the distribution of \underline{f}_u , we do not need it anymore. We can safely discard it. In the offline FITC algorithm we still needed to set up K_{mu} , but the only thing that we need to remember now is the distribution of \underline{f}_u , irrespective of how many measurements we get. As such, the memory required for the online FITC algorithm reduces from $\mathcal{O}(n_m n_u)$ to $\mathcal{O}(n_u^2)$.

Finally, we can look at the learning progress of the algorithm. One way of doing this is by examining the posterior variance $\Sigma_{u_i u_i}$ of each individual inducing function value \underline{f}_{-u_i} compared to the prior variance $K_{u_i u_i}$. That is, we examine the ratio between variances $\Sigma_{u_i u_i} / K_{u_i u_i}$ or alternatively the ratio between standard deviations $\sqrt{\Sigma_{u_i u_i} / K_{u_i u_i}}$. We call the latter ratio the *learning index* of the inducing function value. When we keep track of the learning indices during the learning process, we find Figure 4.5.

4.2.4. ONLINE PITC REGRESSION

We can set up an online PITC algorithm in the same way. How this online algorithm works depends on what we want to do with it.

The most common way to set up an online PITC algorithm is to add a small subgroup of measurements (X_+, \underline{f}_+) , all together in one update. In this case, the updating process is identical to the updating process of the online FITC algorithm. We first set up the prior distribution of \underline{f}_{-+} and \underline{f}_u , given the data that we have. Identically to (4.32), it equals

$$\begin{aligned} \begin{bmatrix} \underline{f}_{-+} \\ \underline{f}_u \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_+ \\ \boldsymbol{\mu}_u \end{bmatrix}, \begin{bmatrix} \Sigma_{++} & \Sigma_{+u} \\ \Sigma_{u+} & \Sigma_{uu} \end{bmatrix}\right), \\ \begin{bmatrix} \Sigma_{++} & \Sigma_{+u} \\ \Sigma_{u+} & \Sigma_{uu} \end{bmatrix} &= \begin{bmatrix} K_{++} - K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}) K_{uu}^{-1} K_{u+} & K_{+u} K_{uu}^{-1} \Sigma_{uu} \\ \Sigma_{uu} K_{uu}^{-1} K_{u+} & \Sigma_{uu} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_+ \\ \boldsymbol{\mu}_u \end{bmatrix} &= \begin{bmatrix} \boldsymbol{m}_+ + K_{+u} K_{uu}^{-1} (\boldsymbol{\mu}_u - \boldsymbol{m}_u) \\ \boldsymbol{\mu}_u \end{bmatrix}. \end{aligned} \quad (4.34)$$

Then we incorporate the measurement $\hat{\underline{f}}_+$ with corresponding measurement noise matrix $\hat{\Sigma}_{f_+}$ for this new subgroup of input points. This turns the update law, identically

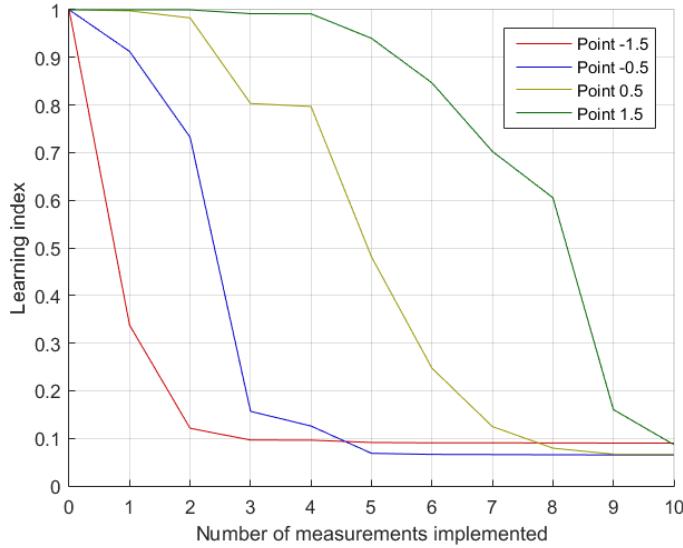


Figure 4.5: The learning indices $\sqrt{\Sigma_{u_i u_i} / K_{u_i u_i}}$ during the execution of the online FITC algorithm, for each of the inducing function values. The data used is the same as the data from Figure 4.1, and measurement points were used in ascending order, based on the input x_m . A learning index of 1 means ‘no data’ while an index of 0 means ‘infinite certainty’. You can see that, when a measurement is performed near an inducing input point, the learning index will jump downward.

to (4.33), into

$$\begin{aligned} \begin{bmatrix} \underline{f}_+ \\ \underline{f}_u \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_+^+ \\ \boldsymbol{\mu}_u^+ \end{bmatrix}, \begin{bmatrix} \Sigma_{++}^+ & \Sigma_{+u}^+ \\ \Sigma_{u+}^+ & \Sigma_{uu}^+ \end{bmatrix}\right), \quad (4.35) \\ \begin{bmatrix} \Sigma_{++}^+ & \Sigma_{+u}^+ \\ \Sigma_{u+}^+ & \Sigma_{uu}^+ \end{bmatrix} &= \begin{bmatrix} \Sigma_{++} (\Sigma_{++} + \hat{\Sigma}_{f_+})^{-1} \hat{\Sigma}_{f_+} & \hat{\Sigma}_{f_+} (\Sigma_{++} + \hat{\Sigma}_{f_+})^{-1} \Sigma_{+u} \\ \Sigma_{u+} (\Sigma_{++} + \hat{\Sigma}_{f_+})^{-1} \hat{\Sigma}_{f_+} & \Sigma_{uu} - \Sigma_{u+} (\Sigma_{++} + \hat{\Sigma}_{f_+})^{-1} \Sigma_{+u} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_+^+ \\ \boldsymbol{\mu}_u^+ \end{bmatrix} &= \begin{bmatrix} (\Sigma_{++}^+)^{-1} (\Sigma_{++}^{-1} \boldsymbol{\mu}_+ + \hat{\Sigma}_{f_+}^{-1} \hat{\boldsymbol{f}}_+) \\ \boldsymbol{\mu}_u + \Sigma_{u+} (\Sigma_{++} + \hat{\Sigma}_{f_+})^{-1} (\hat{\boldsymbol{f}}_+ - \boldsymbol{\mu}_+) \end{bmatrix}. \end{aligned}$$

This is the usual update law for the PITC algorithm.

However, instead of adding a completely new subgroup, we could also add a single measurement $(\underline{x}_+, \hat{\boldsymbol{f}}_+)$ to an already existing subgroup X_{m_i} of measurement points. (Or do so with multiple new measurements at the same time; the process is identical.) Because this new point \underline{x}_+ is linked to the points within X_{m_i} , we cannot just ignore the measurements done on \underline{f}_{m_i} . To be precise, these measurements $\hat{\boldsymbol{f}}_{m_i}$ affect the prior distribution of \underline{f}_+ in ways which we could not derive from the distribution of \underline{f}_u . As such, we need to adjust (4.34).

We will only discuss the outcome of this process. (For some of the mathematics, see Theorem B.37.) But before we can do that, we first have to make a slight change in the

way we write our expressions. We will use the notation

$$\Lambda_{ab} \equiv K_{ab} - K_{au} K_{uu}^{-1} K_{ub}, \quad (4.36)$$

for any sensible subscripts a and b . In this case, the expression for Σ_{++} from (4.34) can also be written as $\Lambda_{++} + K_{+u} K_{uu}^{-1} \Sigma_{uu} K_{uu}^{-1} K_{u+}$. Keeping this in mind, we will now define

$$\tilde{\Lambda}_{++} \equiv \Lambda_{++} - \Lambda_{+m_i} (\Lambda_{m_i m_i} + \hat{\Sigma}_{m_i m_i})^{-1} \Lambda_{m_i +}, \quad (4.37)$$

$$\tilde{K}_{u+} \equiv K_{u+} - K_{um_i} (\Lambda_{m_i m_i} + \hat{\Sigma}_{m_i m_i})^{-1} \Lambda_{m_i +}, \quad (4.38)$$

$$\tilde{K}_{+u} \equiv K_{+u} - \Lambda_{+m_i} (\Lambda_{m_i m_i} + \hat{\Sigma}_{m_i m_i})^{-1} K_{m_i u}, \quad (4.39)$$

$$\tilde{m}_+ \equiv m_+ + \Lambda_{+m_i} (\Lambda_{m_i m_i} + \hat{\Sigma}_{m_i m_i})^{-1} (\hat{f}_{m_i} - \mathbf{m}_{m_i}). \quad (4.40)$$

4

Given these new definitions, we can set up the new prior distribution of \underline{f}_+ together with \underline{f}_u . It follows as

$$\begin{aligned} \begin{bmatrix} \underline{f}_+ \\ \underline{f}_u \end{bmatrix} &\sim \mathcal{N} \left(\begin{bmatrix} \mu_+ \\ \boldsymbol{\mu}_u \end{bmatrix}, \begin{bmatrix} \Sigma_{++} & \Sigma_{+u} \\ \Sigma_{u+} & \Sigma_{uu} \end{bmatrix} \right), \\ \begin{bmatrix} \Sigma_{++} & \Sigma_{+u} \\ \Sigma_{u+} & \Sigma_{uu} \end{bmatrix} &= \begin{bmatrix} \tilde{\Lambda}_{++} + \tilde{K}_{+u} K_{uu}^{-1} \Sigma_{uu} K_{uu}^{-1} \tilde{K}_{u+} & \tilde{K}_{+u} K_{uu}^{-1} \Sigma_{uu} \\ \Sigma_{uu} K_{uu}^{-1} \tilde{K}_{u+} & \Sigma_{uu} \end{bmatrix}, \\ \begin{bmatrix} \mu_+ \\ \boldsymbol{\mu}_u \end{bmatrix} &= \begin{bmatrix} \tilde{m}_+ + \tilde{K}_{+u} K_{uu}^{-1} (\boldsymbol{\mu}_u - \mathbf{m}_u) \\ \boldsymbol{\mu}_u \end{bmatrix}. \end{aligned} \quad (4.41)$$

After calculating this distribution, we can incorporate the measurement performed on \underline{f}_+ in the usual way through (4.33). This then allows us to update the distribution of \underline{f}_u , which was our main goal.

With the two PITC updating methods that we just examined, we can hence both add new subgroups to the PITC algorithm and make existing subgroups larger. Both update methods can be valuable in the right situations. In fact, when using the latter update method, we should be careful not to let the subgroups grow to big, which would slow our algorithm down. But if a subgroup does happen to become too large, we can of course use the first update method to set up a new subgroup and continue from there.

4.2.5. NUMERICAL STABILITY OF THE ONLINE METHODS

The online methods we have developed make Gaussian process regression much more powerful. It is always possible to efficiently incorporate new data into the regression method.

There is only one significant downside to the online methods. Every time we update Σ_{uu} , incorporating a new measurement, small numerical errors seep into Σ_{uu} . Though in theory this matrix always remains positive definite, numerical problems may occur after a large number of measurements, which may cause it to be singular. This could potentially make future predictions invalid.

I personally know of two reasonable ways to solve this issue. The first one is to simply apply the offline FITC equation (4.13) after all. Doing this just once is numerically more robust than applying n_m updates to Σ_{uu} .

Of course applying offline FITC regression is not always possible. Another way to prevent Σ_{uu} from becoming singular is to regularly add a small amount ϵ to the diagonal of Σ_{uu} . This effectively adds extra ‘noise’ in the distribution of \underline{f}_u , but if ϵ is small enough, this should not significantly affect the predictions made by the algorithm, while it does prevent numerical problems.

Other than these two methods, there might be different ways of keeping track of the distribution of \underline{f}_u that are numerically more stable. For instance, instead of keeping track of Σ_{uu} we could also set up online updates of the quantity $K_{uu} - \Sigma_{uu}$ (or some other transformation of Σ_{uu}) and use that to make predictions. Another example of this is discussed in Section 4.3.4. I will leave the whole analysis of the numerical stability of these methods as a subject for future research though.

4.3. CHOOSING THE INDUCING INPUT POINTS

4

So far we have assumed that the inducing input set X_u was known. Just like any hyper-parameter, we can either choose it based on knowledge of the function we are approximating, or tune it automatically. And both can be done offline or online. Let’s take a look at how that works.

We will start with the offline case. First we look at how to manually choose inducing input points (Section 4.3.1) and then we examine how we can automatically tune them (Section 4.3.2). We follow up with the online case (Section 4.3.3). At the end we also look at an alternative way of keeping track of the distribution of the inducing function values (Section 4.3.4).

4.3.1. MANUALLY CHOOSING THE INDUCING INPUT POINTS OFFLINE

Suppose that we have a set of measurement input points X_m with corresponding measurements $\hat{\mathbf{f}}_m$, as well as a set of trial input points X_* for which we want to predict the trial function values \underline{f}_* . What would be good locations to place the inducing input points?

The first thing we should realize here is that in the training step inducing input points ‘absorb’ information. That is, we use the measurements to predict the inducing function values \underline{f}_u . If our measurements do not give any information about a specific inducing function value f_{u_i} , then the corresponding inducing input point x_{u_i} is useless and might as well be removed. As such, we should put our inducing input points where our measurements provide data.

In the prediction step things are reversed: now the inducing input points ‘provide’ information to make predictions. If we make a prediction \underline{f}_* for a trial point x_* far away from any inducing input point, then the prediction will likely be very inaccurate. As such, we should put our inducing input points where we want to make predictions.

Finally we should note that having two inducing input points on exactly the same place is pointless. (Or worse: detrimental because it makes K_{uu} singular.) Using only one of the two points will give exactly the same results as both points together. As such, there should be a reasonable spread between inducing input points.

Based on this, we can come up with a basic rule for choosing inducing input points. *Make sure to pick inducing input points that are sufficiently spread out in areas where*

you have both measurement data and where you want to make predictions. The term ‘sufficiently spread out’ here can be read as ‘having roughly a distance λ_x between them,’ though this depends on the trade-off between the desired accuracy you want to get and the computational complexity you can deal with.

There is also a slightly different way to look at this problem. In the ultimate situation, after we have incorporated infinitely many measurements, we will have $\Sigma_{uu} = 0$. This means that we know the inducing function values $\underline{f}_{\mathbf{u}}$ deterministically. Still, this is the only data that we have of making predictions \underline{f}_* . So when choosing inducing input points, we could also ask ourselves, ‘Which function values do we want to know with infinite precision, such that we can estimate the entire function?’

4.3.2. AUTOMATICALLY TUNING THE INDUCING INPUT POINTS OFFLINE

4

Next to choosing inducing points manually, we can also automatically tune them. How to do so depends on the exact assumptions that we make and on what data we have.

First, let’s consider the FITC algorithm from Section 4.1.4. In this algorithm, our assumptions cause K_{mm} to change. To be precise, it becomes

$$K_{mm} \leftarrow \text{diag}(K_{mm} - K_{mu}K_{uu}^{-1}K_{um}) + K_{mu}K_{uu}^{-1}K_{um} = \Lambda_{mm} + K_{mu}K_{uu}^{-1}K_{um}, \quad (4.42)$$

with Λ_{mm} still defined according to (4.11). This means that K_{mm} now depends on K_{uu} .

An idea here is to use the theory from Section 3.1 and treat X_u as just another hyper-parameter³. That is, we maximize the likelihood $p(\hat{\mathbf{f}}_m | X_m, X_u)$ with respect to X_u . This likelihood equals

$$p(\hat{\mathbf{f}}_m | X_m, X_u) = \mathcal{N}\left(\hat{\mathbf{f}}_m | \mathbf{m}_m, K_{mm} + \hat{\Sigma}_{f_m}\right). \quad (4.43)$$

Intuitively, this method comes down to picking the inducing input points X_u that can best explain the measured data. The method is hence also known as *evidence maximization*.

In practice, optimizing the above likelihood often causes numerical problems, so instead we maximize the log-likelihood. Identically to (3.9), this equals

$$-\frac{n_m}{2} \log(2\pi) - \frac{1}{2} \log|K_{mm} + \hat{\Sigma}_{f_m}| - \frac{1}{2} (\hat{\mathbf{f}}_m - \mathbf{m}_m)^T (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m). \quad (4.44)$$

To obtain some extra computational efficiency, we can also apply

$$|\Lambda_{mm} + \hat{\Sigma}_{f_m} + K_{mu}K_{uu}^{-1}K_{um}| = \frac{|K_{uu} + K_{um}(\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1}K_{mu}| |\Lambda_{mm} + \hat{\Sigma}_{f_m}|}{|K_{uu}|}, \quad (4.45)$$

$$(\Lambda_{mm} + \hat{\Sigma}_{f_m} + K_{mu}K_{uu}^{-1}K_{um})^{-1} = (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} - (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \quad (4.46)$$

$$\left(K_{uu} + K_{um}(\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1}K_{mu}\right)^{-1} K_{um} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1}.$$

The first follows from Theorem A.9 and the second from Theorem A.7. When we now maximize (4.44) with respect to K_{uu} , we get the results shown in Figure 4.6 (left).

³This approach was first introduced by Snelson and Ghahramani (2006a), although it seems they were not aware of the assumption they were silently making and how it affected the covariance matrix K_{mm} , resulting in a more complicated derivation.

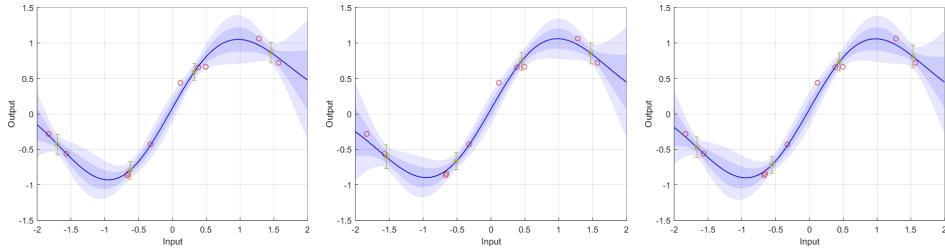


Figure 4.6: Predictions after tuning of the inducing input points. For the left figure, we did not use any data of the trial input set X_* but directly maximized (4.43). For the middle figure, we used a given trial set $X_* = [-2, -\frac{3}{2}, -1, -\frac{1}{2}, 0, \frac{1}{2}, 1, \frac{3}{2}, 2]$ and minimized the determinant of (4.49). For the right figure, we used a distribution $\mathbf{x}_* \sim \mathcal{N}(0, 1)$ and minimized (4.53).

4

Next, let's consider the sparse GP regression algorithm from Section 4.1.3. In this algorithm, our assumptions only cause K_{m*} to change into $K_{mu}K_{uu}^{-1}K_{u*}$, but K_{mm} is unaffected. Because of this, K_{mm} does *not* depend on K_{uu} , and the above method hence does not work. We need something else.

A first idea is to pick our inducing input points such that the inducing function values \underline{f}_u absorb a lot of data and hence have a small variance. The variances of the inducing input points $\underline{f}_{u_1}, \dots, \underline{f}_{u_{n_u}}$ are the diagonal elements of

$$\Sigma_{uu} = K_{uu} - K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu}. \quad (4.47)$$

As such, we can minimize the product of the diagonal elements of the above matrix. However, this idea will not work. If we would do this, we find an optimum with all inducing input points $\mathbf{x}_{u_1}, \dots, \mathbf{x}_{u_{n_u}}$ being equal to the point \mathbf{x}_u minimizing

$$k(\mathbf{x}_u, \mathbf{x}_u) - k(\mathbf{x}_u, X_m) (K_{mm} + \hat{\Sigma}_{f_m})^{-1} k(X_m, \mathbf{x}_u). \quad (4.48)$$

This does not correspond well to our idea of ‘spreading’ the inducing input points.

A solution arises when we already have a trial input set X_* given. In this case, we want to maximize the amount of information contained in the posterior distribution of \underline{f}_* . (Using knowledge of the trial set like this is known as *transduction learning*.) This ‘amount of information’ depends on the covariance matrix

$$\Sigma_{**} = K_{**} - K_{*u} K_{uu}^{-1} K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} K_{uu}^{-1} K_{u*}. \quad (4.49)$$

To be precise, we want to minimize the determinant $|\Sigma_{**}|$, although in practice it is numerically more stable to optimize the logarithm $\log |\Sigma_{**}|$ of this determinant. (This is in turn equivalent to optimizing the entropy of \underline{f}_* . For a brief introduction into entropy, see Section 6.4.3.) When we do, we can get results as those shown in Figure 4.6 (middle).

When no trial input set X_* is given, we need to do something else. In this case we can assume that $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_*}, \boldsymbol{\Sigma}_{\mathbf{x}_*})$ for some properly chosen $\boldsymbol{\mu}_{\mathbf{x}_*}$ and $\boldsymbol{\Sigma}_{\mathbf{x}_*}$, and subsequently minimize the *expected posterior variance*

$$\mathbb{E} [\Sigma_{**}] = \int_X \Sigma_{**} \mathcal{N}(\mathbf{x}_* | \boldsymbol{\mu}_{\mathbf{x}_*}, \boldsymbol{\Sigma}_{\mathbf{x}_*}) d\mathbf{x}_*. \quad (4.50)$$

Depending on the covariance function $k(\mathbf{x}, \mathbf{x}')$ that we use, the above integral may or may not be analytically solvable. When we use the squared exponential covariance function (2.35), the integral can be solved analytically. In this case, let's define the shorthand

$$P \equiv K_{uu}^{-1} K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} K_{uu}^{-1}. \quad (4.51)$$

Using this definition, we can expand the integral (4.50) into

$$\begin{aligned} \mathbb{E}[\Sigma_{**}] &= \int_X (K_{**} - K_{*u} P K_{u*}) \mathcal{N}(\mathbf{x}_* | \boldsymbol{\mu}_{\mathbf{x}_*}, \Sigma_{\mathbf{x}_*}) d\mathbf{x}_* \\ &= \int_X k(\mathbf{x}_*, \mathbf{x}_*) \mathcal{N}(\mathbf{x}_* | \boldsymbol{\mu}_{\mathbf{x}_*}, \Sigma_{\mathbf{x}_*}) d\mathbf{x}_* \\ &\quad - \int_X \sum_{i=1}^{n_u} \sum_{j=1}^{n_u} k(\mathbf{x}_*, \mathbf{x}_{u_i}) P_{ij} k(\mathbf{x}_{u_j}, \mathbf{x}_*) \mathcal{N}(\mathbf{x}_* | \boldsymbol{\mu}_{\mathbf{x}_*}, \Sigma_{\mathbf{x}_*}) d\mathbf{x}_*. \end{aligned} \quad (4.52)$$

4

The first integral has $k(\mathbf{x}_*, \mathbf{x}_*) = \lambda_f^2$ as its solution. To solve the second integral, we will need Theorem A.19. Through it, we can derive the final result

$$\begin{aligned} \mathbb{E}[\Sigma_{**}] &= \lambda_f^2 - \lambda_f^4 \sqrt{\frac{|\Lambda_x|}{|\Lambda_x + 2\Sigma_{\mathbf{x}_*}|}} \sum_{i=1}^{n_u} \sum_{j=1}^{n_u} P_{ij} \exp\left(-\frac{1}{2} (\mathbf{x}_{u_i} - \mathbf{x}_{u_j})^T (2\Lambda_x)^{-1} (\mathbf{x}_{u_i} - \mathbf{x}_{u_j})\right) \\ &\quad \exp\left(-\frac{1}{2} \left(\frac{\mathbf{x}_{u_i} + \mathbf{x}_{u_j}}{2} - \boldsymbol{\mu}_*\right)^T \left(\frac{1}{2} \Lambda_x + \Sigma_{\mathbf{x}_*}\right)^{-1} \left(\frac{\mathbf{x}_{u_i} + \mathbf{x}_{u_j}}{2} - \boldsymbol{\mu}_*\right)\right). \end{aligned} \quad (4.53)$$

This quantity can then be minimized with respect to the inducing input points $\mathbf{x}_{u_1}, \dots, \mathbf{x}_{u_n}$. Doing so would result in Figure 4.6 (right).

The two extra tuning methods we have just derived work for the sparse GP regression algorithm. They can also be applied to the FITC algorithm. In this case, we do need to adjust K_{mm} according to (4.42), but then everything works the same. However, we can obtain a computationally more efficient version of the algorithm if, instead of (4.49), we use the rewritten version

$$\Sigma_{**} = K_{**} - K_{*u} \left(K_{uu}^{-1} - \left(K_{uu} + K_{um} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \right)^{-1} \right) K_{u*}, \quad (4.54)$$

where we should also redefine P as

$$P = K_{uu}^{-1} - \left(K_{uu} + K_{um} (\Lambda_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \right)^{-1}. \quad (4.55)$$

For large data sets, these expressions are much more efficient to calculate, although the method itself essentially remains the same.

So out of the three tuning methods shown in Figure 4.6, which should we use? After all, they all seem to work. The answer mainly depends on whether you know the trial function values X_* and whether you want to take that knowledge into account when choosing the inducing input points. If you know X_* exactly, use the second method. If you do not know X_* at all, use the first. And if you only have a rough idea of X_* , use the third method.

The hardest part of each of these methods is the optimization problem. This is difficult because there are lots of local minima. After all, the order of the inducing input points is irrelevant, so even if we have found the optimal inducing input points X_u , then interchanging two inducing input points will result in another optimum. As such, after tuning the inducing input points, it is always worthwhile to do a quick manual check of the results to see if they are sensible, or if we have converged to some senseless local optimum.

4.3.3. ADJUSTING THE INDUCING INPUT POINTS ONLINE

Next, we examine the online case. Suppose that we are continuously obtaining new measurements and updating the distribution of the inducing function values \underline{f}_u . Can we then also adjust the inducing input points X_u themselves?

One option here is to keep track of all measurements X_m as we go, and use them all together to tune the inducing input points. This results in the offline tuning methods that we just discussed though.

Alternatively, we could try to use only the last measurement $(\mathbf{x}_{m_i}, \hat{f}_{m_i})$ we have obtained to tune the inducing input points. However, using a single measurement to tune a multitude of parameters is very unlikely to give good results.

That is why I prefer to use a very simple update rule. ‘If the new measurement point \mathbf{x}_{m_i} is not close to any already existing inducing input point \mathbf{x}_{u_j} , then we will add \mathbf{x}_{m_i} as new inducing input point.’ Here, ‘close’ can mean that we have

$$(\mathbf{x}_{m_i} - \mathbf{x}_{u_j})^T \Lambda_x^{-1} (\mathbf{x}_{m_i} - \mathbf{x}_{u_j}) < c, \quad (4.56)$$

where c is some manually chosen threshold. I personally prefer to start with $c = 1$ and decrease c slowly as the algorithm progresses, though this does depend on the exact application.

The given update rule does pose the question ‘How can we add an inducing input point?’ Or more general, ‘How can we add a new set of inducing input points X_v to the current set X_u ?’ The answer is surprisingly easy. Given all the data that we have, the new inducing function value vector will, identically to (4.9), become

$$\begin{aligned} \begin{bmatrix} \underline{f}_u \\ \underline{f}_v \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_u \\ \boldsymbol{\mu}_v \end{bmatrix}, \begin{bmatrix} \Sigma_{uu} & \Sigma_{uv} \\ \Sigma_{vu} & \Sigma_{vv} \end{bmatrix}\right) \\ \begin{bmatrix} \Sigma_{uu} & \Sigma_{uv} \\ \Sigma_{vu} & \Sigma_{vv} \end{bmatrix} &= \begin{bmatrix} \Sigma_{uu} & \Sigma_{uu} K_{uu}^{-1} K_{uv} \\ K_{vu} K_{uu}^{-1} \Sigma_{uu} & K_{vv} - K_{vu} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}) K_{uu}^{-1} K_{uv} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_u \\ \boldsymbol{\mu}_v \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu}_u \\ \mathbf{m}_v + K_{vu} K_{uu}^{-1} (\boldsymbol{\mu}_u - \mathbf{m}_u) \end{bmatrix}. \end{aligned} \quad (4.57)$$

We can then directly continue processing new measurements using this new inducing function value vector.

If we use this new distribution of the inducing function values to make predictions, then initially nothing changes. We still get exactly the same predictions. (Although the process itself will be a bit slower.) However, we can use this extended set of inducing

input points when incorporating new data. This basically causes more data to be incorporated, causing later predictions to be more accurate than they earlier would have been.

Next to adding inducing input points, it is also possible to remove inducing input points. To do so, simply remove the point from X_u and the corresponding element from \underline{f}_u . Naturally, when doing so, there will be some data loss but, depending on the application, this may be acceptable.

Since we now know how to add and remove inducing input points, we can take things one step further. It is also possible to shift inducing input points by a small (or a large) distance. To do so, just add the new versions of the inducing input points and then immediately remove the old versions. (First removing the old points and then adding the new ones will not be wise, since it will result in more data loss.) To see this process at work, take a look at Figure 4.7.

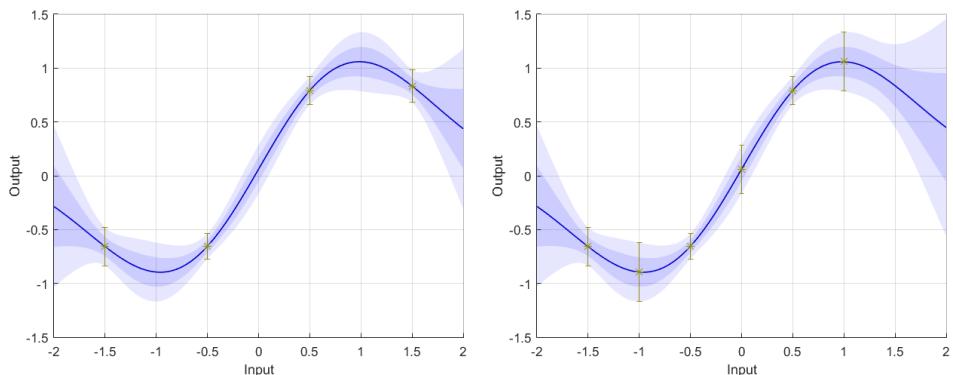


Figure 4.7: The sparse GP regression predictions before and after changing the inducing input set. The left figure equals Figure 4.1 (right). To get the right figure, we added three inducing input points and removed one, although you could also say that we ‘shifted’ the inducing input point from 1.5 to 1. Note that adding inducing input points does not directly improve the accuracy, while removing inducing input points does reduce the accuracy. However, when new measurements will be incorporated in the future (not done here) these new inducing input points will improve the accuracy of future predictions.

When using these techniques in an online way, the most commonly used technique is to add inducing input points as we go, increasing the accuracy where we get most measurement data. However, it is also possible to slightly adjust the set of inducing input points upon receiving each new measurements. When shifting an inducing input point by only a tiny distance, hardly any data is lost, but over time this may significantly improve the distribution of the inducing input points. How to effectively shift around the inducing input points during such an online learning algorithm to improve the prediction accuracy is still an open question though.

4.3.4. A DIFFERENT MERGING ORDER

In the sparse GP algorithm, we have a training step to predict the distribution of \underline{f}_u and a prediction step to find \underline{f}_* . Crucial in this set-up is that we unmerge the prior distribution $\mathcal{N}(\mathbf{m}_u, K_{uu})$ to prevent using it twice. This is currently done in the prediction step. But

you may be wondering, ‘Is it also possible to do this in the training step instead?’

The answer is ‘Yes, but it is not very convenient.’ I will explain why. If we do this, then after the training step we do not wind up with $\mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu})$, but instead get the distribution

$$\begin{aligned}\mathcal{N}(\boldsymbol{\gamma}, \Gamma) &= \mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu}) \ominus \mathcal{N}(\mathbf{m}_u, K_{uu}), \\ \Gamma &= (\Sigma_{uu}^{-1} - K_{uu}^{-1})^{-1} \\ \boldsymbol{\gamma} &= \Gamma (\Sigma_{uu}^{-1} \boldsymbol{\mu}_u - K_{uu}^{-1} \mathbf{m}_u).\end{aligned}\quad (4.58)$$

The intuitive meaning of $\mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu})$ was obvious. This new distribution $\mathcal{N}(\boldsymbol{\gamma}, \Gamma)$ also has an intuitive meaning. It is the *pseudo measurement distribution* at the inducing input points. That is, if we would do measurements at our inducing points (causing $X_m = X_u$) and get a measured vector $\hat{\mathbf{f}}_m = \boldsymbol{\gamma}$ with noise covariance $\hat{\Sigma}_{f_m} = \Gamma$, then we would get exactly the same result as we would get from all our current measurements.

It is interesting here to note the values of both Σ_{uu} and Γ when no measurements are present. In this case Σ_{uu} equals the prior covariance K_{uu} , while Γ equals the infinite covariance matrix ∞ . It effectively tells us we do not have a clue (apart from the prior distribution) what the inducing function values are.

We can also find training and prediction equations for this alternative notation. The training relation for the sparse GP algorithm of Section 4.1.3 (what used to be (4.5)) becomes

$$\Gamma = K_{uu} \left(K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \right)^{-1} K_{uu} - K_{uu}, \quad (4.59)$$

$$\boldsymbol{\gamma} = \mathbf{m}_u + K_{uu} \left(K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu} \right)^{-1} K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m), \quad (4.60)$$

while the prediction equation (what used to be (4.9)) turns into

$$\Sigma_{**} = K_{**} - K_{*u} (K_{uu} + \Gamma)^{-1} K_{u*}, \quad (4.61)$$

$$\boldsymbol{\mu}_* = \mathbf{m}_* + K_{*u} (K_{uu} + \Gamma)^{-1} (\boldsymbol{\gamma} - \mathbf{m}_u). \quad (4.62)$$

Here we see two problems with this new scheme though. First of all, when we only have few measurements ($n_m < n_u$) then the matrix $(K_{um} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} K_{mu})$ will be singular. As a result, the above training equation fails.

Although this can be worked around, we also have another issue. When we do find Γ , then we still need to invert it to make predictions, while we did not need to do so when we were working with Σ_{uu} . As such, the prediction phase in this new set-up costs more time than in our previous set-up as well.

Both these problems cause this set-up to be much less useful than the regular set-up that we considered in the rest of this chapter, which is why we stick with that set-up.

4.4. APPLICATIONS OF SPARSE ONLINE GP REGRESSION

It is time to apply the algorithms we have developed in this chapter. We will do two experiments. In the first one we apply the algorithms to a simple two-dimensional trial function and compare their performance (Section 4.4.1). In the second experiment we

bring in the pitch-plunge problem from Section 2.6 and apply the various algorithms to this more practical five-dimensional test case (Section 4.4.2).

4.4.1. A COMPARISON BETWEEN ALGORITHMS

As an experiment, we will approximate a trial function using the various algorithms. The function that we will use is the semi-randomly chosen function

$$f(x_1, x_2) = \left(\frac{x_1}{4}\right)^2 + \left(\frac{x_2}{2}\right)^2 - 1 - \sin\left(2\pi\frac{x_1}{4}\right)\left(1 - \frac{1}{3}\cos\left(2\pi\frac{x_2}{6}\right)\right) + \sin\left(2\pi\frac{x_2}{4}\right), \quad (4.63)$$

which is also displayed in Figure 4.8. We evaluate it on the interval $x_1, x_2 \in [-2, 2]$.

The above trial function displays nonlinearities that can be approximated by a squared exponential covariance function with $\lambda_f = \lambda_{x_1} = \lambda_{x_2} = 1$. Such an approximation, subject to a relatively strong measurement noise with standard deviation $\hat{\sigma}_{f_m} = 0.5$, is shown in Figure 4.9.

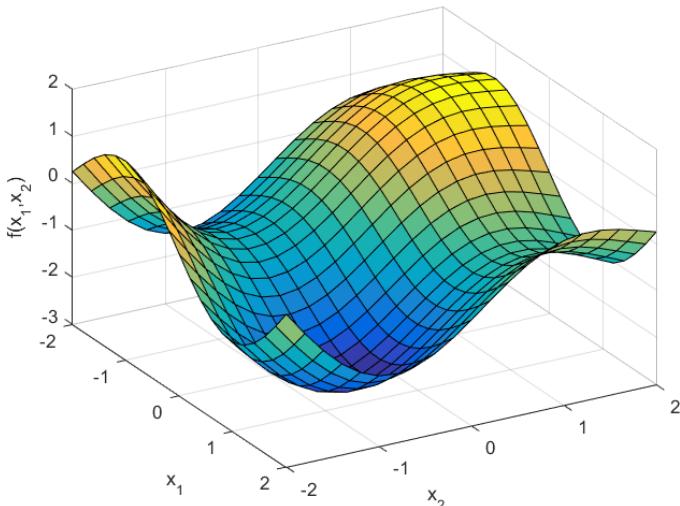


Figure 4.8: The trial function (4.63) that we will approximate in the experiment.

We want to make a comparison between the various sparse algorithms we have seen so far. Doing so in a fair way is tricky, because FITC extracts less data out of measurements compared to the regular GP regression algorithm, but it does so much more quickly. At the same time, PITC is somewhere in-between these two extremes. Because of this, we will examine six cases.

1. Regular GP regression with 10000 measurements.
2. The PITC algorithm with the same number of measurements (10000) as 1.
3. The PITC algorithm with the same runtime as 1.
4. The FITC algorithm with the same number of measurements (10000) as 1.
5. The FITC algorithm with the same number of measurements as 3.

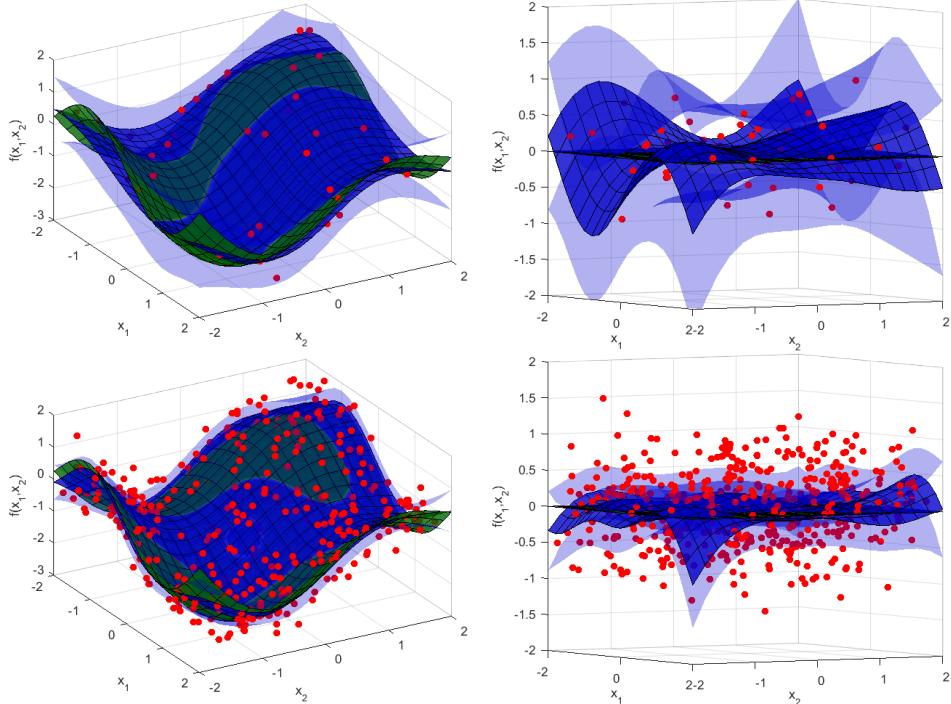


Figure 4.9: Approximation of the trial function (4.63) through the regular GP regression algorithm. First $n_m = 50$ measurements (top) were used and then $n_m = 500$ (bottom). The plots show the approximated function compared to the exact function (left), as well as the error, being the approximated function minus the exact function (right). Also shown are the measurements (left) and the measurement noise (right). From the error plot, we can calculate the root mean squared error of the mean. When $n_m = 50$ this becomes RMSE = 0.32. When $n_m = 500$ this is RMSE = 0.14.

6. The FITC algorithm with the same runtime as 1 and 3.

It is usually hard to predict the runtime of an algorithm in advance. However, with the online algorithms of Section 4.2, we can just keep adding measurements until we are out of time, solving that problem. We will also compare the online algorithms with the offline algorithms of Section 4.1, to see if one is faster than the other.

To compare the accuracy of the algorithms, we will calculate the root mean square error over the input space. A lower RMSE is of course better, and Figure 4.9 has already shown that more measurements generally results in a lower RMSE. When we set up the experiments, we get the results shown in Table 4.2.

Table 4.2: The performance of the six test cases described in the main text. For the PITC algorithm, measurements were added in groups of n_u measurements. The experiments were run on a regular home computer. Note that there are some deviations in runtimes due to other processes running on the computer. For instance, the two runtimes given for case 1 should in theory be identical, but are off by about 3%.

Case	$n_u = 81$ inducing input points				$n_u = 25$ inducing input points			
	n_m	t (offline)	t (online)	RMSE	n_m	t (offline)	t (online)	RMSE
1 (GP)	10k	9.20s	N/A	0.037	10k	9.51s	N/A	0.037
2 (PITC)	10k	0.26s	0.50s	0.037	10k	0.23s	0.46s	0.065
3 (PITC)	642k	7.88s	9.20s	0.0073	970k	6.69s	9.51s	0.059
4 (FITC)	10k	0.20s	0.80s	0.037	10k	0.09s	0.52s	0.066
5 (FITC)	642k	3.62s	49.8s	0.0073	970k	1.73s	48.5s	0.060
6 (FITC)	104k	0.74s	9.20s	0.013	159k	0.35s	9.51s	0.061

There are a couple of things we can notice from Table 4.2. Let's make a list of the most important conclusions.

- **The effects of the number of inducing inputs n_u**

In general, using less inducing input points results in a faster algorithm. However, if we pick too few inducing input points, then the performance of the algorithm decreases significantly. Having a fixed grid of 9×9 inducing input points seems to be sufficient for this problem. In this case, given the same number of measurements, the accuracy of the FITC and PITC algorithms is identical to that of the regular GP algorithm. However, when using a fixed grid of 5×5 inducing input points, these two algorithms will never be able to properly approximate the trial function. You could say that the approximating power of these 25 basis functions is not sufficient to approximate the trial function.

- **The effects of the number of measurements n_m**

The more measurements we use, the more accurate our predictions become. This holds everywhere, seemingly without exception. But at the same time, the more measurements we use, the more time we need to process them. And as predicted, the runtime of the FITC and PITC algorithms is roughly linear in the number of measurements.

- **The differences between offline and online algorithms**

The offline and online algorithms have exactly the same accuracy, as could be ex-

pected. However, the online algorithms are slower than the offline algorithms. The reason behind this is extra overhead. For instance, for online algorithms we have to calculate K_{u+} separately for n_m different points, while for offline algorithms we have to calculate K_{um} only once. The process is the same, but the latter is done faster.

- **The effects of the choice of algorithm**

Given the same number of measurements, the GP regression algorithm always seems to be more accurate than the FITC and PITC algorithms. However, for this two-dimensional problem, if enough inducing input points are used, then this difference is generally very small. Given the same amount of runtime, however, things are very different. Now FITC and PITC significantly outperform the regular GP regression algorithm for the simple reason that they can incorporate more measurements.

When comparing the runtimes of FITC and PITC, something interesting can be seen. Apparently offline FITC is faster than offline PITC, but online FITC is *slower* than online PITC. You would expect FITC to always be faster than PITC. What is going on here?

This can be explained by the implementation of the algorithms in Matlab. In general, Matlab loops are very slow. The online implementations of FITC and PITC both require such loops. However, PITC adds measurements in groups while FITC adds them one by one. Hence, the FITC algorithm requires many more iterations, making it slower than the PITC algorithm. If we would add measurements in groups in the online FITC algorithm too, it would be faster than the online PITC algorithm as well.

In general, the FITC and PITC algorithms perform very similarly. The main conclusion for both of these sparse algorithms is that they are a lot faster than the regular GP regression algorithm. Although if this extra speed is used to incorporate more measurement data, and if a sufficient number of inducing input points is used, then these algorithms will be much more accurate instead.

4.4.2. APPLICATION TO THE PITCH-PLUNGE SYSTEM

Next, we will revisit the pitch-plunge system from Section 2.6 and once more apply regression to identify the system.

Just like previously when making Figure 2.15, we will put the system in a random initial state $[h_k, \alpha_k, \dot{h}_k, \dot{\alpha}_k]$, apply a random control input β_k and see where the system winds up after a time step $\Delta t = 0.1$ s. This constitutes one ‘measurement’. After making n_m such measurements, we can approximate the next height h_{k+1} and pitch angle α_{k+1} .

We will make this approximation with regular GP regression, with the PITC algorithm and with the FITC algorithm. For both the PITC and FITC algorithm, we use $n_u = 49$ inducing input points (a grid of 7 by 7), which is sufficient since using more inducing input points will give exactly the same results. For the PITC algorithm, we can also vary the size of the subgroups of measurements that we use. We will use either groups of 50 or groups of 200 measurements.

For each of these algorithms, we will examine two quantities. First there is the runtime of the algorithm, and secondly the accuracy. This accuracy is quantified through the root mean squared error of the output.

The problem here is that we do not precisely know the ‘correct’ output. However, if

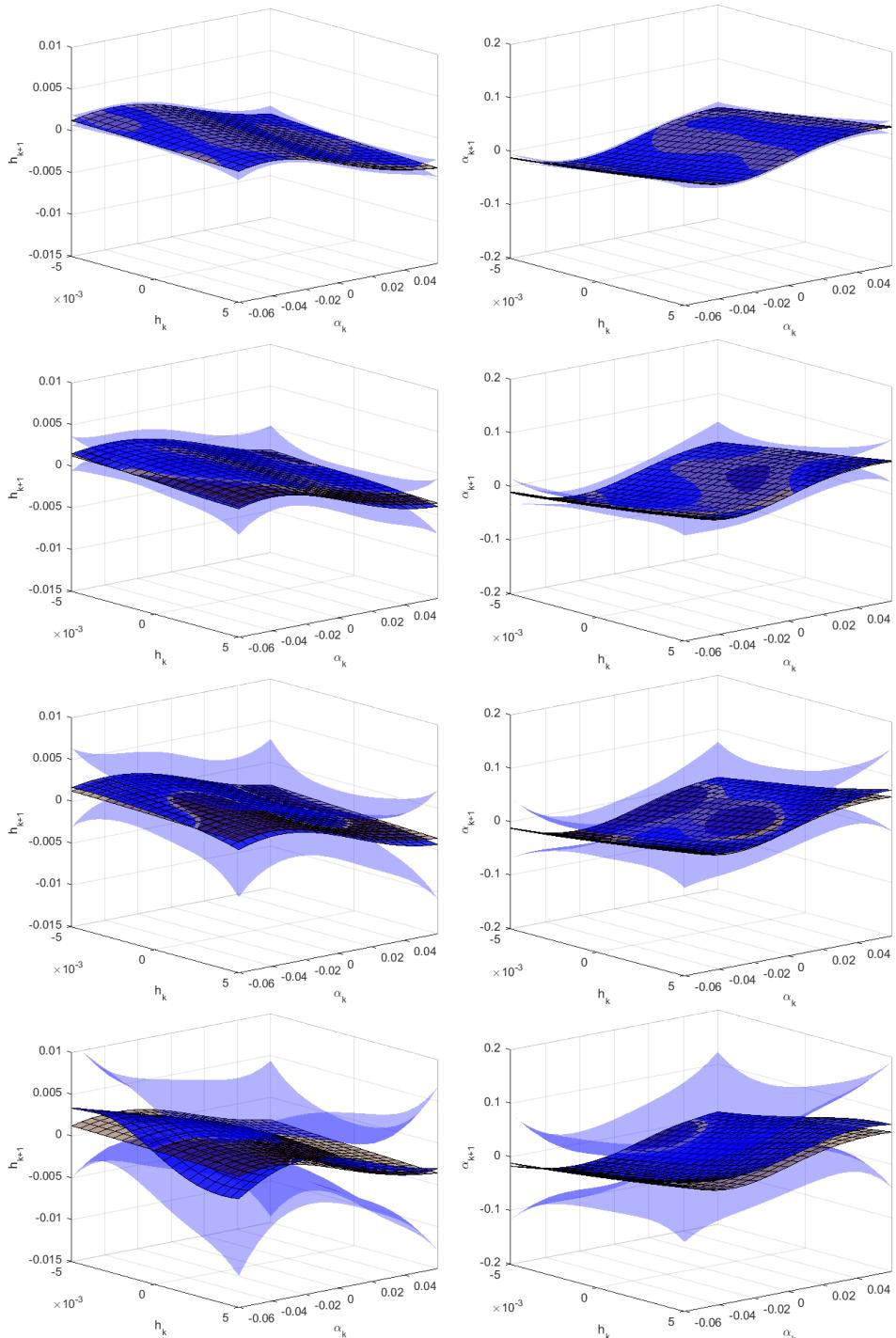


Figure 4.10: A remake of Figure 2.15. We used $n_m = 1000$ measurements, through which we approximated h_{k+1} (left) and α_{k+1} (right). The algorithms applied were regular GP regression (top row), PITC regression with groups of size 200 (second row) and 50 (third row) and FITC regression (bottom row). The accuracy of the algorithms becomes lower as more simplifying assumptions are made.

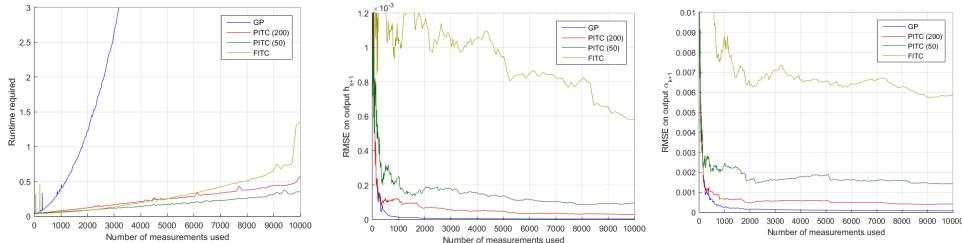


Figure 4.11: The runtime (left) and accuracy (middle and right) of the GP, PITC and FITC algorithms with respect to the number of measurements used. For the PITC, also the size of the added groups of measurements was varied, either being 50 or 200. These measurements were randomly grouped together. Tests were run on a regular home computer. This resulted in the variations (spikes) in the runtime, which are most likely due to background processes.

4

we put the system in a stationary initial state 10000 times like in Figure 2.14 and apply GP regression to this, we get an extremely accurate estimate, which can be considered the ‘true’ output of the system. We use this to calculate the error and subsequently the RMSE of the predictions.

When using $n_m = 1000$ measurements, we get the predictions shown in 4.10. The way in which the runtime and the accuracy of the algorithms depend on the number of measurements is shown in Figure 4.11.

Some very interesting conclusions can be drawn from the figures. First of all, in Figure 4.11 (left) we see that the runtime of both FITC and PITC is much better than the runtime of GP regression for large numbers of measurements. While the runtime of GP regression scales roughly cubically, the runtime of FITC and PITC indeed scales linearly.

However, contrary to the experiment of Section 4.4.1, now there is a strong difference in accuracy between the algorithms. Both the RMSE plots of Figure 4.11 (middle and right) and the approximations of Figure 4.10 show that the GP regression algorithm performs much better than the other algorithms. So what causes this? Why are the results so different?

To get from the GP algorithm to the FITC algorithm, we made two assumptions: the inducing input assumption and the FITC assumption. If we only make the inducing input assumption, and hence apply the sparse GP algorithm of Section 4.1.3, we get nearly the same results as the regular GP regression algorithm. (Results are not shown here, but for $n_u = 49$ the RMSE is less than 1% higher, and for $n_u = 81$ the RMSE is identical.) So the inaccuracies are hence caused by the FITC assumption. This is also confirmed by the fact that the PITC algorithm performs much better than the FITC algorithm. So apparently, while the FITC assumption did not have much effect in two-dimensional applications, it does have a significant effect in higher-dimensional applications.

To solve this issue, we either have to stick with the regular GP regression algorithm and suffer very long runtimes, or use the PITC algorithm with sufficiently large subgroups. Which option to go for depends on how much runtime and how many measurements are available.

One possible improvement can still be made in the algorithm by grouping the right measurements together in the PITC algorithm. Currently, the PITC algorithm just put

random measurements together in groups. By putting measurements together in groups that are strongly correlated, or whose covariance deviates strongly with respect to the covariance assumed by the FITC assumption, we might still be able to gain additional accuracy.

4.5. OVERVIEW OF LITERATURE AND CONTRIBUTIONS

As in every chapter, we take a look at the earlier literature on this subject as well as suggestions for future research.

4.5.1. LITERATURE OVERVIEW

As Gaussian processes gained popularity in the late 90s and early 2000s, so did the interest rise in applying it to larger data sets. Several different people came up with methods to reduce the runtime of the regression algorithm. A few examples are the contributions by [Smola and Bartlett \(2001\)](#), [Csató and Opper \(2002\)](#), [Seeger et al. \(2003\)](#) and [Snelson and Ghahramani \(2006a\)](#).

This all came together in the unifying view presented by [Candela and Rasmussen \(2005\)](#). They showed that the algorithms set up by the other all used some sort of inducing input function. And although the exact details differed – some algorithms used predefined inducing input points, others set them equal to measurements, others tuned them – the main idea of all the algorithms was very similar.

In the meantime, various other techniques are tried to enable GP regression to be applied to big data. The most promising one was developed by [Hensman et al. \(2013\)](#), who use the theories on *stochastic variational inference* from [Hoffman et al. \(2012\)](#), [Hensman et al. \(2012\)](#) to set up a new regression algorithm. They claim to have reduced the runtime of the algorithm from $\mathcal{O}(n_m n_u^2)$ to $\mathcal{O}(n_u^3)$. For more background on variational inference, see [Titsias \(2009\)](#), [Titsias and Lawrence \(2010\)](#), [Gal et al. \(2014\)](#), [McHutchon \(2014\)](#).

When dealing with big data sets, it may of course also be wise to apply multiple processing units which share the computational and memory load among them. Setting up the algorithm in such a distributed way is not as trivial as it initially sounds though. More information on this can be found in the work of [Deisenroth and Ng \(2015\)](#).

There has also been a variety of work done on online GP regression. An early paper on this was written by [Csató and Opper \(2002\)](#). This paper went beyond the FITC assumption and assumed that, given the inducing function values \underline{f}_u , the measurement function values \underline{f}_m could be known deterministically. This assumption is equivalent to assuming $K_{mm} = K_{mu}K_{uu}^{-1}K_{mu}$ and was dubbed the *Deterministic Training Conditional* (DTC) assumption by [Candela and Rasmussen \(2005\)](#).

Others expanded on this work, like [Ranganathan et al. \(2011\)](#) who update and down-date a Cholesky factorization of the covariance matrix K . In the end, they aim to wind up with the most efficient *Subset of Data* (SoD) approximation. That is, they only use the measurements that together give the best approximation. Other work was done by [Kou et al. \(2013\)](#), who use an FITC approximation but constrain themselves to inducing inputs chosen from the set of training inputs. So it seems that, without realizing it, these people have constrained themselves to unnecessary assumptions.

Such extra constraining assumptions were not made by Huber (2013), Huber (2014). His *recursive Gaussian process* method was inspired by the Kalman filter. As such, it does not use conventional GP notations. At the same time I developed my own online GP methods, published through Bijl et al. (2015). After careful analysis, both these methods turn out to be exactly the same and effectively are implementations of the online FITC algorithm. In the meantime, the expressions for this algorithm, as well as those of the online PITC algorithm, have been developed further to the more intuitive form in which they have been presented in this chapter.

4.5.2. SUGGESTIONS FOR FURTHER RESEARCH

In science, and in life in general, every answered question usually raises two more. Below are some things that I am still curious about.

4

- **Constrained FITC regression**

Can the constrained GP regression algorithm of Section 3.3 be combined with the FITC algorithm? If so, what would the intuitive view on this be? And what are the resulting regression equations?

- **Tuning the number of inducing input points**

More inducing input points will always be able to store more information than fewer inducing input points. However, after a certain number of inducing input points, the additional ‘information gain’ seems to be minimal. At the same time, using more inducing input points does significantly increase the computational complexity of all algorithms. Would there be some way to tune the number of inducing input points? In other words, can we figure out when the additional information gain of using an extra inducing input point becomes so small that it is pointless to add more inducing input points? A good starting point to investigate this would be to plot the entropy of \hat{f}_* (or the log-determinant $\log|\Sigma_{**}|$) that can optimally be obtained, versus the number n_u of inducing input points used, for a certain example set of measurements (X_m, \hat{f}_m) .

- **Online tuning/shifting of inducing input points**

We know from Section 4.3.3 how to change the inducing input points online. But is it also possible to tune them in an online way, if we only receive one measurement at a time and have to discard it before we get the next measurement? Can we adjust our inducing points in such a way as to improve one of the quantities like (4.44) or (4.54) that we want to optimize?

- **Improving subgroups used by the PITC algorithm**

In this chapter, when applying the PITC algorithm, we have always randomly put measurements together in groups. However, is there a better way of putting measurements together in groups? Do we get more accurate predictions if we put measurements together that are strongly correlated? Or is there some other way of putting measurements together that results in a better accuracy?

- **Analyzing the numerical stability of online updates**

In Section 4.2.5 we noted that the many updates in online GP regression algorithms may cause numerical problems. When exactly do these problems occur? How sig-

nificant are they? And what would be the best way to prevent them or work around them? Do the different representations discussed in Section 4.3.4 have any beneficial effects? Might it be better to store $\Sigma_{uu}K_{uu}^{-1}$ instead of Σ_{uu} ? Or instead only store the Cholesky decomposition of Σ_{uu} ? Or are there any better representations?

- **Reducing the computational complexity even further.**

The runtime of the algorithm is now $\mathcal{O}(n_m n_u^2)$. When large n_u are necessary, for instance in multi-dimensional problems, this may cause problems. Can we reduce this runtime further?

The main idea here is that we do not need the full matrix Σ_{uu} . After all, when two inducing input points \mathbf{x}_{ui} and \mathbf{x}_{uj} are far apart, then Σ_{uiuj} will be close to zero anyway, even before we start incorporating measurement data. This leads to the idea of using a *cut-off covariance function* $k(\mathbf{x}, \mathbf{x}')$ which is zero whenever the distance between \mathbf{x} and \mathbf{x}' exceeds a given threshold (for instance $2\lambda_x$) and otherwise remains unchanged. If we take this into account in the proper way while doing regression, we should be able to obtain a reduction in the runtime. But how exactly does this work? And what kind of gains could we expect?

- **Approximating slowly changing functions**

So far we have assumed that the function we are approximating, though noisy, is constant in time. Now suppose that this function actually changes slowly over time. (An example application is discussed in Section 6.5.7.) Can we take this into account? Maybe we can do so by increasing the noise variance of older measurements? Or, when using inducing input points, can we slowly increase the covariance matrix Σ_{uu} over time? Or are there better methods to take these slow function changes into account?

- **Detecting a sudden change in the approximated function**

Suppose that we are approximating some function $f(\mathbf{x})$, taking measurements from it. But suddenly, without us being aware of it, something has changed in the system, and now we take our measurements from a rather different function $f'(\mathbf{x})$. Can we, only based on the measurements that we get, detect this change? One idea is to look at the likelihood $p(\hat{f}_m | \mathbf{x}_m)$ that we obtain the measurement value \hat{f}_m . If this likelihood is too small, and its reciprocal (the so-called *surprise*) is hence too high, possibly for multiple measurements in a row, then this could indicate that we are now approximating a different function.

But that leaves the question of how we should deal with it. Should we fully ignore all earlier measurements and start over? Or should we still take them into account in some way? One suggestion is to increase the noise corresponding to these older measurements, based on the likelihood of the measurements we are obtaining now. In other words, being very surprised about a new measurements would cause us to reduce the believed accuracy of earlier measurements. But by how much should we do this? Or is there potentially a better way of taking this into account?

5

NOISY INPUT GAUSSIAN PROCESS REGRESSION

Summary — When applying Gaussian process regression we have always assumed that there is noise on the output measurements f_m , but not on the input points \mathbf{x} . This assumption does of course not always hold: the input points can be subject to noise as well.

When the trial input points \mathbf{x}_ are subject to noise, we can integrate over all possible trial input points. This will not result in a Gaussian distribution for the output though. One solution is to switch to numerical techniques. The more conventional solution is to apply moment matching instead. When we do, we analytically calculate the mean and covariance of the resulting distribution and use those to approximate the result as a Gaussian distribution.*

When the measurement input points \mathbf{x}_m are stochastic, these tricks do not work anymore. One way to work around this is to take the noise on the input points \mathbf{x}_m into account through the output noise variance $\hat{\Sigma}_m$. The more the function we are approximating is sloped, the more we should take input noise into account like this. We can apply this idea to regular Gaussian process regression, resulting in the NIGP algorithm, or we can apply it to sparse methods like FITC, resulting in the SONIG algorithm.

The SONIG algorithm is capable of incorporating new measurements $(\hat{\mathbf{x}}_m, \hat{f}_m)$ one by one in a computationally efficient manner. When doing so, it provides us with Gaussian approximations of the posterior distributions of both the input and the output, as well as the posterior covariance between these distributions. With this data it is possible to set up for instance a nonlinear system identification algorithm.

In Gaussian process regression we have always assumed that the measured function values f_m are subjected to noise, while the input points \mathbf{x}_m or \mathbf{x}_* are known exactly. But what happens if there is also uncertainty in the input points? In this chapter we look at how to deal with that.

We start off by studying the case where only the trial input \mathbf{x}_* is uncertain (Section 5.1). These methods cannot be applied to stochastic measurement points \mathbf{x}_m , so we look into dealing with that afterwards (Section 5.2). We then discuss some extensions to the methods we have developed (Section 5.3), before we do some experiments (Section 5.4) and discuss existing literature (Section 5.5).

It is also worthwhile to note that this chapter is mathematically rather heavy. Usually, most of the mathematics is put in the appendix. However, for this chapter it is fundamental for the algorithms to understand the probability theory behind it, so that is why this theory has *not* been relegated to appendices. For the matrix derivations, we do refer to appendices.

5

5.1. USING STOCHASTIC TRIAL POINTS

Suppose that we have a known set of measurement points X_m and corresponding measured function values \hat{f}_m . We now want to predict the function value f_* (that is, determine its posterior distribution) for some input point \mathbf{x}_* . If \mathbf{x}_* is deterministic, this can directly be done through the GP regression equation (2.30), repeated as

$$\begin{aligned} \underline{f}_* &\sim \mathcal{N}(\mu_*(\mathbf{x}_*), \Sigma_{**}(\mathbf{x}_*)), \\ \mu_*(\mathbf{x}_*) &= m(\mathbf{x}_*) + k(\mathbf{x}_*, X_m)(K_{mm} + \hat{\Sigma}_{f_m})^{-1}(\hat{f}_m - \mathbf{m}_m), \\ \Sigma_{**}(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, X_m)(K_{mm} + \hat{\Sigma}_{f_m})^{-1}k(X_m, \mathbf{x}_*). \end{aligned} \quad (5.1)$$

However, now suppose that \mathbf{x}_* is instead a random variable $\underline{\mathbf{x}}_*$, subject to a Gaussian distribution $\underline{\mathbf{x}}_* \sim \mathcal{N}(\hat{\mathbf{x}}_*, \hat{\Sigma}_{x_*})$. Here, $\hat{\mathbf{x}}_*$ can be seen as the measured value of $\underline{\mathbf{x}}_*$, while $\hat{\Sigma}_{x_*}$ is the corresponding measurement noise variance. What is now the posterior distribution of \underline{f}_* ? That is the question we will find an answer to in this section.

We start by looking at the main idea, which is to integrate over all possible trial points (Section 5.1.1). This provides us with some problems, and one way to work around this is through moment matching, which we will briefly look into (Section 5.1.2). Using moment matching, we then derive expressions for the mean (Section 5.1.3) and the variance (Section 5.1.4). We derive the same expressions in case we are using a sparse GP regression method (Section 5.1.5). Finally, we examine the case where we have multiple stochastic trial input points (5.1.6).

5.1.1. INTEGRATING OVER POSSIBLE TRIAL POINTS

We want to find the posterior distribution $p(f_*)$ of \underline{f}_* . We can calculate this through marginalization. (See Theorem B.1 for background on this.) We then have

$$p(f_*) = \int_X p(f_* | \mathbf{x}_*) p(\mathbf{x}_*) d\mathbf{x}_*. \quad (5.2)$$

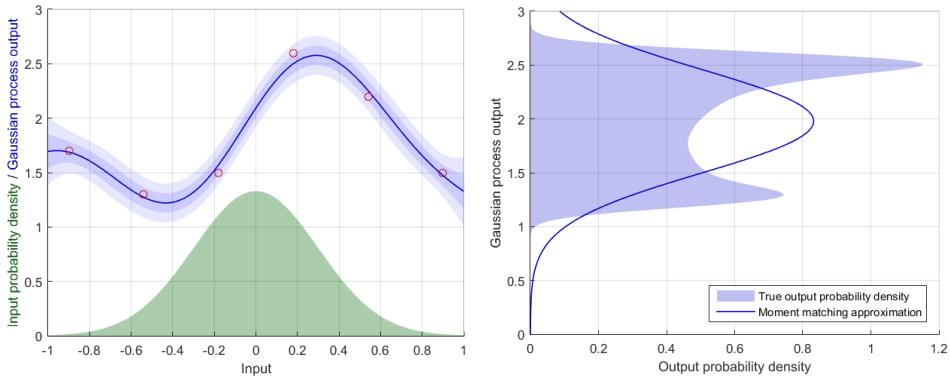


Figure 5.1: An example Gaussian process (left) to which we feed a trial input of $\underline{x}_* \sim \mathcal{N}(0, 0.3^2)$, whose distribution is also shown (left bottom). The resulting probability density of the Gaussian process output is found through (5.2) (right). This distribution is not Gaussian, but it can be approximated as a Gaussian through moment matching (Section 5.1.2), where it is possible to analytically calculate the mean through (5.9) and the variance through (5.19). Note that in this case, because \underline{x}_* has a very large variance, this posterior distribution is not close to being Gaussian. Moment matching is hence a crude approximation. However, usually stochastic trial points have smaller variances, and then this is much less of an issue.

In this expression, the first probability $p(f_* | \underline{x}_*)$ is the distribution of f_* for a known trial input \underline{x}_* , given by (5.1), while the second probability $p(\underline{x}_*)$ is the distribution of \underline{x}_* , which we know as $\mathcal{N}(\hat{\underline{x}}_*, \hat{\Sigma}_{\underline{x}_*})$.

Can we solve the integral from (5.2) to find the probability density function $p(f_*)$? We can most certainly try. If we fill in the respective terms for the probabilities, we get

$$p(f_*) = \int_X \frac{1}{\sqrt{|2\pi\Sigma_{**}(\underline{x}_*)|}} \exp\left(-\frac{1}{2} (f_* - \mu_*(\underline{x}_*))^T \Sigma_{**}^{-1}(\underline{x}_*) (f_* - \mu_*(\underline{x}_*))\right) \frac{1}{\sqrt{|2\pi\hat{\Sigma}_{\underline{x}_*}|}} \exp\left(-\frac{1}{2} (\underline{x}_* - \hat{\underline{x}}_*)^T \hat{\Sigma}_{\underline{x}_*}^{-1} (\underline{x}_* - \hat{\underline{x}}_*)\right) d\underline{x}_*. \quad (5.3)$$

Here we see that, to solve the integral, we effectively have to integrate over an exponential of an expression involving a matrix inverse, which in turn depends in a nonlinear way on the integrating parameter \underline{x}_* . Solving this integral analytically is therefore very complicated, if not impossible.

Luckily, there are numerical methods through which we can at least visualize the process that is going on. This gives us an intuitive view of what (5.2) actually does, which is shown in Figure 5.1.

This figure also directly shows us a problem with this approach. The resulting distribution is generally not Gaussian, and we always like to work with Gaussian distributions. After all, then we only need two numbers (the mean and the variance) to describe the full distribution. We hence have two options, if we want to continue.

Option one is to accept that our distributions are not Gaussian anymore and deal with any kind of distribution we may encounter. This can be done in various ways. For example, we could approximate all non-Gaussian distributions as sum of multiple

Gaussian distributions (the *Gaussian mixture model*). Another option is using numerical (particle) methods like the *Monte Carlo* methods described briefly in Section 6.2. Though fascinating, this all results in a whole field of research on its own, so we will not look further into this.

Option two is force all distributions to be Gaussian anyway. This is called moment matching, and before we continue, we will study some background theory behind moment matching.

5.1.2. INTERMEZZO: BACKGROUND BEHIND MOMENT MATCHING

Suppose we have some parameter f with a very complicated distribution. Generally put, the main idea behind *moment matching* is to take this complicated distribution and approximate it as some easier distribution with exactly the same moments. We then assume that f has this easier distribution. With *moments*, we mean the mean (first moment), the variance (second moment) and possibly higher moments like the skewness (third moment), the kurtosis (fourth moment), and so on.

First, consider the case where we only keep the first moment the same. So we replace the distribution of f with a simpler one with the same mean $\mathbb{E}[f]$. The most simple and obvious case of this is to just substitute the random variable f with a deterministic value $f = \mathbb{E}[f]$. So we replace the uncertain parameter by its mean. Effectively, this sets all higher order moments to zero. It is very simple and effective, but any information about the spread of f is lost.

A more powerful form of moment matching also takes into account the second moment. We now replace the distribution of f by one with the same mean and variance. The most simple and obvious case of this is to go for a Gaussian distribution. Again, this effectively sets all higher order moments to minimum values (now not always zero) forcing the distribution of f to be Gaussian.

It is also possible to take higher order moments into account. However, in this case the resulting approximated distribution will not be Gaussian anymore, because a Gaussian distribution is already fully defined by its first and second moments. (In fact, the third moment of a Gaussian distribution is always zero.) In addition, for multivariate parameters f it is rather difficult to incorporate third and higher moments. Because of this, we will not use third or higher moments here and stick with moment matching using only the first two moments.

Let's take a brief moment to study what moment matching effectively comes down to. Effectively, we take the distribution of f , calculate its mean and variance, remember only these two parameters, and then throw all other data (the actual distribution of f) out of the window. Next, knowing only the mean and variance, we have to come up with a new distribution for f .

How do we do this? Well, ideally this distribution should have as little information as possible, other than that it has the correct mean and variance. After all, we cannot just assume data that isn't there.

One way to describe this mathematically, is to demand that the distribution should have the highest possible *entropy*, where the entropy is a measure of how much uncertainty or how little information is contained within the distribution. (A deterministic number has zero entropy. For a quick introduction into entropy, see Section 6.4.3.) The

so-called *maximum entropy probability distribution* can now be shown to be the Gaussian distribution. So if we only know the mean and variance of \underline{f} , it is wisest to assume that \underline{f} has a Gaussian distribution.

5.1.3. THE EXPECTED VALUE OF THE TRIAL FUNCTION VALUE

Let's go back to our parameter \underline{f}_* . We know that its distribution $p(\underline{f}_*)$ is given by (5.2), and that this distribution is non-Gaussian. To approximate as a Gaussian, we need to know its mean μ_* and its covariance matrix Σ_{**} . (Note that these two parameters are the mean and covariance *after* integrating over all possible values of \underline{x}_* , while $\mu_*(\underline{x}_*)$ and $\Sigma_{**}(\underline{x}_*)$ are the mean and covariance for a specific given value of \underline{x}_* .) We will start by finding an expression for the mean μ_* . Later on we derive the covariance matrix Σ_{**} .

Per definition, the mean of \underline{f}_* can be found through

$$\mu_* \equiv \mathbb{E} [\underline{f}_*] = \int_{-\infty}^{\infty} f_* p(f_*) df_* = \int_{-\infty}^{\infty} \int_X f_* p(f_* | \underline{x}_*) p(\underline{x}_*) d\underline{x}_* df_*. \quad (5.4)$$

Contrary to the integral we faced earlier, this integral can be solved analytically for various kernels like the squared exponential kernel. To do so, we have to interchange the integrals. This gives us

$$\mu_* = \int_X \int_{-\infty}^{\infty} f_* p(f_* | \underline{x}_*) p(\underline{x}_*) d\underline{x}_* df_* = \int_X \left(\int_{-\infty}^{\infty} f_* p(f_* | \underline{x}_*) df_* \right) p(\underline{x}_*) d\underline{x}_*. \quad (5.5)$$

The inner integral in this expression is the mean of f_* at some known trial input point \underline{x}_* , which equals $\mu_*(\underline{x}_*)$. We therefore have

$$\begin{aligned} \mu_* &= \int_X \mu_*(\underline{x}_*) p(\underline{x}_*) d\underline{x}_* \\ &= \int_X \left(m(\underline{x}_*) + k(\underline{x}_*, X_m) (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{f}_m - \mathbf{m}_m) \right) p(\underline{x}_*) d\underline{x}_*. \end{aligned} \quad (5.6)$$

So effectively, to find the mean of \underline{f}_* , we integrate over all possible means which \underline{f}_* can have for varying inputs \underline{x}_* .

Let's now define the parameters¹ \bar{m}_* and \bar{K}_{*m} as

$$\bar{m}_* \equiv \int_X m(\underline{x}_*) p(\underline{x}_*) d\underline{x}_*, \quad (5.7)$$

$$\bar{K}_{*m} \equiv \int_X k(\underline{x}_*, X_m) p(\underline{x}_*) d\underline{x}_*. \quad (5.8)$$

In this case, the posterior mean μ_* , given that \underline{x}_* has the distribution $\mathcal{N}(\hat{\mathbf{x}}_*, \hat{\Sigma}_{x_*})$, equals

$$\begin{aligned} \mu_* &= \int_X m(\underline{x}_*) p(\underline{x}_*) d\underline{x}_* + \left(\int_X k(\underline{x}_*, X_m) p(\underline{x}_*) d\underline{x}_* \right) (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{f}_m - \mathbf{m}_m) \\ &= \bar{m}_* + \bar{K}_{*m} (K_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{f}_m - \mathbf{m}_m). \end{aligned} \quad (5.9)$$

¹In other literature, it is often assumed that the mean function is zero, so \bar{m}_* is ignored. Additionally, Deisenroth (2010) used the notation \mathbf{q} for \bar{K}_{*m} . I will go for \bar{K}_{*m} because it makes it much more clear how we should eventually use it in the regression equation.

It is a familiar expression with slightly adjusted parameters. The values of \bar{m}_* and \bar{K}_{*m} here naturally depend on which mean and covariance function we choose.

Let's pick the ones that we usually use: the zero mean function $m(\mathbf{x}) = 0$ and the squared exponential covariance function (2.35). In this case, \bar{m}_* is obviously zero, while \bar{K}_{*m} can be found element-wise, directly from Theorem A.18, as

$$\begin{aligned}\bar{K}_{*m_i} &= \int_X k(\mathbf{x}_*, \mathbf{x}_{m_i}) \mathcal{N}(\mathbf{x}_* | \hat{\mathbf{x}}_*, \hat{\Sigma}_{x_*}) d\mathbf{x}_* \\ &= \lambda_f^2 \sqrt{\frac{|\Lambda_x|}{|\Lambda_x + \hat{\Sigma}_{x_*}|}} \exp\left(-\frac{1}{2} (\hat{\mathbf{x}}_* - \mathbf{x}_{m_i})^T (\Lambda_x + \hat{\Sigma}_{x_*})^{-1} (\hat{\mathbf{x}}_* - \mathbf{x}_{m_i})\right).\end{aligned}\quad (5.10)$$

It is interesting to note that, if $\hat{\Sigma}_{x_*} = 0$, and there is hence no uncertainty in $\underline{\mathbf{x}}_*$, then \bar{K}_{*m} reduces back to K_{*m} , as can be expected.

5.1.4. THE VARIANCE OF THE TRIAL FUNCTION VALUE

To apply moment matching, we also need to know the covariance Σ_{**} of \underline{f}_* . This is derived in a similar way, but the process is a bit more complicated.

5

Per definition, we have

$$\Sigma_{**} = \mathbb{V}[\underline{f}_*] = \mathbb{E}\left[\left(\underline{f}_* - \mathbb{E}[\underline{f}_*]\right)^2\right]. \quad (5.11)$$

The key to solving this is to introduce $\mu_*(\mathbf{x}_*)$ into the expression. Note that $\mu_*(\mathbf{x}_*)$ is the mean for a specific trial input \mathbf{x}_* , while μ_* is the expected mean, integrated over all potential trial inputs. Using this, we can rewrite the above as

$$\begin{aligned}\Sigma_{**} &= \mathbb{E}\left[\left(\underline{f}_* - \mu_*(\mathbf{x}_*)\right) + \left(\mu_*(\mathbf{x}_*) - \mu_*\right)\right]^2 \\ &= \mathbb{E}\left[\left(\underline{f}_* - \mu_*(\mathbf{x}_*)\right)^2\right] + \mathbb{E}\left[\left(\mu_*(\mathbf{x}_*) - \mu_*\right)^2\right] + 2\mathbb{E}\left[\left(\underline{f}_* - \mu_*(\mathbf{x}_*)\right)\left(\mu_*(\mathbf{x}_*) - \mu_*\right)\right].\end{aligned}\quad (5.12)$$

We will solve these terms one by one, starting with the third, continuing with the first and ending with the second. The third term actually turns out to be zero. The short way of explaining this would be to say that only the first term within the expectation operator depends on \underline{f}_* , and taking the expectation over \underline{f}_* would turn this term into zero. This explanation may be a bit too quick to follow though, so we will just expand the equation to show what we mean with it. When we do, we find

$$\begin{aligned}\mathbb{E}\left[\left(\underline{f}_* - \mu_*(\mathbf{x}_*)\right)\left(\mu_*(\mathbf{x}_*) - \mu_*\right)\right] &= \int_{-\infty}^{\infty} \int_X (f_* - \mu_*(\mathbf{x}_*)) (\mu_*(\mathbf{x}_*) - \mu_*) p(f_* | \mathbf{x}_*) p(\mathbf{x}_*) d\mathbf{x}_* df_* \\ &= \int_X \int_{-\infty}^{\infty} (f_* - \mu_*(\mathbf{x}_*)) (\mu_*(\mathbf{x}_*) - \mu_*) p(f_* | \mathbf{x}_*) p(\mathbf{x}_*) df_* d\mathbf{x}_* \\ &= \int_X \left(\int_{-\infty}^{\infty} (f_* - \mu_*(\mathbf{x}_*)) p(f_* | \mathbf{x}_*) df_* \right) (\mu_*(\mathbf{x}_*) - \mu_*) p(\mathbf{x}_*) d\mathbf{x}_*.\end{aligned}\quad (5.13)$$

Now the inner integral evaluates as zero. After all, given \mathbf{x}_* , the expectation of \underline{f}_* becomes $\mu_*(\mathbf{x}_*)$. And because the inner integral is zero, the whole term is zero as well.

Next, we continue with the first term of the sum (5.12). It can be evaluated as

$$\begin{aligned}\mathbb{E} \left[(\underline{f}_* - \mu_*(\mathbf{x}_*))^2 \right] &= \int_{-\infty}^{\infty} \int_X (\underline{f}_* - \mu_*(\mathbf{x}_*))^2 p(f_* | \mathbf{x}_*) p(\mathbf{x}_*) d\mathbf{x}_* df_* \\ &= \int_X \left(\int_{-\infty}^{\infty} (\underline{f}_* - \mu_*(\mathbf{x}_*))^2 p(f_* | \mathbf{x}_*) df_* \right) p(\mathbf{x}_*) d\mathbf{x}_* \\ &= \int_X \Sigma_{**}(\mathbf{x}_*) p(\mathbf{x}_*) d\mathbf{x}_*.\end{aligned}\quad (5.14)$$

Note that we have reduced the inner integral to $\Sigma_{**}(\mathbf{x}_*)$. This is because the inner integral denotes the variance of \underline{f}_* for a given trial input point \mathbf{x}_* . Since we know $\Sigma_{**}(\mathbf{x}_*)$ from (5.1), we can solve this integral. We will do so soon.

But first we look at the second term from (5.12). Note that there is no f_* in the integrand here, so the integral over $p(f_* | \mathbf{x}_*)$ reduces to one. As a result, we have

$$\begin{aligned}\mathbb{E} \left[(\mu_*(\mathbf{x}_*) - \mu_*)^2 \right] &= \int_{-\infty}^{\infty} \int_X (\mu_*(\mathbf{x}_*) - \mu_*)^2 p(f_* | \mathbf{x}_*) p(\mathbf{x}_*) d\mathbf{x}_* df_* \\ &= \int_X (\mu_*(\mathbf{x}_*) - \mu_*)^2 p(\mathbf{x}_*) d\mathbf{x}_*.\end{aligned}\quad (5.15)$$

We also know the expression of $\mu_*(\mathbf{x}_*)$ from (5.1), which means we should be able to solve this integral too.

Concluding, the expression for Σ_{**} equals

$$\Sigma_{**} = \int_X \left(\Sigma_{**}(\mathbf{x}_*) + (\mu_*(\mathbf{x}_*) - \mu_*)^2 \right) p(\mathbf{x}_*) d\mathbf{x}_*. \quad (5.16)$$

This expression is actually quite intuitive. The first term $\Sigma_{**}(\mathbf{x}_*)$ adds up all the variances that are already present in \underline{f}_* , while the second term takes into account a varying mean $\mu_*(\mathbf{x}_*)$.

Now let's see how we can solve the integral. To do so, we will use the shorthand notation $P = (K_{mm} + \hat{\Sigma}_{f_m})$ and $\boldsymbol{\alpha} = P^{-1}(\hat{\mathbf{f}}_m - \mathbf{m}_m)$, just like we did in Section 3.1. The expression for Σ_{**} can now be written as

$$\begin{aligned}\Sigma_{**} &= \int_X \left(\Sigma_{**}(\mathbf{x}_*) + \mu_*(\mathbf{x}_*)^2 \right) p(\mathbf{x}_*) d\mathbf{x}_* - \mu_*^2 \\ &= \int_X \left(k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, X_m) P^{-1} k(X_m, \mathbf{x}_*) + (m(\mathbf{x}_*) + k(\mathbf{x}_*, X_m) \boldsymbol{\alpha})^2 \right) p(\mathbf{x}_*) d\mathbf{x}_* - \mu_*^2 \\ &= \int_X \left(k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, X_m) (P^{-1} - \boldsymbol{\alpha} \boldsymbol{\alpha}^T) k(X_m, \mathbf{x}_*) \right. \\ &\quad \left. + m(\mathbf{x}_*) k(\mathbf{x}_*, X_m) \boldsymbol{\alpha} + m(\mathbf{x}_*)^2 \right) p(\mathbf{x}_*) d\mathbf{x}_* - \mu_*^2.\end{aligned}\quad (5.17)$$

To solve this further, we need to specify which mean and covariance function we use. If we once more go for the zero mean function, then everything involving $m(\mathbf{x}_*)$ will drop out. Going for the squared exponential covariance function also means that $k(\mathbf{x}_*, \mathbf{x}_*) =$

λ_f^2 . That leaves us with one more difficult integral to solve. The key here is to define $Q = P^{-1} - \alpha\alpha^T$. If we denote the elements of Q as Q_{ij} , then we have

$$\Sigma_{**} = \lambda_f^2 - \int_X \sum_{i=1}^{n_m} \sum_{j=1}^{n_m} k(\mathbf{x}_*, \mathbf{x}_{m_i}) Q_{ij} k(\mathbf{x}_{m_j}, \mathbf{x}_*) \mathcal{N}(\mathbf{x}_* | \hat{\mathbf{x}}_*, \hat{\Sigma}_{x_*}) d\mathbf{x}_* - \mu_*^2. \quad (5.18)$$

Using Theorem A.19, the solution of this directly follows as²

$$\begin{aligned} \Sigma_{**} &= \lambda_f^2 - \lambda_f^4 \sqrt{\frac{|\Lambda_x|}{|\Lambda_x + 2\hat{\Sigma}_{x_*}|}} \sum_{i=1}^{n_m} \sum_{j=1}^{n_m} Q_{ij} \exp\left(-\frac{1}{2} \left(\mathbf{x}_{m_i} - \mathbf{x}_{m_j}\right)^T (2\Lambda_x)^{-1} \left(\mathbf{x}_{m_i} - \mathbf{x}_{m_j}\right)\right) \\ &\quad \exp\left(-\frac{1}{2} \left(\frac{\mathbf{x}_{m_i} + \mathbf{x}_{m_j}}{2} - \hat{\mathbf{x}}_*\right)^T \left(\frac{1}{2} \Lambda_x + \hat{\Sigma}_{x_*}\right)^{-1} \left(\frac{\mathbf{x}_{m_i} + \mathbf{x}_{m_j}}{2} - \hat{\mathbf{x}}_*\right)\right) - \mu_*^2. \end{aligned} \quad (5.19)$$

This is not the most intuitive expression ever, but it will have to do. With this expression, together with (5.9), we can approximate the distribution of f_{*} subject to a stochastic trial input point $\underline{\mathbf{x}}_* \sim \mathcal{N}(\hat{\mathbf{x}}_*, \hat{\Sigma}_{x_*})$.

5

5.1.5. THE MEAN AND VARIANCE OF SPARSE PREDICTIONS

Suppose that we use the sparse methods of Section 4.1. (If you have not read Section 4.1, feel free to skip this paragraph and continue with Section 5.1.6.) In this case, the regression equations of (5.1) turn, according to (4.9), into

$$\begin{aligned} f_{*} &\sim \mathcal{N}(\mu_*(\mathbf{x}_*), \Sigma_{**}(\mathbf{x}_*)), \\ \mu_*(\mathbf{x}_*) &= m(\mathbf{x}_*) + k(\mathbf{x}_*, X_u) K_{uu}^{-1} (\boldsymbol{\mu}_u - \mathbf{m}_u), \\ \Sigma_{**}(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, X_u) K_{uu}^{-1} (K_{uu} - \Sigma_{uu}) K_{uu}^{-1} k(X_u, \mathbf{x}_*). \end{aligned} \quad (5.20)$$

How can we now calculate the posterior mean μ_* and variance Σ_{**} of the trial function value f_{*} subject to an uncertain trial input point $\mathbf{x}_* \sim \mathcal{N}(\hat{\mathbf{x}}_*, \hat{\Sigma}_{x_*})$?

Because the above expression is so similar to (5.1), the process is actually very similar too. We can find the posterior mean μ_* through

$$\mu_* = \bar{m}_* + \tilde{K}_{*u} K_{uu}^{-1} (\boldsymbol{\mu}_u - \mathbf{m}_u), \quad (5.21)$$

where \bar{m}_* still satisfies (5.7). When we use the zero mean function, \bar{m}_* still becomes zero. If we also use the squared exponential covariance function, then \tilde{K}_{*u} can be found through

$$\tilde{K}_{*u} = \sqrt{\frac{|\Lambda_x|}{|\Lambda_x + \hat{\Sigma}_{x_*}|}} \exp\left(-\frac{1}{2} (\hat{\mathbf{x}}_* - \mathbf{x}_{u_i})^T (\Lambda_x + \hat{\Sigma}_{x_*})^{-1} (\hat{\mathbf{x}}_* - \mathbf{x}_{u_i})\right). \quad (5.22)$$

²In other literature, specifically in the work by Deisenroth (2010), the solution method used is slightly different. There they use a trace function instead of a summation. Naturally both methods are correct. I personally prefer the summation though, because when using the trace function in a computer script, you inherently calculate a lot of terms which you eventually do not need, making the script slower than it needs to be.

To find the posterior variance Σ_{**} , we first have to redefine $\boldsymbol{\alpha} = K_{uu}^{-1}(\boldsymbol{\mu}_u - \mathbf{m}_u)$ and

$$Q = K_{uu}^{-1}(K_{uu} - \Sigma_{uu})K_{uu}^{-1} - \boldsymbol{\alpha}\boldsymbol{\alpha}^T. \quad (5.23)$$

We can now find Σ_{**} directly through

$$\begin{aligned} \Sigma_{**} &= \lambda_f^2 - \lambda_f^4 \sqrt{\frac{|\Lambda_x|}{|\Lambda_x + 2\hat{\Sigma}_{x_*}|}} \sum_{i=1}^{n_u} \sum_{j=1}^{n_u} Q_{ij} \exp\left(-\frac{1}{2} (\mathbf{x}_{ui} - \mathbf{x}_{uj})^T (2\Lambda_x)^{-1} (\mathbf{x}_{ui} - \mathbf{x}_{uj})\right) \\ &\quad \exp\left(-\frac{1}{2} \left(\frac{\mathbf{x}_{ui} + \mathbf{x}_{uj}}{2} - \hat{\mathbf{x}}_*\right)^T \left(\frac{1}{2} \Lambda_x + \hat{\Sigma}_{x_*}\right)^{-1} \left(\frac{\mathbf{x}_{ui} + \mathbf{x}_{uj}}{2} - \hat{\mathbf{x}}_*\right)\right) - \mu_*^2. \end{aligned} \quad (5.24)$$

So it is still possible to deal with stochastic trial input points when we use any kind of sparse GP regression algorithm.

5.1.6. USING MULTIPLE TRIAL INPUT POINTS

Suppose that we have multiple stochastic trial input points $\underline{x}_{*_1}, \dots, \underline{x}_{*_n}$, each with their own distributions $\underline{x}_{*_i} \sim \mathcal{N}(\hat{\mathbf{x}}_{*_i}, \hat{\Sigma}_{x_{*_i} x_{*_i}})$. For simplicity, we assume that all inputs \underline{x}_{*_i} are independent. That is, the covariance $\hat{\Sigma}_{x_{*_i} x_{*_j}} \equiv \mathbb{V}[\underline{x}_{*_i}, \underline{x}_{*_j}]$ between two different trial input points \underline{x}_{*_i} and \underline{x}_{*_j} is assumed to be zero³. As a result, we can simply write $\hat{\Sigma}_{x_{*_i} x_{*_i}}$ into $\hat{\Sigma}_{x_{*_i}}$, which means the same. How do we now make predictions?

The first key insight here is that previously the trial function value f_* was a scalar. Now it is a vector \underline{f}_* . So the posterior mean μ_* becomes a vector $\boldsymbol{\mu}_*$ and the posterior variance Σ_{**} now turns into a covariance matrix Σ_{**} .

The second key insight is that we can actually predict the posterior distributions of f_{*_1}, \dots, f_{*_n} all separately. That is, we can use the familiar equations

$$\mu_{*_i} = \bar{m}_{*_i} + \bar{K}_{*_i m} (K_{mm} + \hat{\Sigma}_{fm})^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m), \quad (5.25)$$

$$\begin{aligned} \Sigma_{*_i *_i} &= \lambda_f^2 - \lambda_f^4 \sqrt{\frac{|\Lambda_x|}{|\Lambda_x + 2\hat{\Sigma}_{*_i}|}} \sum_{i=1}^{n_m} \sum_{j=1}^{n_m} Q_{ij} \exp\left(-\frac{1}{2} (\mathbf{x}_{mi} - \mathbf{x}_{mj})^T (2\Lambda_x)^{-1} (\mathbf{x}_{mi} - \mathbf{x}_{mj})\right) \\ &\quad \exp\left(-\frac{1}{2} \left(\frac{\mathbf{x}_{mi} + \mathbf{x}_{mj}}{2} - \hat{\mathbf{x}}_{*_i}\right)^T \left(\frac{1}{2} \Lambda_x + \hat{\Sigma}_{*_i}\right)^{-1} \left(\frac{\mathbf{x}_{mi} + \mathbf{x}_{mj}}{2} - \hat{\mathbf{x}}_{*_i}\right)\right). \end{aligned}$$

This gives us $\boldsymbol{\mu}_*$ and the diagonal elements of Σ_{**} . However, it does not give us the off-diagonal elements of Σ_{**} . In other words, it does not tell us how two different trial function values f_{*_i} and f_{*_j} are correlated. In some cases this covariance might still be needed, so we will derive it. It does mean we will have to do all the derivations once more.

³It is still possible to solve the equations if this covariance is not zero. In fact, the solution method is almost exactly the same. You will just have to replace $p(\mathbf{x}_{*_i}, \mathbf{x}_{*_j})$ by $p(\mathbf{x}_{*_i} | \mathbf{x}_{*_j})p(\mathbf{x}_{*_j})$ instead of $p(\mathbf{x}_{*_i})p(\mathbf{x}_{*_j})$ in (5.29) and work on from there. The main issue is that it will nearly double the size of all equations, so if you are planning to do the derivations, bring a lot of paper.

Our starting point is the same as before. That is,

$$\begin{aligned}\Sigma_{*i*j} &= \mathbb{E} \left[\left(f_{*i} - \mu_{*i} \right) \left(f_{*j} - \mu_{*j} \right) \right] \\ &= \mathbb{E} \left[\left(\left(f_{*i} - \mu_*(\underline{x}_{*i}) \right) + \left(\mu_*(\underline{x}_{*i}) - \mu_{*i} \right) \right) \left(\left(f_{*j} - \mu_*(\underline{x}_{*j}) \right) + \left(\mu_*(\underline{x}_{*j}) - \mu_{*j} \right) \right) \right].\end{aligned}\quad (5.26)$$

Expanding the brackets will give us four terms, but the cross terms will reduce to zero like before. We remain with

$$\begin{aligned}\Sigma_{*i*j} &= \mathbb{E} \left[\left(f_{*i} - \mu_*(\underline{x}_{*i}) \right) \left(f_{*j} - \mu_*(\underline{x}_{*j}) \right) \right] + \mathbb{E} \left[\left(\mu_*(\underline{x}_{*i}) - \mu_{*i} \right) \left(\mu_*(\underline{x}_{*j}) - \mu_{*j} \right) \right] \\ &= \int_X \int_X \left(\Sigma_{**}(\underline{x}_{*i}, \underline{x}_{*j}) + \mu_*(\underline{x}_{*i}) \mu_*(\underline{x}_{*j}) \right) p(\underline{x}_{*i}, \underline{x}_{*j}) d\underline{x}_{*i} d\underline{x}_{*j} - \mu_{*i} \mu_{*j}.\end{aligned}\quad (5.27)$$

We can expand this using the expression of $\mu_*(\underline{x}_*)$ from (5.1), as well as

5

$$\Sigma_{**}(\underline{x}_{*i}, \underline{x}_{*j}) = k(\underline{x}_{*i}, \underline{x}_{*j}) - k(\underline{x}_{*i}, X_m) (K_{mm} + \hat{\Sigma}_{fm})^{-1} k(X_m, \underline{x}_{*j}). \quad (5.28)$$

Let's once more use the zero mean function and the squared exponential covariance function. Also, we again define $Q = (P^{-1} - \boldsymbol{\alpha} \boldsymbol{\alpha}^T)$. In this case Σ_{*i*j} reduces to

$$\begin{aligned}\Sigma_{*i*j} &= \int_X \int_X \left(k(\underline{x}_{*i}, \underline{x}_{*j}) - k(\underline{x}_{*i}, X_m) Q k(X_m, \underline{x}_{*j}) \right) p(\underline{x}_{*i}, \underline{x}_{*j}) d\underline{x}_{*i} d\underline{x}_{*j} - \mu_{*i} \mu_{*j} \\ &= \int_X \int_X k(\underline{x}_{*i}, \underline{x}_{*j}) p(\underline{x}_{*i}) p(\underline{x}_{*j}) d\underline{x}_{*i} d\underline{x}_{*j} \\ &\quad - \sum_{k=1}^{n_m} \sum_{l=1}^{n_m} \left(\int_X k(\underline{x}_{*i}, \underline{x}_{m_k}) p(\underline{x}_{*i}) d\underline{x}_{*i} \right) Q_{kl} \left(\int_X k(\underline{x}_{m_l}, \underline{x}_{*j}) p(\underline{x}_{*j}) d\underline{x}_{*j} \right) - \mu_{*i} \mu_{*j}.\end{aligned}\quad (5.29)$$

Here we have used our assumption that \underline{x}_i and \underline{x}_j are independent, causing $p(\underline{x}_{*i}, \underline{x}_{*j})$ to equal $p(\underline{x}_{*i}) p(\underline{x}_{*j})$. The first double integral in the above expression is directly solved by Theorem A.20. We can write its solution as

$$\bar{K}_{*i*j} \equiv \int_X \int_X k(\underline{x}_{*i}, \underline{x}_{*j}) p(\underline{x}_{*i}) p(\underline{x}_{*j}) d\underline{x}_{*i} d\underline{x}_{*j} = \lambda_f^2 \sqrt{\frac{|\Lambda|}{|\Lambda + \hat{\Sigma}_{*i*} + \hat{\Sigma}_{*j*}|}}. \quad (5.30)$$

The second double integral has already been split up, and the individual integrals equal \bar{K}_{*m_k} and \bar{K}_{m_l*} , respectively. As a result, we find as solution

$$\Sigma_{*i*j} = \bar{K}_{*i*j} - \bar{K}_{*i} m Q \bar{K}_{m*j} - \mu_{*i} \mu_{*j}. \quad (5.31)$$

In fact, this gives rise to the idea of introducing a new covariance function which takes uncertainty into account. Let's define the *uncertainty incorporating squared exponential covariance function* of two random variables $\underline{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and $\underline{x}' \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ as

$$\bar{k}(\underline{x}, \underline{x}') = \lambda_f^2 \sqrt{\frac{|\Lambda|}{|\Lambda + \Sigma + \Sigma'|}} \exp \left(-\frac{1}{2} (\boldsymbol{\mu} - \boldsymbol{\mu}')^T (\Lambda_x + \Sigma + \Sigma')^{-1} (\boldsymbol{\mu} - \boldsymbol{\mu}') \right). \quad (5.32)$$

In this case, we can use the uncertainty incorporating Gaussian process regression equations

$$\underline{f}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_{**}) \quad (5.33)$$

$$\boldsymbol{\mu}_* = \bar{\mathbf{m}}_* + \bar{K}_{*m} (\bar{K}_{mm} + \hat{\Sigma}_{f_m})^{-1} (\hat{\mathbf{f}}_m - \mathbf{m}_m),$$

$\Sigma_{**} = \bar{K}_{**} + \bar{K}_{*m} Q \bar{K}_{m*} - \boldsymbol{\mu}_* \boldsymbol{\mu}_*^T$. Note: only for non-diagonal elements.

The main exception is that this equation does *not* work for the diagonal elements of Σ_{**} . For that, we need to use the more complicated expression (5.19). Also, it is important to keep in mind that the distribution of \underline{f}_* is not actually Gaussian, but we only approximate it as such.

Finally, I want to note that there are still many extensions that can be thought of to these ideas. For example, Deisenroth (2010) uses multiple (assumed independent) Gaussian processes, plugs in the same stochastic trial input point \underline{x}_* and calculates the posterior covariance of the resulting outputs. And there are plenty of other set-ups that you can think of. But with the tools discussed in this section, you should be able to figure out how to calculate all the covariances that result from them.

5.2. USING STOCHASTIC MEASUREMENT POINTS

Previously we looked at how to deal with noisy trial input points \underline{x}_* . Now we will assume those are deterministic again, but we assume that the measurement points $\underline{x}_{m_1}, \dots, \underline{x}_{m_{nm}}$ are noisy. To be precise, they are distributed according to $\underline{x}_{m_i} \sim \mathcal{N}(\hat{\mathbf{x}}_{m_i}, \hat{\Sigma}_{x_{m_i} x_{m_i}})$. For simplicity, we do assume that the covariance $\hat{\Sigma}_{x_{m_i} x_{m_j}}$ between two different measured input points \underline{x}_{m_i} and \underline{x}_{m_j} is zero. So the noise on the measured input points is uncorrelated.

The main question that we will now look into is, ‘How can we incorporate this uncertainty present in \underline{X}_m into our predictions?’ The idea we used for stochastic trial points (see Section 5.1) will not work, and first we will check out why (Section 5.2.1). We then look at a way to tackle this problem that does work (Section 5.2.2). Next, we try to implement these methods in a sparse and online way too. This is more complicated, so we will investigate the main ideas first (Section 5.2.3). Then we look at how we can find the posterior distribution of the measurement input point \underline{x}_{m+} (Section 5.2.4). We then make some more approximations, resulting in a sparse online method that can deal with stochastic measurement points (Section 5.2.5). This method makes use of a lot of derivatives, which we finally also take a look at (Section 5.2.6).

5.2.1. THE PROBLEM BEHIND INTEGRATING OVER MEASUREMENT POINTS

The first idea is to apply the ideas of the previous Section 5.1. When we do, we will find that these will not work, although it is still worthwhile to see exactly where it fails.

To do so, we will try to calculate the posterior mean μ_* for a deterministic trial input point \underline{x}_* . If \underline{X}_m was deterministic too, we would know that $\mu_*(\underline{X}_m)$ depends on X_m according to

$$\mu_*(\underline{X}_m) = m(\underline{x}_*) + k(\underline{x}_*, \underline{X}_m) (k(\underline{X}_m, \underline{X}_m) + \hat{\Sigma}_{f_m})^{-1} (\hat{\mathbf{f}}_m - m(\underline{X}_m)). \quad (5.34)$$

The posterior mean μ_* of f_* , taking into account all possible values of X_m , now follows through marginalization as

$$\begin{aligned}\mu_* &= \int_X \mu_*(X_m) p(X_m) dX_m \\ &= \int_X \left(m(\mathbf{x}_*) + k(\mathbf{x}_*, X_m) \left(k(X_m, X_m) + \hat{\Sigma}_{f_m} \right)^{-1} \left(\hat{\mathbf{f}}_m - m(X_m) \right) \right) p(X_m) dX_m.\end{aligned}\quad (5.35)$$

And here we can see the problem. Previously we needed to integrate over $k(\mathbf{x}_*, X_m)$ with respect to \mathbf{x}_* . We integrated over a nonlinear covariance function, which is still manageable for many covariance functions. Now, however, we need to integrate over (among others) the matrix inverse $(k(X_m, X_m) + \hat{\Sigma}_{f_m})^{-1}$, where the matrix depends on the integrating parameters in a nonlinear way. This is a bit more complicated than what we can solve analytically. As a result, we need to find other methods to deal with this.

5.2.2. THE NOISY INPUT GAUSSIAN PROCESS REGRESSION ALGORITHM

An effective solution, called the *Noisy Input Gaussian Process* (NIGP) regression algorithm, was proposed by [McHutchon and Rasmussen \(2011\)](#). The main idea here is to model the input noise as output noise too. If we do that, we can use our regular GP regression tools to take it into account.

So how does this work? The key here is to ask ourselves, ‘When plotting the GP, and when only looking at vertical distances in this graph, how far are our measurement points expected to be away from the function we are approximating?’ That is, what is the vertical distance between the circles (measurements) and the line (the approximated function) in any Gaussian process plot?

There are actually two things contributing to this vertical distance. The first is the obvious one: the output noise $\hat{\sigma}_{f_m}^2$. If we have more output noise, then measurement points will be further removed from the approximated function. But the crucial factor here is the second one: the input noise $\hat{\Sigma}_{m_i}$. And the influence of this input noise actually depends on the slope of the function we are approximating. If the function is fully flat, then input noise does not affect the vertical distance between our measurement point and the approximated function. But when the function is highly sloped, then there will be large vertical distances between measurements and the approximated function. The idea now is to add this extra vertical distance due to input noise to the output noise.

How much does the input noise contribute to this vertical distance though? We can calculate that. Let’s write the input noise as $\underline{\epsilon}_x$. So we have $\underline{\epsilon}_x \sim \mathcal{N}(\mathbf{0}, \hat{\Sigma}_{x_m})$. If the slope of the function is given by⁴ $\frac{\partial f}{\partial \mathbf{x}}$, and if this slope is assumed constant, then the extra vertical distance will be $\frac{\partial f}{\partial \mathbf{x}} \underline{\epsilon}_x$. This stochastic parameter will have a zero mean and a variance of

$$\mathbb{E} \left[\left(\frac{\partial f}{\partial \mathbf{x}} \underline{\epsilon}_x \right)^2 \right] = \mathbb{E} \left[\frac{\partial f}{\partial \mathbf{x}} \underline{\epsilon}_x \underline{\epsilon}_x^T \left(\frac{\partial f}{\partial \mathbf{x}} \right)^T \right] = \frac{\partial f}{\partial \mathbf{x}} \hat{\Sigma}_{x_m} \left(\frac{\partial f}{\partial \mathbf{x}} \right)^T. \quad (5.36)$$

So before doing Gaussian process regression, we walk through all the input points \underline{x}_{m_i}

⁴In our notation, we assume that the derivative of a multivariate function is a row vector. In other words, we will use $\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \dots & \frac{\partial f}{\partial x_n} \end{bmatrix}$.

and replace their measurement noise $\hat{\sigma}_{f_m}^2$ by

$$\hat{\sigma}_{f_{m_i}}^2 \leftarrow \hat{\sigma}_{f_m}^2 + \frac{\partial f}{\partial \mathbf{x}} \hat{\Sigma}_{x_m} \left(\frac{\partial f}{\partial \mathbf{x}} \right)^T. \quad (5.37)$$

If we incorporate this, we should be able to take into account the input noise.

There is just one problem with the suggested approach. We do not know the function $f(\mathbf{x})$ which we are approximating, and so we cannot know the derivative. Hence, we cannot approximate $f(\mathbf{x})$. This actually results in a chicken-and-egg story. We need $f(\mathbf{x})$ to know the derivatives, and we need the derivatives to find (approximate) $f(\mathbf{x})$.

The solution is to use multiple iterations. First we approximate $f(\mathbf{x})$ without any input noise. Then, based on this first approximation of $f(\mathbf{x})$, we calculate the derivatives $\frac{\partial f}{\partial \mathbf{x}}$ and use these to approximate $f(\mathbf{x})$ again. We do this a few times, until things have converged.

There is one small addition to make here. We are now using an approximation of $f(\mathbf{x})$, made through Gaussian process regression, to find the derivatives $\frac{\partial f}{\partial \mathbf{x}}$. This means that the derivatives $\frac{\partial f}{\partial \mathbf{x}}$ are actually random variables too. To be precise, they have a mean $\mathbb{E}[\frac{\partial f}{\partial \mathbf{x}}]$ and a variance $\text{V}[\frac{\partial f}{\partial \mathbf{x}}]$, both of which should be taken into account. When we do, we get a new noise incorporation law⁵

$$\hat{\sigma}_{f_{m_i}}^2 \leftarrow \hat{\sigma}_{f_m}^2 + \mathbb{E} \left[\frac{\partial f}{\partial \mathbf{x}} \right] \hat{\Sigma}_{x_m} \mathbb{E} \left[\frac{\partial f}{\partial \mathbf{x}} \right]^T + \text{tr} \left(\mathbb{V} \left[\frac{\partial f}{\partial \mathbf{x}} \right] \hat{\Sigma}_{x_m} \right). \quad (5.38)$$

This completes the regression method of taking into account noisy measurement points.

5.2.3. SPARSE AND ONLINE ALGORITHMS – THE MAIN IDEAS

The next question we will ask ourselves is, ‘Can we also set up a sparse and an online regression algorithm which takes into account noisy measurement points?’ For sparse algorithms (discussed in Section 4.1) the answer is a clear ‘yes’. In the NIGP algorithm we just described, we can simply use any kind of sparse GP regression method instead of regular GP regression. For online algorithms (discussed in Section 4.2) the answer is a lot more complex. So let’s take some time to figure this out.

Suppose that we are applying online FITC regression. For this algorithm, we have a set of inducing input points X_u , which is of course fully deterministic, since we have chosen these points ourselves. We also have already incorporated n_m measurements, which results in a posterior distribution $\mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu})$ of \underline{f}_u . But now we get a new measurement, which we write as $(\hat{\mathbf{x}}_+, \hat{f}_+)$. Here $\hat{\mathbf{x}}_+$ is the actual measured input and \hat{f}_+ is the actual measured function value. However, these are subject to noise. As a result, the measurement input point is actually a random variable $\underline{\mathbf{x}}_+$, and its distribution (as indicated by the measurement) is given by $\mathcal{N}(\hat{\mathbf{x}}_+, \hat{\Sigma}_{x_+})$. Similarly, the measured function value is a random variable \underline{f}_+ with prior distribution $\mathcal{N}(\hat{f}_+, \hat{\sigma}_{f_+}^2)$.

⁵In the main paper of McHutchon and Rasmussen (2011), this derivative variance was not taken into account, but in the online tools accompanying the paper it was. In the documentation of these tools, they also indicated that incorporating this derivative variance had a nearly negligible effect on the final result. My own experiments verified this.

If our measurement input point \underline{x}_+ would not be subject to noise, we can use our usual update law for the distribution of the inducing function values \underline{f}_u . We write this new distribution, which depends on \underline{x}_+ , as $\underline{f}_u(\underline{x}_+) \sim \mathcal{N}(\boldsymbol{\mu}_u^+(\underline{x}_+), \Sigma_{uu}^+(\underline{x}_+))$. The superscript $+$ again indicates that the new measurement has been taken into account. The distribution is given (through combining (4.32) and (4.33)) by

$$\begin{aligned}\underline{f}_u(\underline{x}_+) &\sim \mathcal{N}(\boldsymbol{\mu}_u^+(\underline{x}_+), \Sigma_{uu}^+(\underline{x}_+)) \\ \Sigma_{uu}^+(\underline{x}_+) &= \Sigma_{uu} - \Sigma_{uu} K_{uu}^{-1} K_{u+} \hat{\Sigma}_{++}^{-1} K_{u+} K_{uu}^{-1} \Sigma_{uu}, \\ \boldsymbol{\mu}_u^+(\underline{x}_+) &= \boldsymbol{\mu}_u + \Sigma_{uu} K_{uu}^{-1} K_{u+} \hat{\Sigma}_{++}^{-1} (\hat{f}_+ - \hat{\mu}_+),\end{aligned}\tag{5.39}$$

where we have defined the prior distribution of the measured value \hat{f}_+ for a given input point \underline{x}_+ (based on the current inducing function value distribution \underline{f}_u) as

$$\begin{aligned}\hat{f}_+ &\sim \mathcal{N}(\hat{\mu}_+, \hat{\Sigma}_{++}), \\ \hat{\Sigma}_{++} &= K_{++} + \hat{\sigma}_{f_+}^2 - K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}) K_{uu}^{-1} K_{u+}, \\ \hat{\mu}_+ &= m_+ + K_{+u} K_{uu}^{-1} (\boldsymbol{\mu}_u - \boldsymbol{m}_u).\end{aligned}\tag{5.40}$$

5

Note that a *lot* of parameters, like $\hat{\Sigma}_{++}$, $\hat{\mu}_+$, K_{++} , K_{u+} and m_+ , depend on the value of \underline{x}_+ . Officially we should write them as $\hat{\Sigma}_{++}(\underline{x}_+)$, $\hat{\mu}_+(\underline{x}_+)$, and so on, but that would make the equations rather hard to read.

Naturally, we cannot use the above update law directly here, because \underline{x}_+ is not known precisely. Instead, we will integrate over all possible values of the new measurement input point \underline{x}_+ . This gives us a posterior distribution of \underline{f}_u equal to

$$p(\underline{f}_u | \hat{f}_+, \underline{f}_u) = \int_X p(\underline{f}_u^+ | \underline{x}_+, \hat{f}_+, \underline{f}_u) p(\underline{x}_+ | \hat{f}_+, \underline{f}_u) d\underline{x}_+.\tag{5.41}$$

In this expression it is important to note that all distributions are based on the previous distribution of \underline{f}_u (expressed through $\boldsymbol{\mu}_u$ and Σ_{uu}) as well as on the new measurement \hat{f}_+ . So the distribution $p(\underline{f}_u | \underline{x}_+, \hat{f}_+, \underline{f}_u)$ indicates the distribution of \underline{f}_u given an exact value of \underline{x}_+ , given the measurement \hat{f}_+ and given the previous distribution of \underline{f}_u . This hence follows from our update law (5.39).

However, the more interesting term is $p(\underline{x}_+ | \hat{f}_+, \underline{f}_u)$. This is *not* the prior distribution $\mathcal{N}(\hat{\underline{x}}_+, \hat{\Sigma}_{x_+})$ of \underline{x}_+ . Instead, it is the posterior distribution of \underline{x}_+ based on our new measurement \hat{f}_+ and on the current Gaussian process that we have, indicated through the current distribution $\mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu})$ of \underline{f}_u . Let's see how we can find or approximate it.

5.2.4. THE POSTERIOR DISTRIBUTION OF THE MEASUREMENT POINT

The key to finding $p(\underline{x}_+ | \hat{f}_+, \underline{f}_u)$ is to use Bayes' law as

$$p(\underline{x}_+ | \hat{f}_+, \underline{f}_u) = \frac{p(\hat{f}_+ | \underline{x}_+, \underline{f}_u) p(\underline{x}_+ | \underline{f}_u)}{p(\hat{f}_+ | \underline{f}_u)}.\tag{5.42}$$

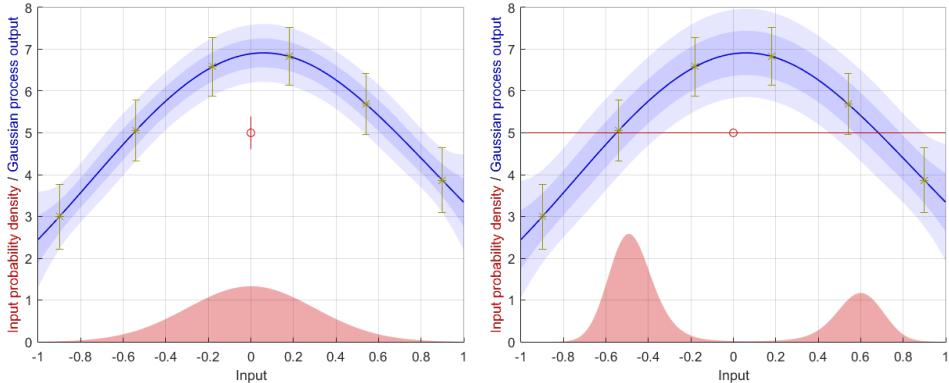


Figure 5.2: An example Gaussian process (left) which is given an additional noisy measurement $(\hat{x}_+, \hat{f}_+) = (0, 5)$. Measurement noise standard deviations are $\hat{\sigma}_{x_+} = 0.3$ and $\hat{\sigma}_{f_+} = 0.4$. The resulting prior distribution $\mathcal{N}(\hat{x}_+, \hat{\sigma}_{x_+}^2)$ of the true measurement input point x_+ is also shown (left bottom). The key to finding the posterior distribution, is to first incorporate the output noise into the GP (right), resulting in the prior probability distribution for the measurement \hat{f}_+ for each possible input point x_+ . Then, when we draw a horizontal line through the measurement, we effectively find the ‘proof’ $p(\hat{f}_+ | x_+, \underline{f}_u)$ that we obtained measurement \hat{f}_+ at x_+ . We multiply this proof by the prior $p(x_+)$ and normalize the result to find the posterior measurement input point distribution (right bottom).

5

Let's discuss the three new terms in this expression. $p(\hat{f}_+ | x_+, \underline{f}_u)$ is the probability (density) that we obtained measurement \hat{f}_+ at the known input point x_+ of our current Gaussian process, described through \underline{f}_u . We have just seen this at (5.40).

For the second probability $p(x_+ | \underline{f}_u)$ it is important to realize that \underline{f}_u itself does not tell us anything about what the value of x_+ is likely to be. As a result, this probability equals the probability we got from our measurement, being $p(x_+) = \mathcal{N}(x_+ | \hat{x}_+, \hat{\Sigma}_{x_+})$. Finally, the third probability $p(\hat{f}_+ | \underline{f}_u)$ is just a constant (not depending on x_+) and we can hence get rid of its effects by normalizing the distribution which we wind up with. The process of finding the posterior distribution of x_+ is visualized in Figure 5.2.

Though the approach so far works, is has one very important drawback. It results in a non-Gaussian distribution for $p(x_+ | \hat{f}_+, \underline{f}_u)$, which makes it very difficult to deal with in the rest of our algorithm. We want to find a Gaussian distribution, and as Figure 5.2 shows, applying moment matching might not always be the best approach, because the distribution may not even resemble anything Gaussian. We will go for a different solution.

Let's denote the prior distribution of the measured value \hat{f}_+ by \hat{f}_+ . Its distribution is described by (5.40). The solution now lies in linearizing this, with respect to some linearization point \bar{x}_+ . In other words, we assume that \hat{f}_+ is now a Gaussian process with a linear mean function and a constant variance. So,

$$\hat{f}_+ \sim p(\hat{f}_+ | x_+, \underline{f}_u) = \mathcal{N}\left(\hat{f}_+ | \hat{\mu}_+(\bar{x}_+) + \frac{\partial \hat{\mu}_+(\bar{x}_+)}{\partial x_+} (x_+ - \bar{x}_+), \hat{\Sigma}_{++}(\bar{x}_+)\right). \quad (5.43)$$

If we use this instead, then the posterior distribution of x_+ will be Gaussian, as is visual-

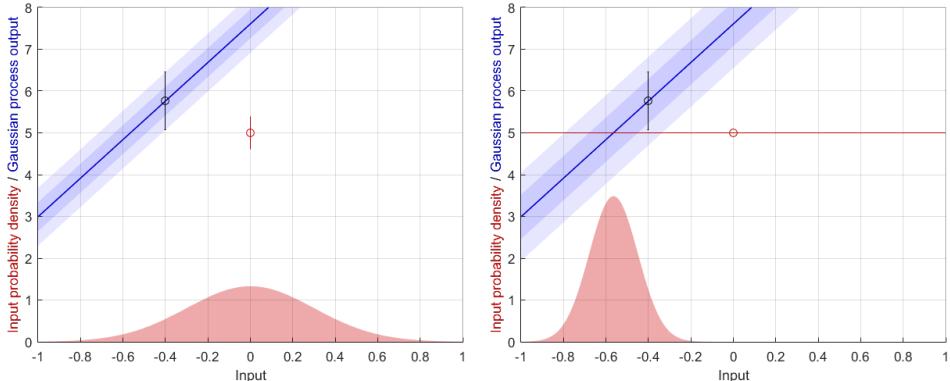


Figure 5.3: The Gaussian process of Figure 5.2, linearized about the point $\bar{x}_+ = -0.4$. The distribution of the corresponding output $f(\bar{x}_+) \sim \mathcal{N}(\hat{\mu}_+(\bar{x}_+), \hat{\Sigma}_{++}(\bar{x}_+))$ (though without noise) is indicated. For this linearized Gaussian process, we apply the same steps. From the linearized Gaussian process (left), we incorporate measurement noise (right), consider the proof $p(\hat{f}_+ | \underline{x}_+, \underline{f}_u)$ (the horizontal line), multiply it by the prior $p(x_+)$ (left bottom) and normalize the result to find the posterior distribution of \underline{x}_+ (right bottom). This result can be found analytically through (5.44). It is important to note here that this result may strongly depend on the linearization point \bar{x}_+ . Usually we reset \bar{x}_+ equal to the posterior mean \hat{x}_+^+ of \underline{x}_+ and reiterate until convergence.

5

ized in Figure 5.3.

There are still two important questions left. The first is how to calculate the posterior distribution of $p(x_+ | \hat{f}_+, \underline{f}_u)$. This is done through (5.42). So we use the linearized version (5.43) of \hat{f}_+ and multiply this by $\mathcal{N}(x_+ | \hat{x}_+, \hat{\Sigma}_{x_+})$. We can obtain the result of this multiplication through Theorem A.17, where we only have to make a few clever substitutions. If we then normalize the final result, we get

$$\begin{aligned} \underline{x}_+ &\sim p(x_+ | \hat{f}_+, \underline{f}_u) = \mathcal{N}(x_+ | \hat{x}_+^+, \hat{\Sigma}_{x_+}^+), \\ \hat{\Sigma}_{x_+}^+ &= \left(\left(\frac{\partial \hat{\mu}_+(\bar{x}_+)}{\partial \underline{x}_+} \right)^T (\hat{\Sigma}_{++}(\bar{x}_+))^{-1} \left(\frac{\partial \hat{\mu}_+(\bar{x}_+)}{\partial \underline{x}_+} \right) + \hat{\Sigma}_{x_+}^{-1} \right)^{-1}, \\ \hat{x}_+^+ &= \hat{x}_+ + \hat{\Sigma}_{x_+}^+ \left(\left(\frac{\partial \hat{\mu}_+(\bar{x}_+)}{\partial \underline{x}_+} \right)^T (\hat{\Sigma}_{++}(\bar{x}_+))^{-1} \left((\hat{f}_+ - \hat{\mu}_+(\bar{x}_+)) - \frac{\partial \hat{\mu}_+(\bar{x}_+)}{\partial \underline{x}_+} (\hat{x}_+ - \bar{x}_+) \right) \right). \end{aligned} \quad (5.44)$$

Again, note that the parameters with the superscript $^+$ are the posterior parameters. So the above is the posterior distribution of the measurement input point \underline{x}_+ . Or at least, our Gaussian approximation of it. It is also interesting to know here that, even if $\hat{\Sigma}_{x_+}$ may be diagonal, $\hat{\Sigma}_{x_+}^+$ generally is not. So there will be posterior correlations between the various elements of \underline{x}_+ .

The second question is which linearization point \bar{x}_+ we should use. Based on (5.44), the easiest point to linearize about would be \hat{x}_+ . However, if we would linearize about \hat{x}_+ in the example of Figure 5.3 (this linearization is not shown), this would result in a very detrimental result, because we would get a nearly flat linearization. A more ideal choice would be to linearize about \hat{x}_+^+ from (5.44), but we do not know this parameter yet. This results in another chicken-and-egg story, which can be solved by doing a few

iterations. That is, we first pick $\hat{\mathbf{x}}_+$ as linearization point. We use this to find $\hat{\mathbf{x}}_+^+$, then use this point as linearization point, find a new value for $\hat{\mathbf{x}}_+^+$ and continue for a few iterations, until $\hat{\mathbf{x}}_+^+$ has converged⁶. This should result in a more representative Gaussian posterior distribution of $\underline{\mathbf{x}}_+$ than we would get by directly applying moment matching.

5.2.5. UPDATING THE DISTRIBUTION OF THE INDUCING FUNCTION VALUES

Let's turn our attention back to integral (5.41). We have assumed that the second probability in this integral is Gaussian. In other words, we say that $\underline{\mathbf{x}}_+$ is a Gaussian parameter with known (posterior) mean $\hat{\mathbf{x}}_+^+$ and covariance matrix $\hat{\Sigma}_{\underline{\mathbf{x}}_+}^+$. This significantly simplifies matters.

However, the next problem we run into is that the resulting posterior distribution for $\underline{f}_{\underline{\mathbf{u}}}$ will still not be Gaussian. To solve this, we will once more apply moment matching, similarly to what we did in Sections 5.1.3 and 5.1.4. This results in a posterior mean (like (5.6)) and covariance (like (5.16)) of

$$\boldsymbol{\mu}_{\underline{\mathbf{u}}}^+ = \int_X \boldsymbol{\mu}_{\underline{\mathbf{u}}}^+(\underline{\mathbf{x}}_+) p(\underline{\mathbf{x}}_+) d\underline{\mathbf{x}}_+, \quad (5.45)$$

$$\Sigma_{uu}^+ = \int_X \left(\Sigma_{uu}^+(\underline{\mathbf{x}}_+) + (\boldsymbol{\mu}_{\underline{\mathbf{u}}}^+(\underline{\mathbf{x}}_+) - \boldsymbol{\mu}_{\underline{\mathbf{u}}}^+) (\boldsymbol{\mu}_{\underline{\mathbf{u}}}^+(\underline{\mathbf{x}}_+) - \boldsymbol{\mu}_{\underline{\mathbf{u}}}^+)^T \right) p(\underline{\mathbf{x}}_+) d\underline{\mathbf{x}}_+. \quad (5.46)$$

The second problem is that solving these integrals is very hard, if not impossible. The reason is the inverse $\hat{\Sigma}_{\underline{\mathbf{x}}_+}^{-1}$ in (5.39). Although it is not a matrix inverse this time but merely a scalar inverse, it is still too difficult to evaluate.

To work around this, we will not use the posterior Gaussian process $\underline{f}_{\underline{\mathbf{u}}}(\underline{\mathbf{x}}_+)$, but instead use a Taylor polynomial approximation of it. This Taylor polynomial will be taken about the posterior mean $\hat{\mathbf{x}}_+^+$ of $\underline{\mathbf{x}}_+$, as given by (5.44). Additionally, we will evaluate the Taylor polynomial element-wise. So for every element of $\underline{f}_{\underline{u}_i}^+(\underline{\mathbf{x}}_+)$ we write

$$\underline{f}_{\underline{u}_i}(\underline{\mathbf{x}}_+) = \underline{f}_{\underline{u}_i}(\hat{\mathbf{x}}_+^+) + \frac{\partial \underline{f}_{\underline{u}_i}(\hat{\mathbf{x}}_+^+)}{\partial \underline{\mathbf{x}}_+} (\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+^+) + \frac{1}{2} (\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+^+)^T \frac{\partial^2 \underline{f}_{\underline{u}_i}(\hat{\mathbf{x}}_+^+)}{\partial \underline{\mathbf{x}}_+^2} (\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+^+) + \dots \quad (5.47)$$

This does give us infinitely many terms though. To manage this, we will make a third approximation (after moment matching and the Taylor polynomial). We will assume that the input noise covariance $\hat{\Sigma}_{\underline{\mathbf{x}}_+}$ is small. More specifically, we assume that is small enough that $\hat{\Sigma}_{\underline{\mathbf{x}}_+}^2$ and higher powers of $\hat{\Sigma}_{\underline{\mathbf{x}}_+}$ are negligible⁷. As a result, because the posterior covariance $\hat{\Sigma}_{\underline{\mathbf{x}}_+}^+$ is guaranteed to be smaller (that is, has a lower determinant) than the prior covariance $\hat{\Sigma}_{\underline{\mathbf{x}}_+}$, the same holds for $\hat{\Sigma}_{\underline{\mathbf{x}}_+}^+$. So if we ever encounter a term in any of our derivations with four or more factors of $(\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+^+)$, we know it will eventually drop out of our equations. In addition, because $\underline{\mathbf{x}}_+$ is (assumed) Gaussian, also terms with

⁶It is worthwhile to note here that this set-up is not guaranteed to converge. When it does not, the default option is to just ignore the measurement altogether. Finding a better way of dealing with this is left as a suggestion for further research.

⁷The idea of using Taylor polynomials of Gaussian process is not new. Girard and Murray-Smith (2003) made the assumption that higher order derivatives (like $\partial^2 \underline{f}_{\underline{u}_i}(\hat{\mathbf{x}}_+^+)/\partial \underline{\mathbf{x}}_+^2$) are negligible. We instead assume that higher powers of $\hat{\Sigma}_{\underline{\mathbf{x}}_+}$ are negligible. The effect is nearly (but not entirely) the same, while our assumption is easier to justify/verify.

three (or any odd number) of such factors will drop out. This significantly simplifies all our expressions. We only have to keep track of terms with at most two factors ($\mathbf{x}_+ - \hat{\mathbf{x}}_+^+$).

The next step is to find the corresponding approximated mean $\mu_{u_i}^+(\mathbf{x}_+)$ and covariance $\Sigma_{u_i u_j}^+(\mathbf{x}_+)$ of this Taylor polynomial for any value of \mathbf{x}_+ . The approximated mean is given by

$$\begin{aligned}\mu_{u_i}^+(\mathbf{x}_+) &= \mathbb{E} \left[f_{-u_i}(\mathbf{x}_+) \right] \\ &\approx \mu_{u_i}^+(\hat{\mathbf{x}}_+^+) + \frac{\partial \mu_{u_i}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+} (\mathbf{x}_+ - \hat{\mathbf{x}}_+^+) + \frac{1}{2} (\mathbf{x}_+ - \hat{\mathbf{x}}_+^+)^T \frac{\partial^2 \mu_{u_i}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+^2} (\mathbf{x}_+ - \hat{\mathbf{x}}_+^+) + \dots,\end{aligned}\quad (5.48)$$

where the dots denote terms which will eventually disappear. We can get a similar expression for the approximated covariance. The actual derivation is a lot more involved but, not very surprisingly, the outcome is

$$\begin{aligned}\Sigma_{u_i u_j}^+(\mathbf{x}_+) &= \mathbb{E} \left[\left(f_{-u_i}(\mathbf{x}_+) - \mu_{u_i}^+(\mathbf{x}_+) \right) \left(f_{-u_j}(\mathbf{x}_+) - \mu_{u_j}^+(\mathbf{x}_+) \right)^T \right] \\ &\approx \Sigma_{u_i u_j}^+(\hat{\mathbf{x}}_+^+) + \frac{\partial \Sigma_{u_i u_j}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+} (\mathbf{x}_+ - \hat{\mathbf{x}}_+^+) + \frac{1}{2} (\mathbf{x}_+ - \hat{\mathbf{x}}_+^+)^T \frac{\partial^2 \Sigma_{u_i u_j}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+^2} (\mathbf{x}_+ - \hat{\mathbf{x}}_+^+).\end{aligned}\quad (5.49)$$

5

Through these Taylor approximations, we now have a mean and covariance given by (5.48) and (5.49). We can plug these into (5.45) and (5.46) to find the posterior distribution $\mathcal{N}(\boldsymbol{\mu}_u^+, \Sigma_{uu}^+)$ of \underline{f}_u , which is what this is all about.

Note that we do still need to solve the integrals of (5.45) and (5.46). Luckily, because (5.48) and (5.49) are either linear or quadratic in the integrating parameter \mathbf{x}_+ , and because \mathbf{x}_+ is taken from a Gaussian distribution, these integrals can be solved directly. (Their solutions follow from (B.62) to (B.64).) We then find the solution⁸

$$\mu_{u_i}^+ = \mu_{u_i}^+(\hat{\mathbf{x}}_+^+) + \frac{1}{2} \text{tr} \left(\left(\frac{\partial^2 \mu_{u_i}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+^2} \right) \hat{\Sigma}_{x_+}^+ \right), \quad (5.50)$$

$$\Sigma_{u_i u_j}^+ = \Sigma_{u_i u_j}^+(\hat{\mathbf{x}}_+^+) + \left(\frac{\partial \mu_{u_i}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+} \right) \hat{\Sigma}_{x_+}^+ \left(\frac{\partial \mu_{u_j}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+} \right)^T + \frac{1}{2} \text{tr} \left(\left(\frac{\partial^2 \Sigma_{u_i u_j}^+(\hat{\mathbf{x}}_+^+)}{\partial \mathbf{x}_+^2} \right) \hat{\Sigma}_{x_+}^+ \right). \quad (5.51)$$

These are the update laws for the so-called *Sparse Online Noisy Input GP* (SONIG) regression algorithm.

It is rather difficult to implement all the SONIG update laws without making any programming mistakes. It will require lots of frustrating debugging. To make this process easier, a toolbox with all the SONIG functionalities is available online through [Bijl \(2016b\)](#), although all the code is also available (as usual) through [Bijl \(2016a\)](#).

⁸It is interesting to note that in the NIGP method by [McHutchon and Rasmussen \(2011\)](#), which we looked at in Section 5.2.2, we already used (5.51). However, we did not use (5.50). The last term from this expression was missing. So by incorporating this second derivative of the mean function, we could potentially improve the NIGP method. In fact, later on in Section 5.4.1 we will see that the SONIG algorithm works better than the NIGP algorithm, which is mainly because it takes into account this second derivative.

5.2.6. DERIVATIVES NEEDED FOR THE SONIG ALGORITHM

To apply the SONIG update laws which we just derived, we need to calculate a lot of derivatives. These derivatives all follow from (5.39), but they may still be daunting to calculate, especially when the input \mathbf{x}_+ is a vector instead of just a scalar. So let's make an overview of all the derivatives that we need. You can implement them yourself, but it might be easier to download the code from [Bijl \(2016a\)](#) and use that.

We will start with the derivatives of $\boldsymbol{\mu}_u^+(\mathbf{x}_+)$ with respect to \mathbf{x}_+ . To be precise, we will take the derivatives with respect to a single element x_{+i} of \mathbf{x}_+ , so we can find the derivatives element-wise. These derivatives equal

$$\begin{aligned} \frac{\partial \boldsymbol{\mu}_u^+(\mathbf{x}_+)}{\partial x_{+i}} &= \Sigma_{uu} K_{uu}^{-1} \left(\frac{\partial K_{u+}}{\partial x_{+i}} \hat{\Sigma}_{++}^{-1} \hat{\mu}_+ + K_{u+} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+i}} \hat{\mu}_+ - K_{u+} \hat{\Sigma}_{++}^{-1} \frac{\partial \hat{\mu}_+}{\partial \mathbf{x}_+} \right), \\ \frac{\partial^2 \boldsymbol{\mu}_u^+(\mathbf{x}_+)}{\partial x_{+i} \partial x_{+j}} &= \Sigma_{uu} K_{uu}^{-1} \left(\frac{\partial^2 K_{u+}}{\partial x_{+i} \partial x_{+j}} \hat{\Sigma}_{++}^{-1} \hat{\mu}_+ + \frac{\partial K_{u+}}{\partial x_{+i}} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+j}} \hat{\mu}_+ + \frac{\partial K_{u+}}{\partial x_{+i}} \hat{\Sigma}_{++}^{-1} \frac{\partial \hat{\mu}_+}{\partial x_{+j}} \right. \\ &\quad + \frac{\partial K_{u+}}{\partial x_{+j}} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+i}} \hat{\mu}_+ + K_{u+} \frac{\partial^2 \hat{\Sigma}_{++}^{-1}}{\partial x_{+i} \partial x_{+j}} \hat{\mu}_+ + K_{u+} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+i}} \frac{\partial \hat{\mu}_+}{\partial x_{+j}} \\ &\quad \left. + \frac{\partial K_{u+}}{\partial x_{+j}} \hat{\Sigma}_{++}^{-1} \frac{\partial \hat{\mu}_+}{\partial x_{+i}} + K_{u+} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+j}} \frac{\partial \hat{\mu}_+}{\partial x_{+i}} + K_{u+} \hat{\Sigma}_{++}^{-1} \frac{\partial^2 \hat{\mu}_+}{\partial x_{+i} \partial x_{+j}} \right). \end{aligned} \quad (5.52)$$

Next, we look at the derivatives of $\Sigma_{uu}^+(\mathbf{x}_+)$. These are similarly given by

$$\begin{aligned} \frac{\partial \Sigma_{uu}^+(\mathbf{x}_+)}{\partial x_{+i}} &= \Sigma_{uu} K_{uu}^{-1} \left(\frac{\partial K_{u+}}{\partial x_{+i}} \hat{\Sigma}_{++}^{-1} K_{+u} + K_{u+} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+i}} K_{+u} + K_{u+} \hat{\Sigma}_{++}^{-1} \frac{\partial K_{+u}}{\partial x_{+i}} \right) K_{uu}^{-1} \Sigma_{uu}, \\ \frac{\partial^2 \Sigma_{uu}^+(\mathbf{x}_+)}{\partial x_{+i} \partial x_{+j}} &= \Sigma_{uu} K_{uu}^{-1} \left(\frac{\partial^2 K_{u+}}{\partial x_{+i} \partial x_{+j}} \hat{\Sigma}_{++}^{-1} K_{+u} + \frac{\partial K_{u+}}{\partial x_{+i}} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+j}} K_{+u} + \frac{\partial K_{u+}}{\partial x_{+i}} \hat{\Sigma}_{++}^{-1} \frac{\partial K_{+u}}{\partial x_{+j}} \right. \\ &\quad + \frac{\partial K_{u+}}{\partial x_{+i}} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+i}} K_{+u} + K_{u+} \frac{\partial^2 \hat{\Sigma}_{++}^{-1}}{\partial x_{+i} \partial x_{+j}} K_{+u} + K_{u+} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+i}} \frac{\partial K_{+u}}{\partial x_{+i}} \\ &\quad \left. + \frac{\partial K_{u+}}{\partial x_{+i}} \hat{\Sigma}_{++}^{-1} \frac{\partial K_{+u}}{\partial x_{+i}} + K_{u+} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial x_{+i}} \frac{\partial K_{+u}}{\partial x_{+i}} + K_{u+} \hat{\Sigma}_{++}^{-1} \frac{\partial^2 K_{+u}}{\partial x_{+i} \partial x_{+j}} \right). \end{aligned} \quad (5.53)$$

Note that a derivative like $\partial \Sigma_{uu}^+(\mathbf{x}_+)/\partial \mathbf{x}_+^2$ is actually a four-dimensional matrix. $\Sigma_{uu}^+(\mathbf{x}_+)$ already has two dimensions, and the second derivative with respect to the vector \mathbf{x}_+ results in two more. When setting up equations involving these parameters, it is wise to learn how to work with higher-dimensional matrices, and to keep track of which index represents what. Feel free to check out the code at [Bijl \(2016a\)](#) to see how I set this up.

In the above derivatives, we have used the derivatives of the inverse $\hat{\Sigma}_{++}^{-1}$. Note that this parameter is a scalar, so we do not even need Theorem A.2 to find this derivative. Instead, it directly follows that

$$\begin{aligned} \frac{\partial \hat{\Sigma}_{++}^{-1}}{\partial \mathbf{x}_+} &= -\hat{\Sigma}_{++}^{-2} \frac{\partial \hat{\Sigma}_{++}}{\partial \mathbf{x}_+}, \\ \frac{\partial^2 \hat{\Sigma}_{++}^{-1}}{\partial \mathbf{x}_+^2} &= 2\hat{\Sigma}_{++}^{-3} \left(\frac{\partial \hat{\Sigma}_{++}}{\partial \mathbf{x}_+} \right)^T \left(\frac{\partial \hat{\Sigma}_{++}}{\partial \mathbf{x}_+} \right) - \hat{\Sigma}_{++}^{-2} \left(\frac{\partial^2 \hat{\Sigma}_{++}}{\partial \mathbf{x}_+^2} \right). \end{aligned} \quad (5.54)$$

In these derivatives, we have in turn used the derivatives of $\hat{\Sigma}_{++}$. These can be found through the definition of $\hat{\Sigma}_{++}$ in (5.40) as

$$\begin{aligned}\frac{\partial \hat{\Sigma}_{++}}{\partial x_{+i}} &= \frac{\partial K_{++}}{\partial \mathbf{x}_+} - 2K_{+u}K_{uu}^{-1}(K_{uu} - \Sigma_{uu})K_{uu}^{-1}\frac{\partial K_{u+}}{\partial x_{+i}}, \\ \frac{\partial^2 \hat{\Sigma}_{++}}{\partial x_{+i} \partial x_{+j}} &= \frac{\partial^2 K_{++}}{\partial \mathbf{x}_+^2} - 2\frac{\partial K_{+u}}{\partial x_{+j}}K_{uu}^{-1}(K_{uu} - \Sigma_{uu})K_{uu}^{-1}\frac{\partial K_{u+}}{\partial x_{+i}} \\ &\quad - 2K_{+u}K_{uu}^{-1}(K_{uu} - \Sigma_{uu})K_{uu}^{-1}\frac{\partial^2 K_{u+}}{\partial x_{+i} \partial x_{+j}}.\end{aligned}\quad (5.55)$$

Similarly, we can find the derivatives of $\hat{\mu}_+$ as

$$\begin{aligned}\frac{\partial \hat{\mu}_+}{\partial \mathbf{x}_+} &= \frac{\partial m_+}{\partial x_{+i}} + \frac{\partial K_{+u}}{\partial x_{+i}}K_{uu}^{-1}(\boldsymbol{\mu}_u - \mathbf{m}_u), \\ \frac{\partial^2 \hat{\mu}_+}{\partial \mathbf{x}_+^2} &= \frac{\partial^2 m_+}{\partial x_{+i} \partial x_{+j}} + \frac{\partial K_{+u}}{\partial x_{+i} \partial x_{+j}}K_{uu}^{-1}(\boldsymbol{\mu}_u - \mathbf{m}_u).\end{aligned}\quad (5.56)$$

5

The above expressions are so far all valid for any mean and covariance function. They do contain various additional derivatives though, like $\partial K_{u+}/\partial x_{+i}$, and if we want to find these derivatives, then we do have to choose a specific mean and covariance function. We will use the zero mean function and the squared exponential covariance function, as usual. This results in derivatives of K_{u+} which can be found element-wise through

$$\frac{\partial K_{u_i+}}{\partial \mathbf{x}_+} = \lambda_f^2 \exp\left(-\frac{1}{2}(\mathbf{x}_{u_i} - \mathbf{x}_+)^T \Lambda_x^{-1}(\mathbf{x}_{u_i} - \mathbf{x}_+)\right)(\mathbf{x}_{u_i} - \mathbf{x}_+)^T \Lambda^{-1}, \quad (5.57)$$

$$\frac{\partial^2 K_{u_i+}}{\partial \mathbf{x}_+^2} = \lambda_f^2 \exp\left(-\frac{1}{2}(\mathbf{x}_{u_i} - \mathbf{x}_+)^T \Lambda_x^{-1}(\mathbf{x}_{u_i} - \mathbf{x}_+)\right)\left(\Lambda^{-1}(\mathbf{x}_{u_i} - \mathbf{x}_+)(\mathbf{x}_{u_i} - \mathbf{x}_+)^T \Lambda^{-1} - \Lambda^{-1}\right).$$

Since $K_{+u_i} = K_{u_i+}$, also the derivative of K_{+u_i} can be found through the above relations. Finally, the last two sets of derivatives which we are missing are trivial. For the zero mean function and the squared exponential covariance function, we have

$$\frac{\partial m_+}{\partial x_{+i}} = \frac{\partial^2 m_+}{\partial x_{+i} \partial x_{+j}} = 0, \quad (5.58)$$

$$\frac{\partial K_{++}}{\partial x_{+i}} = \frac{\partial^2 K_{++}}{\partial x_{+i} \partial x_{+j}} = 0. \quad (5.59)$$

Those were all the derivatives you should need for the SONIG method. Again, if you want to apply them, you may of course implement all these derivatives yourself. But using the code from [Bijl \(2016a\)](#) might save you a lot of time and frustration.

5.3. EXTENSIONS TO THE SONIG ALGORITHM

We have found a way to implement noisy measurement points in an online way to sparse Gaussian process regression. But the resulting SONIG algorithm has a variety of potential expansions. We will look at a few of them.

First we examine whether it is possible to implement hyperparameters, both for noisy trial and noisy measurement points (Section 5.3.1). Next, we look at how to approximate functions with multiple outputs (Section 5.3.2). We then examine if we can also determine the posterior distribution of the measured function value \underline{f}_+ (Section 5.3.3) as well as its posterior covariance with the measured input \underline{x}_+ (Section 5.3.4). Finally we combine all ideas to set up a system identification algorithm making good use of all the possibilities of the SONIG algorithm (Section 5.3.5).

5.3.1. APPLYING HYPERPARAMETER TUNING

When we are dealing with noisy input points, is it possible to tune the hyperparameters?

If we are dealing with noisy trial points, like we examined in Section 5.1, there is no problem at all. After all, for tuning the hyperparameters, we do not need the trial points at all! Things are different when we have noisy measurement points though.

In this case, it is possible to add hyperparameter tuning to the NIGP algorithm of Section 5.2.2. In this algorithm we are already doing various iterations of predicting a Gaussian process, increasing the noise covariance of each measured output based on the derivative of the previous Gaussian process. After this step, we just need to add a hyperparameter tuning step, and then everything will work out. This does mean we need to tune the hyperparameters multiple times, but the algorithm is supposed to converge relatively quickly, so it is not expected that the hyperparameters change much between successive iterations.

The next question is, ‘Is it also possible to tune the hyperparameters for the SONIG algorithm?’ Sadly, the answer here is no, because it already was not possible to tune the hyperparameters during the online FITC algorithm. The main way to work around this is to grab the first couple of measurements (say, the first few hundred, depending on the problem complexity) and apply the offline NIGP hyperparameter tuning methods to this. This is known as the *subset of data* approach to hyperparameter tuning, and according to Chalupka et al. (2013) its results are nearly as good as when we apply hyperparameter tuning to the full data set of possible thousands or tens of thousands of data points.

5.3.2. USING MULTIPLE OUTPUTS

Suppose that we want to approximate a vector function $\mathbf{f}(\mathbf{x})$ with d_x inputs and d_f outputs. We have looked into ways to deal with this in Section 2.4.2, and examined a simplification – just using a separate GP for every output – in Section 2.4.3. We will also consider this same simplification.

So we will set up a separate Gaussian process for each output $f_i(\mathbf{x})$ of the vector function $\mathbf{f}(\mathbf{x})$. Each Gaussian process may have its own mean function $m^i(\mathbf{x})$ and covariance function $k^i(\mathbf{x}, \mathbf{x}')$. It may also have its own set of inducing input points X_u^i . (In practice this is usually the same for each output $f_i(\mathbf{x})$, but it does not have to be.) And there is a corresponding inducing function value distribution $\underline{f}_u^i \sim \mathcal{N}(\boldsymbol{\mu}_u^i, \Sigma_{uu}^i)$.

Now, what is different in the SONIG algorithm? We will see that there are only minor changes. To see exactly what is different, we will just walk through the steps. First of all, the prior distribution of the measured value $\hat{\underline{f}}_+$ is now a vector. Its distribution, similarly

to (5.40), is given element-wise by

$$\begin{aligned}\hat{f}_+^i &\sim \mathcal{N}\left(\hat{\mu}_+^i, \hat{\Sigma}_{++}^i\right), \\ \hat{\Sigma}_{++}^i &= K_{++}^i + \hat{\sigma}_{f_+^i}^2 - K_{+u}^i \left(K_{uu}^i\right)^{-1} \left(K_{uu}^i - \Sigma_{uu}^i\right) \left(K_{uu}^i\right)^{-1} K_{u+}^i, \\ \hat{\mu}_+^i &= m_+^i + K_{+u}^i \left(K_{uu}^i\right)^{-1} \left(\boldsymbol{\mu}_u^i - \boldsymbol{m}_u^i\right).\end{aligned}\quad (5.60)$$

Here, $\hat{\Sigma}_{++}$ is a diagonal matrix, because we have assumed that all outputs are independent. So its off-diagonal elements are zero.

Next, we use the above distribution to find the posterior distribution of the measurement input point \underline{x}_+ . This is, very similarly to (5.44), equal to

$$\begin{aligned}\underline{x}_+ &\sim \mathcal{N}\left(\underline{x}_+ | \hat{\boldsymbol{x}}_+^+, \hat{\Sigma}_{x_+}^+\right), \\ \hat{\Sigma}_{x_+}^+ &= \left(\left(\frac{\partial \hat{\boldsymbol{\mu}}_+(\bar{\boldsymbol{x}}_+) }{\partial \underline{x}_+} \right)^T \left(\hat{\Sigma}_{++}(\bar{\boldsymbol{x}}_+) \right)^{-1} \left(\frac{\partial \hat{\boldsymbol{\mu}}_+(\bar{\boldsymbol{x}}_+) }{\partial \underline{x}_+} \right) + \hat{\Sigma}_{x_+}^{-1} \right)^{-1}, \\ \hat{\boldsymbol{x}}_+^+ &= \hat{\boldsymbol{x}}_+ + \hat{\Sigma}_{x_+}^+ \left(\left(\frac{\partial \hat{\boldsymbol{\mu}}_+(\bar{\boldsymbol{x}}_+) }{\partial \underline{x}_+} \right)^T \left(\hat{\Sigma}_{++}(\bar{\boldsymbol{x}}_+) \right)^{-1} \left(\left(\hat{f}_+ - \hat{\boldsymbol{\mu}}_+(\bar{\boldsymbol{x}}_+) \right) - \frac{\partial \hat{\boldsymbol{\mu}}_+(\bar{\boldsymbol{x}}_+)}{\partial \underline{x}_+} (\hat{\boldsymbol{x}}_+ - \bar{\boldsymbol{x}}_+) \right) \right).\end{aligned}\quad (5.61)$$

Note that $\hat{\boldsymbol{\mu}}_+(\underline{x}_+)$ has become a vector, and $\hat{\Sigma}_{++}(\underline{x}_+)$ is now a matrix instead of a scalar. These functions are given by (5.60).

After this, the rest of the algorithm is exactly the same. That is, we can still use the SONIG update laws (5.50) and (5.51), although we need to do so separately for each Gaussian process that we have set up.

5.3.3. THE POSTERIOR DISTRIBUTION OF THE MEASURED OUTPUT

We know how to find (or approximate) the posterior distribution of both \underline{x}_+ and \underline{f}_u . But what is the posterior distribution of the function value \underline{f}_+^+ ? This is an interesting question because, although we may know the posterior distribution of \underline{x}_+ , we do not know the position of this input point precisely. Can we then still say something about the posterior distribution of the measured function value?

The key to solving this is again by integrating over all possible values of \underline{x}_+ . That is,

$$p(f_+ | \hat{f}_+, \underline{f}_u) = \int_X p(f_+ | \underline{x}_+, \hat{f}_+, \underline{f}_u) p(\underline{x}_+ | \hat{f}_+, \underline{f}_u) d\underline{x}_+. \quad (5.62)$$

This is exactly the same type of integral as we encountered at (5.41). The second probability in the integral is again the posterior distribution of \underline{x}_+ . The first probability is slightly different though. It is not the distribution of \underline{f}_u for a given input point \underline{x}_+ , but the distribution of the function value \underline{f}_+^+ . So let's find that first.

First of all, we can establish that the prior distribution of \underline{f}_+^+ for a given input point

\underline{x}_+ , before we obtain the measurement \hat{f}_+ , follows from (4.32) as⁹

$$\begin{aligned}\underline{f}_+(\mathbf{x}_+) &\sim \mathcal{N}(\mu_+(\mathbf{x}_+), \Sigma_{++}(\mathbf{x}_+)), \\ \Sigma_{++}(\mathbf{x}_+) &= K_{++} - K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}) K_{uu}^{-1} K_{u+}, \\ \mu_+(\mathbf{x}_+) &= m_+ + K_{+u} K_{uu}^{-1} (\boldsymbol{\mu}_u - \mathbf{m}_u).\end{aligned}\quad (5.63)$$

But of course we do have a measurement \hat{f}_+ now. We can incorporate this using (4.33), which tells us that

$$\begin{aligned}\underline{f}_+(\mathbf{x}_+) &\sim \mathcal{N}(\mu_+(\mathbf{x}_+), \Sigma_{++}^+(\mathbf{x}_+)), \\ \Sigma_{++}^+(\mathbf{x}_+) &= \Sigma_{++} \left(\Sigma_{++} + \hat{\sigma}_{f_+}^2 \right)^{-1} \hat{\sigma}_{f_+}^2 = \Sigma_{++} \hat{\Sigma}_{++}^{-1} \hat{\sigma}_{f_+}^2, \\ \mu_+(\mathbf{x}_+) &= \mu_+ + \Sigma_{++} \left(\Sigma_{++} + \hat{\sigma}_{f_+}^2 \right)^{-1} (\hat{f}_+ - \mu_+) = \mu_+ + \Sigma_{++} \hat{\Sigma}_{++}^{-1} (\hat{f}_+ - \mu_+).\end{aligned}\quad (5.64)$$

Note that, in the above expressions, we have omitted the dependency on \mathbf{x}_+ , but naturally it still exists.

From here on, we can take exactly the same path as before. That is, we approximate $\underline{f}_+(\mathbf{x}_+)$ through a Taylor polynomial

$$\underline{f}_+(\mathbf{x}_+) = \underline{f}_+(\hat{\mathbf{x}}_+) + \frac{\partial \underline{f}_+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+} (\mathbf{x}_+ - \hat{\mathbf{x}}_+) + \frac{1}{2} (\mathbf{x}_+ - \hat{\mathbf{x}}_+)^T \frac{\partial^2 \underline{f}_+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+^2} (\mathbf{x}_+ - \hat{\mathbf{x}}_+) + \dots, \quad (5.65)$$

and we use this to evaluate (5.62). The result will, identically to the SONIG update laws (5.50) and (5.51), be¹⁰

$$\mu_+^+ = \mu_+(\hat{\mathbf{x}}_+) + \frac{1}{2} \text{tr} \left(\left(\frac{\partial^2 \mu_+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+^2} \right) \hat{\Sigma}_{x_+}^+ \right), \quad (5.66)$$

$$\Sigma_{++}^+ = \Sigma_{++}^+(\hat{\mathbf{x}}_+) + \left(\frac{\partial \mu_+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+} \right) \hat{\Sigma}_{x_+}^+ \left(\frac{\partial \mu_+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+} \right)^T + \frac{1}{2} \text{tr} \left(\left(\frac{\partial^2 \Sigma_{++}^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+^2} \right) \hat{\Sigma}_{x_+}^+ \right). \quad (5.67)$$

These expressions derive the posterior distribution of \underline{f}_+ for single-output functions. Naturally, we can also expand these ideas to multi-output functions, as those considered in Section 5.3.2. In this case μ_+^+ is a vector and Σ_{++}^+ is a matrix. If we denote individual elements according to μ_{+i}^+ and Σ_{+i+j}^+ , then the above expressions turn into

$$\mu_{+i}^+ = \mu_{+i}^+(\hat{\mathbf{x}}_+) + \frac{1}{2} \text{tr} \left(\left(\frac{\partial^2 \mu_{+i}^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+^2} \right) \hat{\Sigma}_{x_+}^+ \right), \quad (5.68)$$

$$\Sigma_{+i+j}^+ = \Sigma_{+i+j}^+(\hat{\mathbf{x}}_+) + \left(\frac{\partial \mu_{+i}^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+} \right) \hat{\Sigma}_{x_+}^+ \left(\frac{\partial \mu_{+j}^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+} \right)^T + \frac{1}{2} \text{tr} \left(\left(\frac{\partial^2 \Sigma_{+i+j}^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+^2} \right) \hat{\Sigma}_{x_+}^+ \right). \quad (5.69)$$

⁹The main difference between this prior distribution and the prior distribution of \hat{f}_+ , given by (5.40), is that this is the prior distribution of the *actual* function value $f(\mathbf{x})$, and not that of the *measured* function value \hat{f}_+ . This is why the noise variance $\hat{\sigma}_{f_+}^2$ is not taken into account in this new distribution.

¹⁰Once more, keep in mind the distinction between a parameter like $\Sigma_{++}^+(\mathbf{x}_+)$, which holds for a given value of \mathbf{x}_+ , and a parameter like Σ_{++}^+ , which is the result of integrating over all possible values of \mathbf{x}_+ .

It is important to realize here that the matrix $\Sigma_{++}(\mathbf{x}_+)$, for a given input point \mathbf{x}_+ , is actually diagonal by assumption, and hence the same holds for $\Sigma_{++}^+(\mathbf{x}_+)$. This will simplify the above expression for off-diagonal terms, but it does not imply that Σ_{++}^+ itself will be diagonal.

5.3.4. THE POSTERIOR COVARIANCE BETWEEN INPUT AND OUTPUT

We now have a posterior distribution $\underline{\mathbf{x}}_+ \sim \mathcal{N}(\hat{\mathbf{x}}_+, \hat{\Sigma}_{\mathbf{x}_+}^+)$ for the input and a posterior distribution $\underline{f}_+ \sim \mathcal{N}(\mu_+^+, \Sigma_{++}^+)$ for the output. Since these are now both random variables, they naturally also have a covariance.

To find this covariance, we should use both the Taylor approximation (5.65) of $f_+(\mathbf{x}_+)$ and relation (5.66) for μ_+^+ . The covariance then follows as

$$\begin{aligned} \mathbb{V}[f_+, \underline{\mathbf{x}}_+] &= \mathbb{E}\left[\left(f_+(\mathbf{x}_+) - \mu_+^+\right)\left(\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+\right)^T\right] \\ &= \mathbb{E}\left[\left(f_+(\hat{\mathbf{x}}_+) - \mu_+^+(\hat{\mathbf{x}}_+)\right)\left(\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+\right)^T - \frac{1}{2}\text{tr}\left(\left(\frac{\partial^2 \mu_+^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+^2}\right)\hat{\Sigma}_{\mathbf{x}_+}^+\right)\left(\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+\right)^T\right. \\ &\quad \left. + \frac{\partial f_+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+}\left(\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+\right)\left(\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+\right)^T + \dots\right] \\ &= \frac{\partial \mu_+^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+}\hat{\Sigma}_{\mathbf{x}_+}^+. \end{aligned} \quad (5.70)$$

5

Note that most terms have dropped out, because taking the expectation of $(\underline{\mathbf{x}}_+ - \hat{\mathbf{x}}_+^+)$ will result in zero. Also, the higher order terms have dropped out due to our assumption that higher powers of $\hat{\Sigma}_{\mathbf{x}_+}^+$ are negligible.

When we deal with multi-output functions, then the above expression can be used individually for every output. In fact, in that case we could also write that

$$\mathbb{V}[f_+, \underline{\mathbf{x}}_+] = \frac{\partial \mu_+^+(\hat{\mathbf{x}}_+)}{\partial \mathbf{x}_+}\hat{\Sigma}_{\mathbf{x}_+}^+. \quad (5.71)$$

5.3.5. AN ONLINE SYSTEM IDENTIFICATION ALGORITHM

Consider a *nonlinear autoregressive model with exogenous inputs* (a NARX system) of the form

$$\mathbf{y}_{k+1} = \phi(\mathbf{y}_k, \mathbf{y}_{k-1}, \mathbf{u}_k), \quad (5.72)$$

with $\phi(\dots)$ a nonlinear function of past outputs and inputs. Suppose that we are running such a system. That is, at the start of every time step we need to specify a certain input \mathbf{u}_k , and at the end of the time step we get a noisy measurement of the output \mathbf{y}_{k+1} , which we can use to determine the next input.

We can use the SONIG algorithm to identify this system. To do so, we set up the function input vector

$$\mathbf{x}_{k+1} = \begin{bmatrix} \mathbf{y}_k \\ \mathbf{y}_{k-1} \\ \mathbf{u}_k \end{bmatrix}. \quad (5.73)$$

We hence have $\mathbf{y}_{k+1} = \phi(\mathbf{x}_{k+1})$. We should note here that all outputs \mathbf{y}_k are actually random variables, and we hence write them as $\underline{\mathbf{y}}_k$. After all, we have only made noisy

measurements \hat{y}_k of them. As a result, also \underline{x}_k is a random variable. We can determine its prior distribution based on the output noise covariance $\hat{\Sigma}_{yy}$, which we assume is known. For example, we have

$$\underline{x}_3 = \begin{bmatrix} \underline{y}_2 \\ \underline{y}_1 \\ \underline{u}_2 \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \hat{y}_2 \\ \hat{y}_1 \\ \underline{u}_2 \end{bmatrix}, \begin{bmatrix} \hat{\Sigma}_{yy} & 0 & 0 \\ 0 & \hat{\Sigma}_{yy} & 0 \\ 0 & 0 & 0 \end{bmatrix} \right). \quad (5.74)$$

Here we assume that there is no prior correlation between measurement noise at different time steps.

Now, suppose that we measure \underline{y}_3 . We write the resulting measurement as \hat{y}_3 . This effectively tells us that \underline{y}_3 is distributed according to

$$\underline{y}_3 \sim \mathcal{N}(\hat{y}_3, \hat{\Sigma}_{yy}). \quad (5.75)$$

But we know that $\underline{y}_3 = \phi(\underline{x}_3)$, so we can plug this into our SONIG algorithm. One result of this is that the SONIG algorithm goes through an update, taking into account this new data. But additionally the SONIG algorithm also gives posterior distributions¹¹ of both \underline{y}_3 and \underline{x}_3 . This means that we get more data about what the true outputs (without being disturbed by noise) would have been. In fact, where \underline{y}_1 initially had a mean value of \hat{y}_1 , it will now have a mean value which we denote by $\hat{\mu}_{y_1}$, and similarly for \underline{y}_2 and \underline{y}_3 .

Note that, because the SONIG algorithm provides us with a posterior distribution of \underline{x}_3 , we also learn more about the covariance $\hat{\Sigma}_{y_1 y_2}$ between \underline{y}_1 and \underline{y}_2 . In addition, the algorithm also tells us (through (5.71)) the covariance between the function input \underline{x}_3 and the function output \underline{y}_3 . From this we learn the posterior covariances $\hat{\Sigma}_{y_1 y_3}$ and $\hat{\Sigma}_{y_2 y_3}$.

Next, let's consider the next time step. For this time step, we need to determine the prior distribution of the function input \underline{x}_4 . Using all our data, it equals

$$\underline{x}_4 = \begin{bmatrix} \underline{y}_3 \\ \underline{y}_2 \\ \underline{u}_3 \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \hat{\mu}_{y_3} \\ \hat{\mu}_{y_2} \\ \underline{u}_3 \end{bmatrix}, \begin{bmatrix} \hat{\Sigma}_{y_3 y_3} & \hat{\Sigma}_{y_3 y_2} & 0 \\ \hat{\Sigma}_{y_2 y_3} & \hat{\Sigma}_{y_2 y_2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \right). \quad (5.76)$$

So when applying the SONIG algorithm, we should use these parameters for the input distribution $\mathcal{N}(\hat{\mathbf{x}}_+, \hat{\Sigma}_{x_+})$. We can then measure \underline{y}_4 , apply another update, and continue the process like that. By applying all these steps consecutively, always keeping track of the covariances between successive outputs \underline{y}_k , we can identify the system.

Note that so far we have assumed that we know the inputs \underline{u}_k precisely. If there is, due to whatever reason, also noise present in the input, this can of course also be taken into account.

Let's formalize all our thoughts a bit more. We consider the more general NARX system

$$\underline{y}_{k+1} = \phi(\underline{y}_k, \dots, \underline{y}_{k-n_y+1}, \underline{u}_k, \dots, \underline{u}_{k-n_u+1}), \quad (5.77)$$

¹¹At the first few updates, the SONIG algorithm knows nearly nothing of the system, so the posterior distributions of \underline{y}_3 and \underline{x}_3 will not be very different from the prior distributions. When more data has been incorporated into the SONIG algorithm, this will of course change.

where n_y is the output order and n_u is the input order. In this case, we have a function input of the form

$$\mathbf{x}_{k+1} = \begin{bmatrix} \mathbf{y}_k^T & \cdots & \mathbf{y}_{k-(n_y-1)}^T & \mathbf{u}_k^T & \cdots & \mathbf{u}_{k-(n_u-1)}^T \end{bmatrix}^T. \quad (5.78)$$

To identify such a system, we can use Algorithm 1 which outlines all the steps that we need to take.

Input:

A set of inputs $\mathbf{u}_1, \mathbf{u}_2, \dots$ and outputs $\mathbf{y}_1, \mathbf{y}_2, \dots$ of a system that is to be identified. Both the input and the output can be disturbed by noise.

Preparation:

Define hyperparameters by using expert knowledge about the system, or through tuning methods like those described in Section 5.3.1. Optionally, also define an initial set of inducing input points X_u .

Updating:

while there are unprocessed measurements \mathbf{y}_{k+1} **do**

1. Set up $\underline{\mathbf{x}}_{k+1}$ (shortened to $\underline{\mathbf{x}}_+$) according to (5.78). Find its prior Gaussian distribution using known covariances between system outputs $\underline{\mathbf{y}}_k, \dots, \underline{\mathbf{y}}_{k-(n_y-1)}$ and (if necessary) system inputs $\underline{\mathbf{u}}_k, \dots, \underline{\mathbf{u}}_{k-(n_u-1)}$. Also find the prior distribution of the function output $\underline{\mathbf{y}}_{k+1}$ (denoted as $\underline{\mathbf{f}}_{k+1}$ or shortened as $\underline{\mathbf{f}}_+$).
2. Apply (5.61) to find the posterior distribution $\mathcal{N}(\hat{\mathbf{x}}_+, \Sigma_{x_+}^+)$ of $\underline{\mathbf{x}}_+$. Use this to update the posterior distribution of the system outputs $\underline{\mathbf{y}}_k, \dots, \underline{\mathbf{y}}_{k-(n_y-1)}$ and system inputs $\underline{\mathbf{u}}_k, \dots, \underline{\mathbf{u}}_{k-(n_u-1)}$.
3. Optionally, if $\hat{\mathbf{x}}_+$ is far removed from any inducing input point, add it to the set of inducing inputs X_u using (4.57). (Or rearrange the inducing input points in any desired way.)
4. Calculate the posterior distribution of the inducing input vector $\underline{\mathbf{f}}_u$ for each of the outputs of ϕ using (5.50) and (5.51).
5. Calculate the posterior distribution of the output $\underline{\mathbf{y}}_{k+1}$ using (5.68) and (5.69). Additionally, calculate the covariances between $\underline{\mathbf{y}}_{k+1}$ and each of the previous system outputs $\underline{\mathbf{y}}_k, \dots, \underline{\mathbf{y}}_{k-(n_y-1)}$ and inputs $\underline{\mathbf{u}}_k, \dots, \underline{\mathbf{u}}_{k-(n_u-1)}$ through (5.71).

end

Prediction:

For any deterministic set of previous outputs $\mathbf{y}_k, \dots, \mathbf{y}_{k-(n_y-1)}$ and inputs $\mathbf{u}_k, \dots, \mathbf{u}_{k-(n_u-1)}$, apply the sparse GP regression equations (4.9) to predict the next output \mathbf{y}_{k+1} . For stochastic parameters, use the expansions from Section 5.1.

Algorithm 1: System identification through SONIG: an application of the SONIG algorithm to identify a non-linear system in an online way.

5.4. EXPERIMENTS

It is time to put the SONIG algorithm to the test. We first take a look at how well it performs compared to other GP regression algorithms, when applied to a simple function subject to input noise (Section 5.4.1). We then perform system identification through the SONIG algorithm. We do this both for the benchmark problem of a fluid damper, comparing it to other nonlinear system identification algorithms (Section 5.4.2), and for the by now familiar pitch-plunge system (Section 5.4.3).

5.4.1. APPLICATION TO A TEST FUNCTION

We will start with a simple one-dimensional function approximation example. We will set up a Gaussian process with $\lambda_f = \lambda_x = 1$ on the range $[-5, 5]$. From this Gaussian process, we take a random sample function. We then take n_m random input points and perform measurements (\mathbf{x}_m, f_m) on this sample function. Subsequently, we distort these measurements through an input noise with standard deviation $\hat{\sigma}_{x_m} = 0.4$ and output noise $\hat{\sigma}_{f_m} = 0.1$ to get our actual measurement $(\hat{\mathbf{x}}_m, \hat{f}_m)$.

We want to know how good the SONIG algorithm works, in various set-ups. To figure this out, we will use a variety of different GP regression set-ups.

- (1) Regular GP regression with the exact hyperparameters, applied to the data set without input noise ($\hat{\sigma}_f = 0$) but with output noise. This serves as a reference case, to see how much we lose due to the input noise. All other algorithms do get noisy measurement input points and have to tune their own hyperparameters.
- (2) Regular GP regression. Hyperparameters are tuned through the maximum-likelihood method of Section 3.1.3. Measurements are (as usual) distorted by both output and input noise.
- (3) The NIGP algorithm of [McHutchon and Rasmussen \(2011\)](#). This algorithm has its own method of tuning hyperparameters, including $\hat{\sigma}_{x_m}$.
- (4) The SONIG algorithm, starting with $\boldsymbol{\mu}_u = \mathbf{m}_u$ and $\Sigma_{uu} = K_{uu}$, 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 measurements (800 instead of 200). Because the SONIG algorithm is computationally a lot 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 measurements) to predict the distribution $\mathcal{N}(\boldsymbol{\mu}_u, \Sigma_{uu})$ of the inducing input points, followed by the SONIG algorithm applied to the remainder (700) of the measurements, 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, to see how well we do when we ignore the presence of input noise.

For all the algorithms, the most important quality indicator is the *Mean Squared Error* (MSE). To measure this, we let each regression algorithm predict the output $\underline{f}_* \sim$

Table 5.1: Comparison of various GP regression algorithms, applied to noisy measurements of 400 randomly generated sample functions. For details, see the main text.

Set-up	Measurements	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

$\mathcal{N}(\mu_*, \Sigma_{**})$ for a large trial input set X_* . We then compare μ_* with the true function values and derive the MSE from this.

But just looking at the MSE is not enough. A crucial part of Gaussian process regression is that it knows how certain its predictions are. This is indicated by the covariance matrix Σ_{**} , or at least, its diagonal elements $\Sigma_{*,i} \Sigma_{i,*}$. Ideally the *mean variance* for all our predictions should equal the MSE. This indicates that the algorithm is *integer*. If the MSE is much larger than the mean variance, then it means that the algorithm believes it is much more accurate than it actually is, leading to an overconfident algorithm. As such, the ratio between the MSE and the mean prediction variance is an indication of the ‘overconfidence’ of the algorithm.

An example outcome of the experiment is shown in Figure 5.4. Here we see that the SONIG algorithm actually seems to be better at approximating strongly varying functions than the NIGP algorithm. This is initially somewhat surprising, since the SONIG algorithm has more approximating assumptions. However, this difference is mainly because the SONIG algorithm takes into account the second derivative of the mean in its update law (5.50), while the NIGP method does not. If the SONIG algorithm also ignores this second derivative, its performance would be similar to that of the NIGP algorithm.

However, we cannot make significant conclusions based on just one example run. Instead, we run the experiment a large number (400) times. The average of the results is subsequently shown in Table 5.1. There are various things that can be noticed from this table. First of all, it confirms that the SONIG algorithm works better on these kinds of problems than the NIGP algorithm, which we already explained.

Secondly, it seems that all algorithms are rather overconfident in their predictions, apart from the regular Gaussian process regression method. That is, their actual MSE is far higher than what the algorithm expects the MSE to be (being the mean predicted variance).

Thirdly, it appears that using more measurements will provide a better accuracy. This is of course not very surprising. But nevertheless it is worthwhile to note that the FITC method (which does not take into account input uncertainty at all) with 800 measurements performs better than the NIGP method with 200 measurements, and is still faster. So sometimes it may be worthwhile to go for a ‘quick and dirty’ approach.

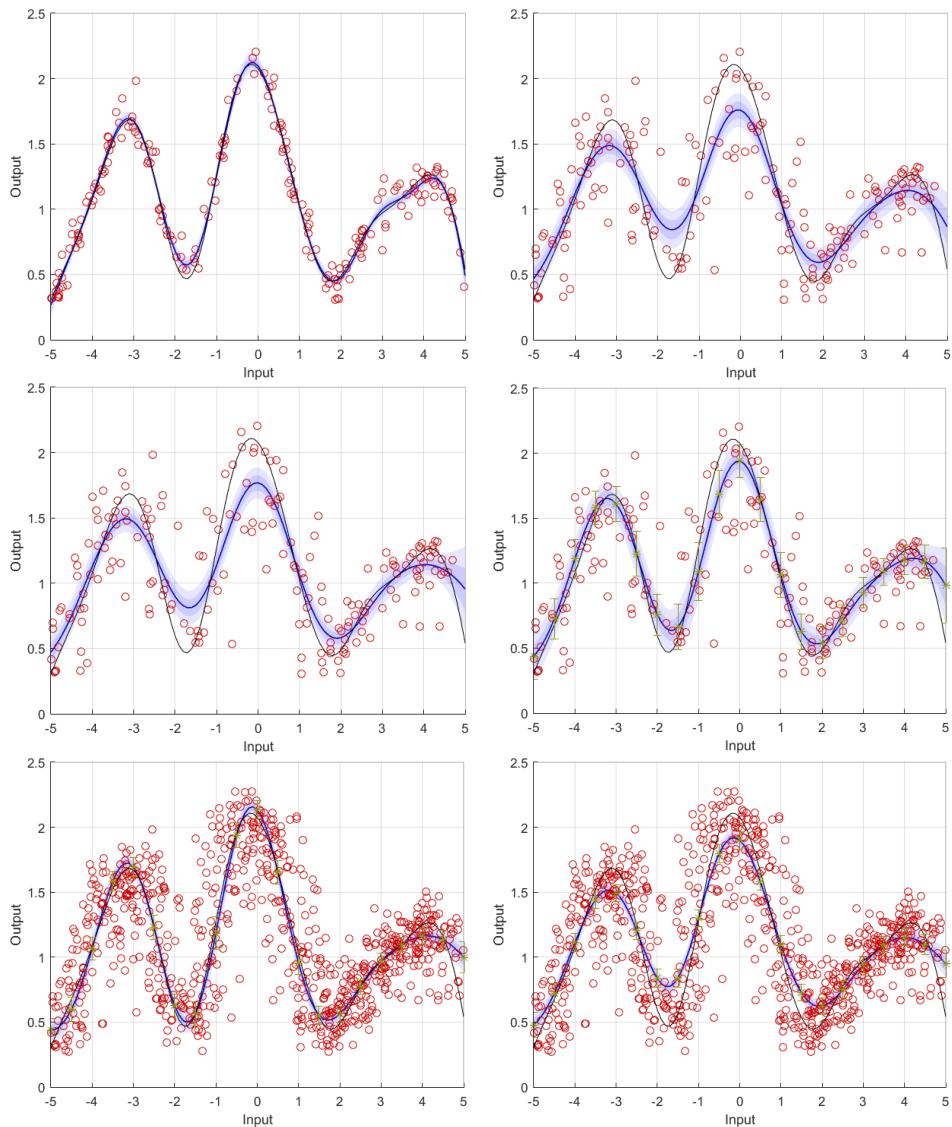


Figure 5.4: Example results after running the various algorithms described in the main text. Shown are the results for (1) and (2) in the top, (3) and (4) in the middle and (5) and (7) in the bottom. The results for (6) are not shown because they are nearly identical to those of (5).

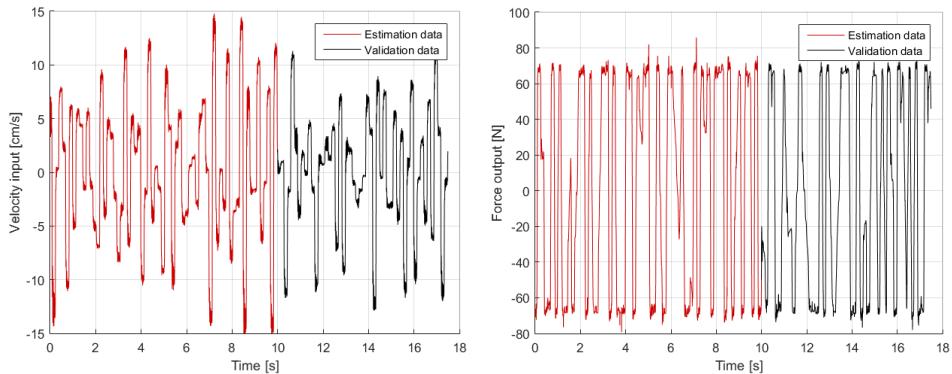


Figure 5.5: The measurements obtained from the magneto-rheological fluid damper. These are used to identify the dynamical behavior of the system.

5.4.2. IDENTIFICATION OF A MAGNETO-RHEOLOGICAL FLUID DAMPER

5

The next step is to test Algorithm 1, applying system identification through SONIG. We will start doing so with a benchmark problem which is used more often in literature: modeling the dynamical behavior of a magneto-rheological fluid damper. In particular, the fluid damper is given a certain velocity (the input) and the damping force resulting from this (the output) is then measured.

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). More recently, it has been used in the context of Gaussian Process State Space Models (GP-SSM) by Svensson et al. (2016) in their Reduced Rank GP-SSM (RR GP-SSM) algorithm. These are the methods which we will compare the SONIG algorithm to.

The measurement data has 3499 measurements, sampled every $\Delta t = 0.05$ seconds, and consisting of a single input u and a single output y . These are shown in Figure 5.5. We will use the first 2000 measurements (10 seconds) for training (estimation) and the next 1499 measurements (7.5 seconds) for evaluation (validation). The MathWorks Inc. (2015) recommended to use one past output and three past inputs to predict subsequent outputs. Based on this, we should use a model of the form

$$y_{k+1} = \phi(y_k, u_k, u_{k-1}, u_{k-2}). \quad (5.79)$$

We can tune the hyperparameters through the NIGP algorithm. Passing all 2000 measurements to this algorithm will be very slow, so we feed a subset to it. Subsequently, we do some further manual tuning of the hyperparameters, winding up with

$$\begin{aligned} \Lambda_x &= \text{diag}(70^2, 20^2, 10^2, 10^2), & \lambda_f^2 &= 70^2, \\ \Sigma_{+x} &= \text{diag}(2^2, 0.1^2, 0.1^2, 0.1^2), & \Sigma_{+f} &= 2^2. \end{aligned} \quad (5.80)$$

Interestingly, these hyperparameters tell us something about the significance of the various input parameters. It seems that u_k does not affect the output y_{k+1} as much as earlier inputs u_{k-1} and u_{k-2} , as shown by its larger length scale.

After processing a measurement \hat{y}_{k+1} , the SONIG algorithm will provide us with a posterior distribution of \underline{y}_{k+1} , \underline{y}_k , \underline{u}_k , \underline{u}_{k-1} and \underline{u}_{k-2} . Of these parameters, we need the joint posterior distribution of \underline{y}_{k+1} , \underline{u}_k and \underline{u}_{k-1} for the prior distribution of the next input point.

We do not pick a specific set of inducing input points. Instead, we add them in an online way according to the method discussed in Section 4.3.3, whenever we encounter a training input point which is not close to any already existing inducing input point. With the exact settings used, which can be found in the source code through [Bijl \(2016a\)](#), this results in 32 inducing input points. This is a modest number, resulting in a relatively fast training time of only a few (roughly 10) seconds for all 2000 measurements.

Now that the SONIG algorithm has been trained, we can apply it to the validation data set. For this, we inform the SONIG algorithm about the starting point of the magneto-rheological fluid damper and subsequently only feed it input data u . Based on this, the algorithm needs to figure out the output y at every subsequent time step.

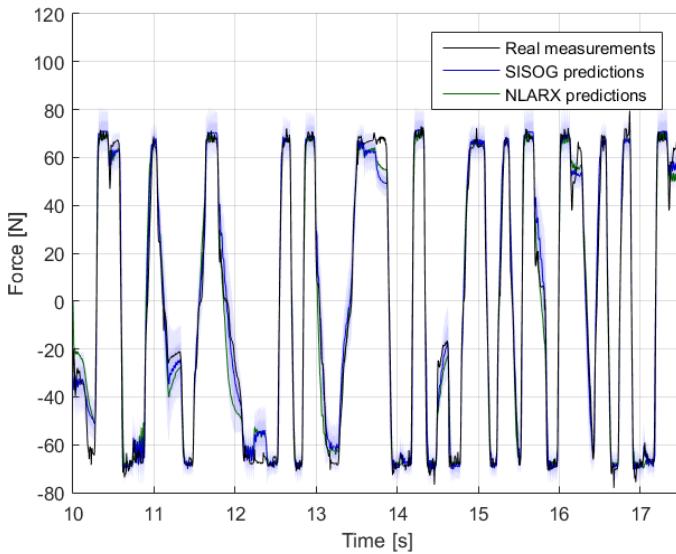
During this process, we also take into account the uncertainty in the SONIG estimates. Note that, when predicting \underline{y}_k , the resulting posterior distribution of \underline{y}_k will have some uncertainty. In other words, this random variable has a nonzero variance. If we then use \underline{y}_k to predict \underline{y}_{k+1} , we take this variance into account according to the methods of Section 5.1.

By applying all these ideas in the proper way, we get the results shown in Figure 5.6. Here we see that the SONIG algorithm is very well capable of predicting future outputs. The *Root Mean Squared Error* (RMSE) of the SONIG predictions, compared to that of other methods, is shown in Table 5.2. This table also lists the results we would get if we'd blindly apply GP regression or the NIGP algorithm to the nonlinear ARX function (5.77). It confirms the good performance of the SONIG algorithm.

Table 5.2: Comparison of the results of various system identification methods when applied to the 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 paper
GP regression	9.87	This paper
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

The caveat for these results is that they only follow from the proper hyperparameters. Changing the hyperparameters slightly will already give results that are slightly worse, though still better than those of the other methods. (Yes, I have done a bit of tweaking, though not all that much.) And using strongly different hyperparameters will of course



5

Figure 5.6: Prediction of the output of the magneto-rheological fluid damper by the SONIG algorithm, compared to the true output and the best non-linear ARX model given by [The MathWorks Inc. \(2015\)](#). Note that in transition regions like those at $t = 11.3\text{s}$, in which the SONIG algorithm is trained less well, the uncertainty is relatively large. Also note that the two system identification methods often make the same mistakes, like for instance at $t = 12.3\text{s}$.

completely invalidate the predictions.

5.4.3. NOISY STATE MEASUREMENTS OF THE PITCH-PLUNGE SYSTEM

Going back to wind energy applications, we apply our system identification algorithm to the nonlinear pitch-plunge system described in Section 2.6. You may recall that this system has four states: h , α , \dot{h} and $\dot{\alpha}$. We usually write $\mathbf{x} = [h \quad \alpha]^T$ so that the state consists of \mathbf{x} and $\dot{\mathbf{x}}$. There was also one input 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

$$\mathbf{x}_{k+1} \approx f(\mathbf{x}_k, \mathbf{x}_{k-1}, \beta_k). \quad (5.81)$$

This is the function that we will strive to find 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 5.7.

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

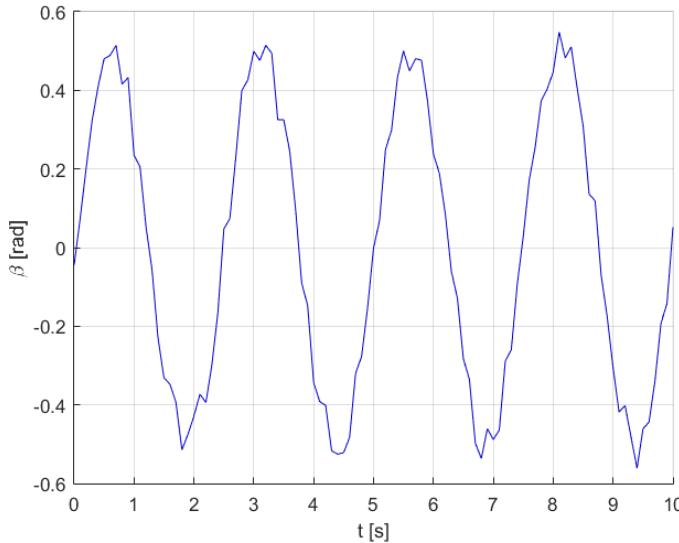


Figure 5.7: The first ten seconds of input $\beta(t)$ provided to the pitch-plunge system during its identification through the SONIG algorithm. It is a sinusoid with distortions added for extra excitations.

5

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.

For the given input signal we run a simulation. This results in the state development shown in Figure 5.8. 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 5.9. For these predictions, only the state x at time $t = 50$ s was given as well as the input β at any subsequent time.

In Figure 5.9 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

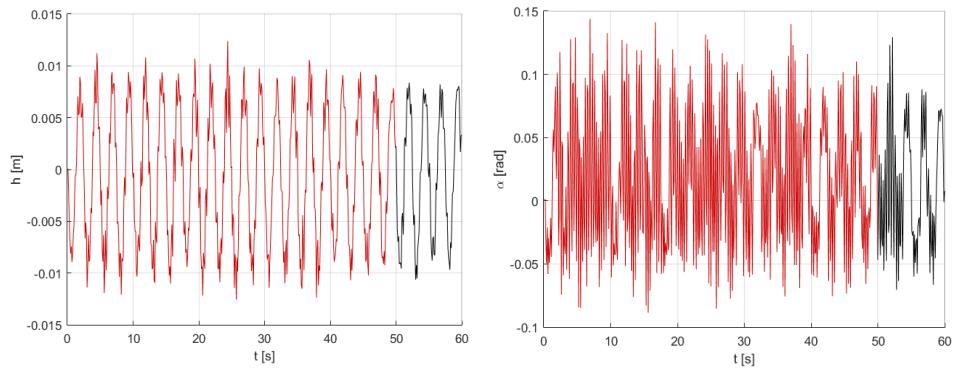


Figure 5.8: The state development of the pitch-plunge system. The first fifty seconds are used for training and the remaining ten seconds for validation. Measurements were taken at a wind speed of $U = 10 \text{ m/s}$.

5

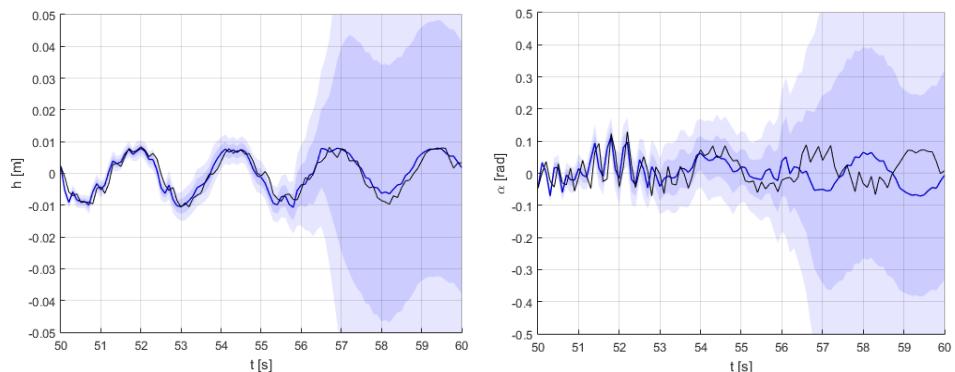


Figure 5.9: The prediction of the state development of the pitch-plunge system for the final ten seconds. Initially the estimates are valid, but as we go further into the future, the uncertainties grow, until the prediction algorithm has no idea anymore what the future will hold.

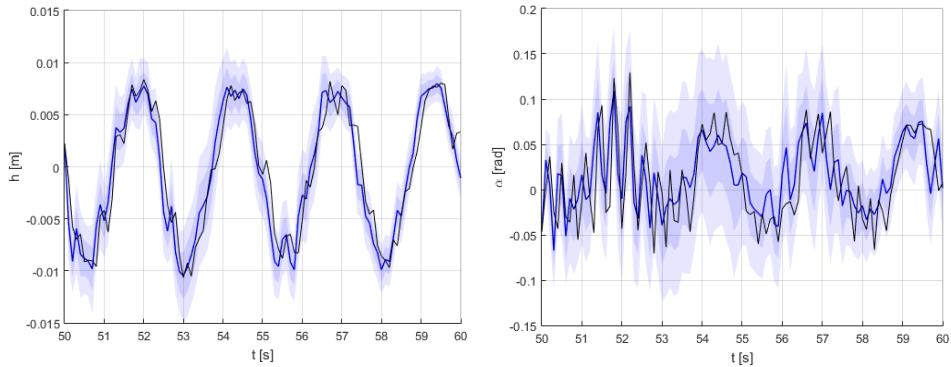


Figure 5.10: The prediction of the state development of the pitch-plunge system for the final ten seconds, when the SONIG algorithm is told to reduce the variance of its estimates by 5% after every time step.

5

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 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 5.10. 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 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 and computational resources are limited. And that strongly depends on the background of the person applying the SONIG algorithm.

5.5. OVERVIEW OF LITERATURE

As usual we take a look at how this subfield of Gaussian process regression developed, and what potential areas of future exploration.

5.5.1. LITERATURE OVERVIEW

The general problem of dealing with uncertain input points in (not necessarily Gaussian process) regression problems is known as the *errors-in-variables regression*. An early analysis into such methods was given by [DellaPortas and Stephens \(1995\)](#).

When it comes to Gaussian processes regression, early work was done by [Girard and Murray-Smith \(2003\)](#). Their approach towards dealing with stochastic trial points con-

sisted of using a Taylor approximation of the Gaussian process and integrating over that. Though promising, it is not the solution which has become commonplace for stochastic trial points. After all, it turns out that a Taylor approximation is not necessary here. Instead, the moment matching approach discussed in Sections 5.1.3 and 5.1.4 suffices. This approach was already described by [Girard et al. \(2003\)](#), [Candela et al. \(2003\)](#), but was further developed by [Deisenroth \(2010\)](#), who then successfully used it in his PILCO algorithm (see [Deisenroth and Rasmussen \(2011\)](#)).

When it comes to stochastic measurement points, there have been various less successful attempts. One attempt was made by [Dallaire et al. \(2009\)](#), who effectively applied the uncertainty incorporating SE covariance function (5.32). A similar approach was taken by [Girard and Murray-Smith \(2003\)](#), who incorporated a Taylor approximation to set up their own uncertainty incorporating covariance function.

The problem with these methods is that they do not take into account posterior data. To see how this works, suppose that our measurements have given us a strong indication that the function $f(\mathbf{x})$ that we are approximating is mostly flat in one part of the input space, and highly sloped in another part. In that case, we know that in the first part the input noise does not have any significant effect, while in the second part it most certainly does. This is useful data which we should take into account. Yet none of the articles which we just described do. As a result, their predictions do not use all available data and hence will not be as accurate as they could be.

A method that does take into account posterior data is the NIGP method developed by [McHutchon and Rasmussen \(2011\)](#), also described in Section 5.2.2. This is a promising and effective method of taking into account uncertainty in the measurement input points. Its main downside is that it cannot be applied to large data sets. That issue is solved by the SONIG method developed in this chapter and published about through [Bijl et al. \(2017a\)](#).

There are also other more circuitous methods to take into account uncertainty in the measurement input points. Recent work makes use of techniques like variational inference, as described by [Titsias \(2009\)](#), [Titsias and Lawrence \(2010\)](#). To learn more about how you can apply these methods to take into account noisy input points, read the work by [McHutchon \(2014\)](#), [Damianou et al. \(2016\)](#).

In addition, it is also possible to see the problem as a heteroscedastic one. In a heteroscedastic problem, the noise variance is yet another function of the input \mathbf{x} . Contributions on this were made by [Goldberg et al. \(1998\)](#), [Le et al. \(2005\)](#), [Snelson and Ghahramani \(2006b\)](#), [Kersting et al. \(2007\)](#), [Lazaro-Gredilla and Titsias \(2011\)](#), [Wang and Neal \(2012\)](#). We will not go further into depth on this though. After all, assuming heteroscedasticity would give more degrees of freedom to the learning algorithm than is actually required – we already know that the ‘additional output noise’ due to the input noise depends on the slope of the function – so these methods are expected to perform less well than other methods which do take this knowledge into account.

5.5.2. SUGGESTIONS FOR FURTHER RESEARCH

The SONIG algorithm developed in this chapter is still very new. As a result, there are plenty of ways in which it can be improved further. We will look at a few.

- **Improving the method to find the posterior input distribution**

In Section 5.2.4 we looked at a method to find/approximate the posterior distribution of the measurement input point. At the end of this section we discussed an iterative procedure of getting the most accurate approximation. The problem with this procedure is that it does not always converge. It would be very interesting to figure out in which cases this happens, and whether this can be worked around in such a way that the method always returns a proper posterior distribution.

- **Improving the toolbox inversion problems**

There is a SONIG Matlab toolbox available at [Bijl \(2016b\)](#). The main problem that this toolbox is subject to is the inability of Matlab to invert certain types of matrices. (For example, take $X_u = \{-10, -9, \dots, 9, 10\}$, use $\lambda_f = 1$, $\lambda_x = 4$ and set up K_{uu} . Matlab will not be able to properly find the inverse K_{uu}^{-1} .) Working around this in some way would significantly improve at least the user-friendliness of the toolbox. However, doing so might require fundamental changes in the way Matlab inverts matrices, or a switch away from Matlab altogether. I would recommend the latter.

- **Incorporating a prior covariance between the input and the output**

In Section 5.3.4 we have looked at how we can calculate the posterior covariance between the input \underline{x}_+ and the output \underline{f}_+ . However, if such a covariance is also present a priori, then the SONIG algorithm is not yet capable of taking that into account. This may for instance happen when the measurement noise present on the input and on the output is correlated. How can we take such a correlation into account?

- **Online tuning of the hyperparameters**

The SONIG algorithm is an online method. However, it depends on offline methods (like the NIGP method) for the tuning of its hyperparameters. Just like we wondered in the previous chapter, we should once more ask ourselves, ‘Is it possible to tune the hyperparameters in an online way?’ And can we then also do so when the measurement input points are subjected to noise?

- **Improving the NIGP algorithm by incorporating the second mean derivative**

In footnote 8 on page 126 we noted that the NIGP method could potentially be improved by incorporating the second derivative of the mean in its expressions. Whether this actually amounts to an improvement is something which needs to be confirmed by experiments. It would be interesting to see how much of an improvement this would actually result in. Also, incorporating this improvement into the NIGP source code could also be useful for people using this code.

- **Sampling from the future development of a system output**

In Sections 5.4.2 and 5.4.3 we saw that we can use the SONIG algorithm to predict the future for systems we identified. The result can again be plotted like a Gaussian process. An interesting question is, ‘Can we also take samples from this prediction, as if it was a Gaussian process?’

For the Gaussian processes we have seen in earlier chapters this used to be possible, because we can know the covariance between any two predicted values. However, the SONIG algorithm does not provide us with the covariance between predicted values that are quite some time apart. So how would sampling work then? Is it possible

to adjust the SONIG algorithm to calculate such covariances after all? And additionally, if it would be possible to sample from the future development of the state, would these samples be sensible, especially after the variance blows up like in Figure 5.9?

6

GAUSSIAN PROCESS OPTIMIZATION

Summary — The maximum of a Gaussian process does not occur at a fixed point, but it is actually a random variable with a distribution of its own. This distribution cannot be calculated analytically, and trying to find it by looking at the derivative of a Gaussian process fails because we also find local optimums. Instead, we can approximate the maximum distribution using a Monte Carlo approach: through particles.

To improve the convergence properties of the resulting Monte Carlo maximum distribution algorithm, we can implement several ideas from sequential Monte Carlo samplers, mostly related to importance sampling. When we do, we can efficiently approximate the distribution of the maximum of a Gaussian process.

We can then apply this algorithm to the Gaussian process optimization problem. The problem here is to optimize an unknown function: which inputs should we try, to find the optimum, while keeping the regret limited? A variety of methods exist here. Most of them make use of an acquisition function, and we pick the input point maximizing this acquisition function. An alternative idea is to sample from the maximum distribution, which results in Thompson sampling.

Experiments show that which optimization method works best mostly depends on which function you are optimizing, as well as how much you tune the optimization method to this function. As such, we cannot say which optimization method works best.

Despite this, various optimization methods have successfully been applied at tuning the controller gains of a wind turbine simulation, reducing the fatigue damage it sustained by minimizing the damage equivalent load. Directly minimizing the damage equivalent load in an online data-driven way is something that most tuning algorithms cannot do, due to the nonlinear nature of the damage equivalent load. However, Gaussian process optimization has been designed to deal with uncertainties and nonlinearities, making it ideally suited for systems suffering from these properties.

We often use Gaussian processes to approximate value functions, cost functions and such. Because of this, it is important to know how to find the maximum (or equivalently, the minimum) of a Gaussian process. We can do this either for a given Gaussian process, or we can choose measurement points ourselves to most efficiently find the maximum.

We will start with looking at how to find the maximum of a given Gaussian process (Section 6.1). We discover that particles methods are an effective way of doing so, but to further improve on the developed method, we need to learn more about particle methods. That is why we take a quick intermezzo on sequential Monte Carlo samplers (Section 6.2) and then use these ideas to further improve our methods (Section 6.3).

After knowing how to find the maximum of a given Gaussian process, we look into the process of Gaussian process optimization, where we have to find the optimum of a function $f(\mathbf{x})$, all the while approximating it through Gaussian process regression (Section 6.4). We continue by testing all our methods in a couple of experiments (Section 6.5) and end with an overview of the literature (Section 6.6).

6.1. FINDING THE MAXIMUM OF A GAUSSIAN PROCESS

Consider a given Gaussian process. How do we find the maximum of this? That is the central question we will look at now. We start with introducing the concept of the maximum distribution (Section 6.1.1). We then apply an analytical approach (Section 6.1.2), a derivative approach (Section 6.1.3) and a particle approach (Section 6.1.4) towards finding this maximum distribution. For the particle approach, we then study the distribution which the particles in the limit converge to (Section 6.1.5) and what this distribution intuitively means (Section 6.1.6).

6

6.1.1. THE MAXIMUM DISTRIBUTION

Consider a Gaussian process with a posterior mean function $\mu(\mathbf{x})$ and posterior covariance function $\Sigma(\mathbf{x}, \mathbf{x}')$. An example is shown in Figure 6.1. We want to find the maximum of this Gaussian process. That is, the *optimal input*¹ \mathbf{x}^* where this maximum appears and the corresponding *optimal output* $f^* = f(\mathbf{x}^*)$. However, first we should ask ourselves ‘What do we mean with the maximum?’

You could argue that the maximum of the GP is simply the maximum of the mean function $\mu(\mathbf{x})$, but this would be incorrect. After all, we have seen in Section 2.3.2 that a Gaussian process is basically a distribution over functions. As such, we should look at various sample functions from this distribution, which are also shown in Figure 6.1.

Here we see something interesting though. All these samples functions from Figure 6.1 have a different maximum! As such, the maximum \mathbf{x}^* is not a fixed point, but is a random variable $\underline{\mathbf{x}}^*$. And just like every random variable, it has a distribution called the *maximum distribution*. Hence, finding the maximum of a Gaussian process comes down to finding the maximum distribution.

How do we do that? Option one is to sample thousands of functions from the Gaussian process, find the maximum for each one and make a histogram of the results. This brute-force method gets the job done pretty well, resulting in Figure 6.2. However, it

¹In literature the optimal input is usually denoted by \mathbf{x}_* , but we use \mathbf{x}^* so as not to confuse it with trial input points \mathbf{x}_* .

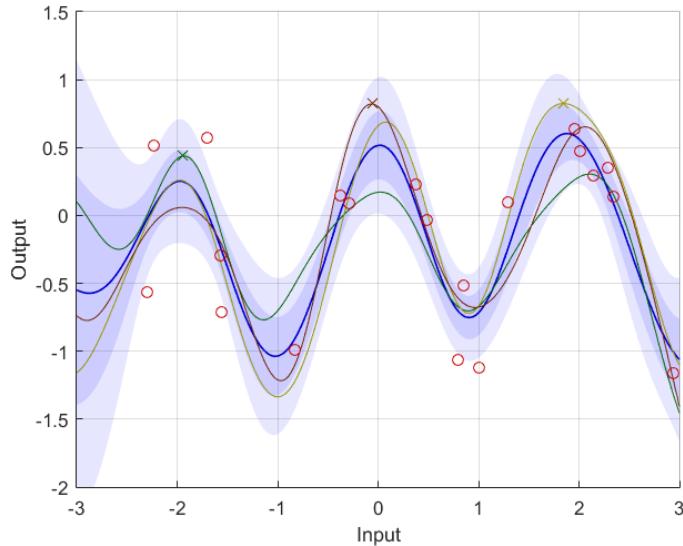


Figure 6.1: An example Gaussian process ($\lambda_f = 1$, $\lambda_x = 0.6$) generated from $n_m = 20$ measurements, with three sample functions. Note that the three sample functions all have their maximums, indicated by crosses, in very different positions. The measurements were generated from the function $f(x) = \cos(3x) - \frac{1}{9}x^2 + \frac{1}{6}x$ (not shown here) and were subjected to a noise with standard deviation $\hat{\sigma}_{f_m} = 0.3$.

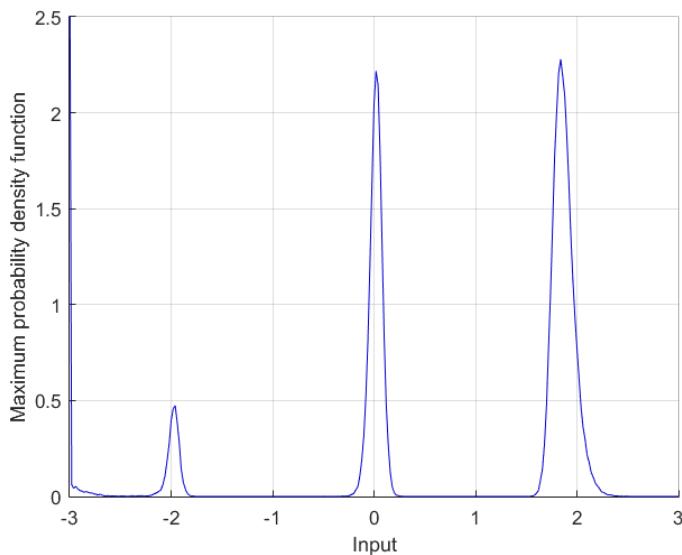


Figure 6.2: The maximum distribution of the Gaussian process of Figure 6.1. It was obtained by taking 100000 sample functions from the GP, finding the position of the maximum for each one and generating a histogram. Note that within the interval $[-3, 3]$ there are three places where maximum is likely to be, each with its own likelihood. There is also a spike at the left end of the interval. Such spikes regularly happen when there is a strong uncertainty near the end of the interval.

is very computationally intensive. And although this can be solved through parallel computing, we still run the risk of finding only local optimums, especially for higher-dimensional nonlinear functions. So isn't there an analytical solution for the maximum distribution?

6.1.2. AN ANALYTICAL APPROACH TO FINDING THE MAXIMUM

Let's suppose that we have a set X of possible input points $\mathbf{x}_1, \mathbf{x}_2, \dots$. (Usually, X equals the set of trial points X_* .) For each of these input points \mathbf{x}_i , we have a function value $\underline{f}_i \equiv f(\mathbf{x}_i)$. We now write the maximum distribution as $p_m(\mathbf{x}_i)$. So we have $p_m(\mathbf{x}_i) \equiv p(\mathbf{x}_i = \mathbf{x}^*)$. In other words, this equals the probability that \underline{f}_i is the maximum.

To make this problem a bit easier, we will for now assume that the number of points is a finite number n . Within Gaussian processes, this does not have to be the case, so this assumption does limit us somewhat, but for now we ignore that. In this case, the probability that \underline{f}_1 is the maximum can be written as

$$p_m(\mathbf{x}_1) = p(\underline{f}_1 > \underline{f}_2, \dots, \underline{f}_1 > \underline{f}_n), \quad (6.1)$$

and similarly for other input points. To find this probability, we can find the difference

$$\begin{aligned} \underline{\tilde{f}} &\equiv \begin{bmatrix} \underline{f}_1 \\ \vdots \\ \underline{f}_n \end{bmatrix} - \begin{bmatrix} \underline{f}_2 \\ \vdots \\ \underline{f}_n \end{bmatrix} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}) \\ &= \mathcal{N}\left(\begin{bmatrix} \mu_1 - \mu_2 \\ \vdots \\ \mu_1 - \mu_n \end{bmatrix}, \begin{bmatrix} \Sigma_{11} - \Sigma_{12} - \Sigma_{21} + \Sigma_{22} & \cdots & \Sigma_{11} - \Sigma_{1n} - \Sigma_{21} + \Sigma_{2n} \\ \vdots & \ddots & \vdots \\ \Sigma_{11} - \Sigma_{12} - \Sigma_{n1} + \Sigma_{n2} & \cdots & \Sigma_{11} - \Sigma_{1n} - \Sigma_{n1} + \Sigma_{nn} \end{bmatrix}\right). \end{aligned} \quad (6.2)$$

The probability that \underline{f}_1 is the maximum is now given by

$$p_m(\mathbf{x}_1) = p(\underline{\tilde{f}} > \mathbf{0}), \quad (6.3)$$

where the inequality must hold for every element of the vector $\underline{\tilde{f}}$. Note that the number of elements of $\underline{\tilde{f}}$ equals $n - 1$. As such, when $n = 1$ we directly have $p_m(\mathbf{x}_1) = 1$ and when $n = 2$ it follows through Theorem B.12 that

$$\begin{aligned} p_m(\mathbf{x}_1) &= p(\underline{\tilde{f}} > \mathbf{0}) \\ &= \Phi(\infty) - \Phi\left(\frac{\mu_2 - \mu_1}{\sqrt{\Sigma_{11} + \Sigma_{22} - 2\Sigma_{12}}}\right) \\ &= \Phi\left(\frac{\mu_1 - \mu_2}{\sqrt{\Sigma_{11} + \Sigma_{22} - 2\Sigma_{12}}}\right), \end{aligned} \quad (6.4)$$

where $\Phi(x)$ is the *Cumulative Density Function* (CDF) of the standard Gaussian distribution, defined in (B.70). Note that we have used $\Phi(\infty) = 1$ and $1 - \Phi(x) = \Phi(-x)$. The CDF $\Phi(x)$ does not have an analytical expression, but at least it is well-known and can be easily computed through numerical methods.

This is a different story when $n > 2$. Now we need to compute the probability

$$p_m(\mathbf{x}_1) = p(\tilde{\mathbf{f}} > \mathbf{0}). \quad (6.5)$$

However, doing so in an efficient and accurate way for arbitrary $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\Sigma}}$ is a very complicated problem.

For small n (up to $n = 4$) the resulting probability is usually approximated through *adaptive quadratures*, while for larger n (up to $n = 25$) it is often better to use some kind of quasi-Monte Carlo integration algorithm. For more details about these methods, see the work by Drezner (1994), Genz (2004).

However, when we have many more dimensions, the problem becomes impossible to solve. Either the results will be very inaccurate, or the runtime will increase to unacceptable proportions. (Or both.) In Gaussian process regression we generally work with hundreds to thousands of points, if not more. As such, we can conclude that solving this problem analytically is not possible.

6.1.3. A DERIVATIVE APPROACH TO FINDING THE MAXIMUM

Another option is to use the derivative to find the position of the maximum. At the position of the maximum, the derivative must be zero. Since the derivative of a Gaussian process is a Gaussian process as well, as we saw in Section 2.5.1, we can just check the probability of $\frac{df}{dx}$ being equal to zero. When we apply this and normalize the result, we get Figure 6.3 (right).

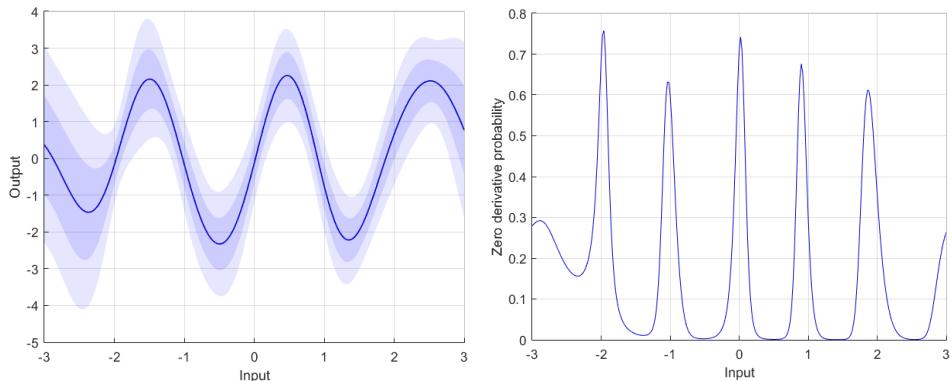


Figure 6.3: For the Gaussian process shown in Figure 6.1, the derivative Gaussian process (left) together with the normalized probability that this derivative equals zero (right).

There are two main problems with this approach. The first is that we also find the minima of the function. This can be solved by taking into account the second derivative as well. That is, we consider the probability

$$p\left(\frac{df}{dx} = \mathbf{0}, \frac{d^2f}{dx^2} < 0\right) = p\left(\frac{d^2f}{dx^2} < 0 \mid \frac{df}{dx} = \mathbf{0}\right) p\left(\frac{df}{dx} = \mathbf{0}\right). \quad (6.6)$$

The easy part here is finding the probability $p\left(\frac{df}{dx} = \mathbf{0}\right)$, since we just calculated this for Figure 6.3. The hard part is finding the probability

$$p\left(\frac{d^2f}{dx^2} < 0 \mid \frac{df}{dx} = \mathbf{0}\right). \quad (6.7)$$

We know how to find the joint distribution of $\frac{df}{dx}$ and $\frac{d^2f}{dx^2}$ and as such we can also find the distribution of $\frac{d^2f}{dx^2} \mid \frac{df}{dx} = 0$. However, you should keep in mind that, since x is a vector, this quantity is a matrix. As such, we need to determine the probability that this random matrix is negative definite. This problem by itself is very hard, if not impossible, to solve efficiently.

We can work around this, if we add an assumption: we assume that all non-diagonal elements of the second derivative $\frac{d^2f}{dx^2}$ are zero. In this case, checking for negative definiteness is easy: all diagonal elements have to be negative. However, this assumption, suggested by Hernández-Lobato et al. (2014a), can also mark saddle points as optimums, which is of course incorrect.

If we are dealing with a single-input function in which the input x equals a scalar x , this saddle point problem does not occur. After all, a scalar value is negative definite when it is negative. By working out this idea further, we can obtain the maximum distribution displayed in Figure 6.4, which directly shows a second problem with this approach.

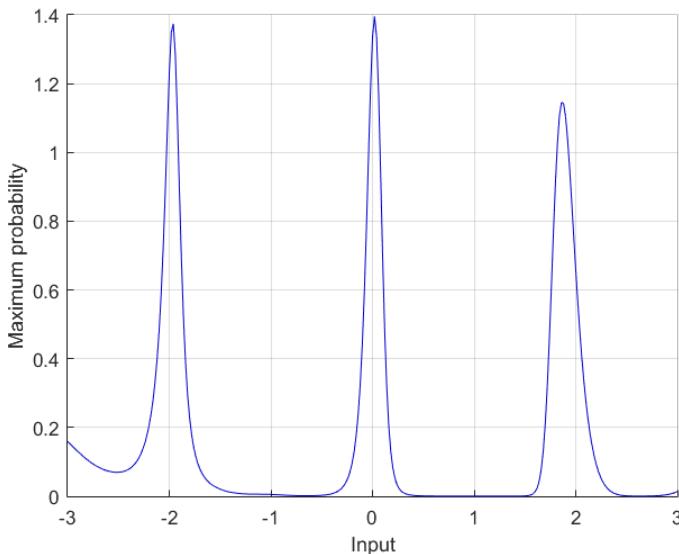


Figure 6.4: The (normalized) probability that the GP from Figure 6.1 has both a zero derivative and a negative second derivative.

Looking at Figure 6.4, we see that all the peaks are more or less equally large, so it

seems as if the maximum is equally likely to be at -2 , at 0 and at -2 . Naturally this is incorrect, and the fault comes from the fact that our method only considers local maximums. We do not consider the value of these maximums and which one might have the largest value.

To solve this, Hernández-Lobato et al. (2014a) suggest to add the requirement that $\underline{f}(\mathbf{x}^*)$ is larger than all previous measurements $\hat{f}_{m_1}, \dots, \hat{f}_{m_{n_m}}$. Or equivalently, by requiring $\underline{f}(\mathbf{x}^*)$ to be larger than the maximum of $\hat{f}_{m_1}, \dots, \hat{f}_{m_{n_m}}$. This is a rather sensitive assumption, as the maximum of these values strongly depends on what noise we happened to have during that particular measurement. A single case of very ‘beneficial’ noise will significantly affect the algorithm. Nevertheless, it may be better than the alternative assumptions, though for more information on this you can read the work by Hernández-Lobato et al. (2014a).

We can conclude that using derivatives may in some cases be an effective way of finding local maximums of a Gaussian process. Finding global maximums is still difficult though, and it requires additional restrictive assumptions. Since this method is not ideal yet, let’s consider a different method.

6.1.4. A PARTICLE APPROACH TO FINDING THE MAXIMUM

Let’s go back to our situation where we had n function values $\underline{f}_1, \dots, \underline{f}_n$. To find the true maximum distribution of Figure 6.2, we basically set up n bins, one for each input \mathbf{x}_i . We then took a sample from the n -dimensional distribution \underline{f} and found the maximum. The bin corresponding to this maximum got one *counter* or *particle*. After a fixed number n_p samples, we were done. We found our approximated maximum probability distribution.

The problem here was that, if n is large, then this will be a computationally intensive process. After all, generating a sample from \underline{f} takes $\mathcal{O}(n^3)$ time. We need to fix that. The idea now is not to compare all n points, but only compare two points at a time. In short, the plan is the following.

1. We start by dividing all n_p particles randomly (or regularly) over all n bins.
2. For each particle, we apply the following steps.
 - (a) Consider the bin \mathbf{x}_i it is in.
 - (b) Pick a random bin \mathbf{x}_j (which in theory might be the same).
 - (c) Set up the joint distribution

$$\begin{bmatrix} \underline{f}(\mathbf{x}_i) \\ \underline{f}(\mathbf{x}_j) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} f_i \\ f_j \end{bmatrix} \middle| \begin{bmatrix} \mu(\mathbf{x}_i) \\ \mu(\mathbf{x}_j) \end{bmatrix}, \begin{bmatrix} \Sigma(\mathbf{x}_i, \mathbf{x}_i) & \Sigma(\mathbf{x}_i, \mathbf{x}_j) \\ \Sigma(\mathbf{x}_j, \mathbf{x}_i) & \Sigma(\mathbf{x}_j, \mathbf{x}_j) \end{bmatrix} \right) \quad (6.8)$$

and take a sample $[\hat{f}_i \quad \hat{f}_j]^T$ from this.

- (d) If $\hat{f}_j > \hat{f}_i$, move the particle from bin \mathbf{x}_i to \mathbf{x}_j .
3. We repeat the above procedure until the distribution of particles has more or less converged.

We call this algorithm the *Monte Carlo Maximum Distribution* (MCMD) algorithm, named

after the Monte Carlo methods which we further discuss in the next Section 6.2.

Let's discuss some terminology. We will call the existing particles, and the bins which they are in, the *champions*. The comparison of a particle and its input point by another input point is called a *challenge* and the corresponding challenging input point is called the *challenger*. The whole process of challenging every individual champion is called a *round of challenges*. We denote the number of rounds used by n_r .

When we apply the algorithm for several rounds, we get the developments shown in Figure 6.5. Here we see that the MCMD algorithm mostly works, but there are two potential issues. The first is that the particles do not seem to converge to the true maximum distribution but to a different distribution. The second is that convergence is somewhat slow. After $n_r = 20$ challenge rounds we are nearly there. For problems with a larger or higher-dimensional input space, we will require far more rounds, which would be unacceptable.

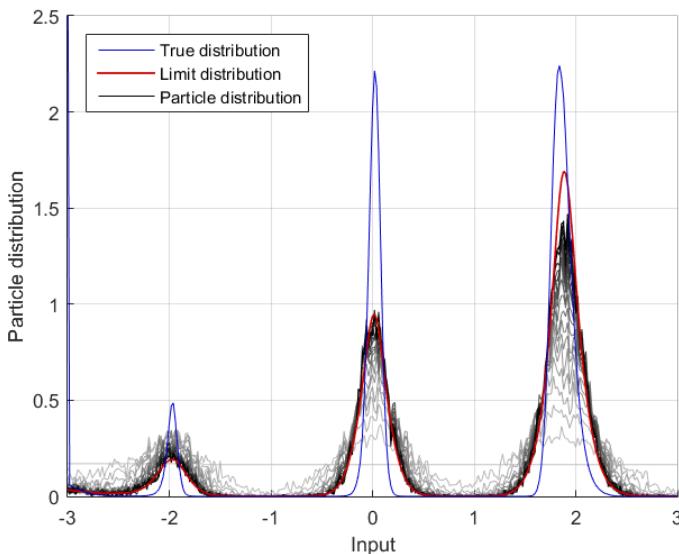


Figure 6.5: The development of the distribution of particles over $n_r = 20$ rounds. Shown are the true maximum distribution from Figure 6.2, the limit distribution of the particles derived in Section 6.1.5 and the distribution of the particles over each of the rounds. The darker the line, the later the round is. Note that the limit distribution of the particles does not equal the true maximum distribution.

We will save the second problem for Section 6.3, after a quick intermezzo into Monte Carlo and particle methods (Section 6.2), but we can look into the problem of the limit distribution now.

6.1.5. FINDING THE LIMIT DISTRIBUTION OF THE PARTICLES

Let's see if we can analytically calculate the limit distribution of the particles in the MCMD algorithm. If we can indeed do so, then we right away know that the distribution is different from the true maximum distribution, since no analytical expression for that distribution exists.

Suppose that, after round k , the number of particles in bin \mathbf{x}_i is denoted by n_i^k . Let's calculate how many particles are expected to be in each bin after another round of challenges. To be precise, let's calculate the quantity n_i^{k+1} .

We know that the particles from any bin \mathbf{x}_j (with $j \neq i$) might move to bin \mathbf{x}_i . There currently are n_j^k particles in bin \mathbf{x}_j . These particles have a $\frac{1}{n}$ chance to be challenged by \mathbf{x}_i . Additionally, let's denote the chance that \mathbf{x}_i will 'beat' \mathbf{x}_j by P_{ij} . As such, a number $\frac{1}{n} P_{ij} n_j^k$ particles will move from bin \mathbf{x}_j to \mathbf{x}_i . This means that the expected number of particles moving into the bin \mathbf{x}_i equals

$$\text{Expected particles entering } \mathbf{x}_i: \frac{1}{n} \sum_{j=1}^n P_{ij} n_j^k. \quad (6.9)$$

In this equation, we can calculate the probability P_{ij} from (6.4). It equals

$$P_{ij} \equiv p(\underline{f}_i > \underline{f}_j) = \Phi\left(\frac{\mu_i - \mu_j}{\sqrt{\Sigma_{ii} + \Sigma_{jj} - 2\Sigma_{ij}}}\right) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{\mu_i - \mu_j}{\sqrt{2(\Sigma_{ii} + \Sigma_{jj} - 2\Sigma_{ij})}}\right), \quad (6.10)$$

with $\operatorname{erf}(\dots)$ being the well-known *error function*.

However, particles may also leave the bin \mathbf{x}_i . There are currently n_i^k particles in the bin \mathbf{x}_i . Each other bin \mathbf{x}_j has a $\frac{1}{n}$ chance of challenging each of these particles, and subsequently a chance of P_{ji} of winning. As such, the expected number of particles leaving bin \mathbf{x}_i equals

$$\text{Expected particles leaving } \mathbf{x}_i: n_i^k \frac{1}{n} \sum_{j=1}^n P_{ji}. \quad (6.11)$$

From this it follows that

$$n_i^{k+1} = n_i^k + \frac{1}{n} \sum_{j=1}^n (P_{ij} n_j^k - P_{ji} n_i^k). \quad (6.12)$$

Note that in the sum we should actually ignore $j = i$, but even if we do sum over it, the corresponding summation term will become zero. This does not even depend on the value which we would assign to P_{ii} , even though I usually define $P_{ii} = \frac{1}{2}$.

Next, let's lump all values n_1^k, \dots, n_n^k into the vector \mathbf{n}^k and let's lump all probabilities P_{ij} into the matrix P . Note that P is not a symmetric matrix, but we instead have $P + P^T = 1_n$, with 1_n being the $n \times n$ matrix in which all elements equal 1. By carefully considering (6.12), and which terms we are multiplying and adding up, we can now write

$$\mathbf{n}^{k+1} = \mathbf{n}^k + \frac{1}{n} (P - \operatorname{diag}(1_n P)) \mathbf{n}^k. \quad (6.13)$$

In the limit case we will have a *stationary distribution* $\bar{\mathbf{n}}$ of particles. As such, we then have $\mathbf{n}^{k+1} = \mathbf{n}^k = \bar{\mathbf{n}}$. This can only happen if the last term in the above expression is zero. Hence, we must have

$$(P - \operatorname{diag}(1_n P)) \bar{\mathbf{n}} = \mathbf{0}. \quad (6.14)$$

We cannot use this expression directly to find $\bar{\mathbf{n}}$, because the above matrix is not invertible. To be precise, it is of rank $n - 1$. However, we also know that the sum of all particles in all bins equals the total number of particles n_p . That is,

$$\mathbf{1}^T \bar{\mathbf{n}} = n_p. \quad (6.15)$$

If we replace the bottom row of (6.14) by this expression, then we do have an invertible matrix and we can calculate the limit distribution of the particles. This limit distribution is also shown in Figure 6.5.

From Figure 6.5 we can see that the limit distribution does not equal the true maximum distribution. It is slightly less peaky. This means that the algorithm is somewhat less certain of where the maximum will be, but it is aware of this uncertainty. In addition, as the Gaussian process gets more measurement data and hence becomes more accurate, also the maximum distribution will become more and more accurate, reducing the effects of this problem.

6.1.6. AN INTUITIVE VIEW ON THE TWO DIFFERENT DISTRIBUTIONS

Let's ask ourselves, 'Why is the limit distribution different from the true maximum distribution?' To figure this out, we will consider an example problem.

Consider the Gaussian random variables \underline{f}_1 , \underline{f}_2 and \underline{f}_3 . We assume that \underline{f}_1 and \underline{f}_2 are always equal, except for a tiny bit of noise. That is, $\underline{f}_2 = \underline{f}_1 + \underline{v}$, with $\underline{v} \sim \mathcal{N}(0, \varepsilon^2)$ and where ε is a very small number. On the flip side, \underline{f}_3 is independent from all other parameters. If we furthermore take the means as zero and the variances as one, then we get the distribution

$$\begin{bmatrix} \underline{f}_1 \\ \underline{f}_2 \\ \underline{f}_3 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 + \varepsilon^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right). \quad (6.16)$$

We can reason for ourselves what the true maximum distribution will be. In 50% of the cases \underline{f}_3 will be larger than $\underline{f}_1 \approx \underline{f}_2$ and hence be the maximum. In the other 50% of the cases, either \underline{f}_1 or \underline{f}_2 will be the largest, depending on the noise \underline{v} . So \underline{f}_1 and \underline{f}_2 each are the maximum in 25% of the cases.

However, now let's consider the matrix P . Any function value \underline{f}_i has a 50% chance to be bigger than any other function value \underline{f}_j . As such, P equals

$$P = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}, \quad (6.17)$$

which eventually results in a stationary particle distribution of

$$\bar{\mathbf{n}} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} n. \quad (6.18)$$

Based on this, the particle method is incapable of taking into account covariances between other function values. That is, if \underline{f}_3 happens to be larger than \underline{f}_2 in a certain case, the algorithm cannot determine that \underline{f}_3 then automatically is also larger than \underline{f}_1 . This leads to a lower estimated probability that \underline{f}_3 is the maximum.

This problem is inherent in the algorithm and in the simplifications that we had to make. However, as we saw in Figure 6.5, for most Gaussian processes it just results in a slightly less peaky maximum distribution, which is not a significant problem. And there also is a way in which we could reduce this effect.

6.1.7. USING MULTIPLE CHALLENGERS

We have found that the limit distribution of the particles does not equal the true maximum distribution of the Gaussian process. We can get the limit distribution closer to the true maximum distribution by adjusting the MCMD algorithm. Instead of challenging every champion particle by a single challenger, we now challenge it by n_c challengers. So for each of the n_p particles \mathbf{x}_i , we pick n_c random bins $\mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_{n_c}}$ and set up the joint distribution

$$\begin{bmatrix} \underline{f}(\mathbf{x}_i) \\ \underline{f}(\mathbf{x}_{j_1}) \\ \vdots \\ \underline{f}(\mathbf{x}_{j_{n_c}}) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} f_i \\ f_{j_1} \\ \vdots \\ f_{j_{n_c}} \end{bmatrix} \mid \begin{bmatrix} \mu(\mathbf{x}_i) \\ \mu(\mathbf{x}_{j_1}) \\ \vdots \\ \mu(\mathbf{x}_{j_{n_c}}) \end{bmatrix}, \begin{bmatrix} \Sigma(\mathbf{x}_i, \mathbf{x}_i) & \Sigma(\mathbf{x}_i, \mathbf{x}_{j_1}) & \cdots & \Sigma(\mathbf{x}_i, \mathbf{x}_{j_{n_c}}) \\ \Sigma(\mathbf{x}_{j_1}, \mathbf{x}_i) & \Sigma(\mathbf{x}_{j_1}, \mathbf{x}_{j_1}) & \cdots & \Sigma(\mathbf{x}_{j_1}, \mathbf{x}_{j_{n_c}}) \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma(\mathbf{x}_{j_{n_c}}, \mathbf{x}_i) & \Sigma(\mathbf{x}_{j_{n_c}}, \mathbf{x}_{j_1}) & \cdots & \Sigma(\mathbf{x}_{j_{n_c}}, \mathbf{x}_{j_{n_c}}) \end{bmatrix} \right). \quad (6.19)$$

We then take a sample from this distribution, find the maximum of this sample vector, and put the particle in the corresponding bin.

Note that in the limit case, if we set the number of challengers n_c equal to the number of bins n , and if we ensure that all challengers are from different bins, then we get the true maximum distribution. This even happens after a single round of challenges. Naturally, for continuous problems with infinitely many bins n , this will not be possible, but this does show that if we take a sufficient number of challengers, we get closer and closer to the true maximum distribution. This is also confirmed by Figure 6.6, which shows that by using multiple challengers we not only get faster convergence but also converge to a distribution closer to the true maximum distribution.

For the rest of this thesis, for reasons of simplicity, we will use only $n_c = 1$ challenger. We can always implement more challengers in the case where we wind up with an incorrect distribution of particles.

6.2. INTERMEZZO: SEQUENTIAL MONTE CARLO SAMPLERS

We can improve the MCMD algorithm significantly by applying ideas from *sequential Monte Carlo samplers* (SMC samplers). To do so, we first need to know how such samplers work. This is explained well by the books of Owen (2013) and Schön and Lindsten (2017). However, this section provides you with a quick introduction into the subject, with most of the theory taken from either of these books.

We start by examining the main idea behind Monte Carlo methods (Section 6.2.1). Then we introduce a technique called importance sampling (Section 6.2.2) and extend

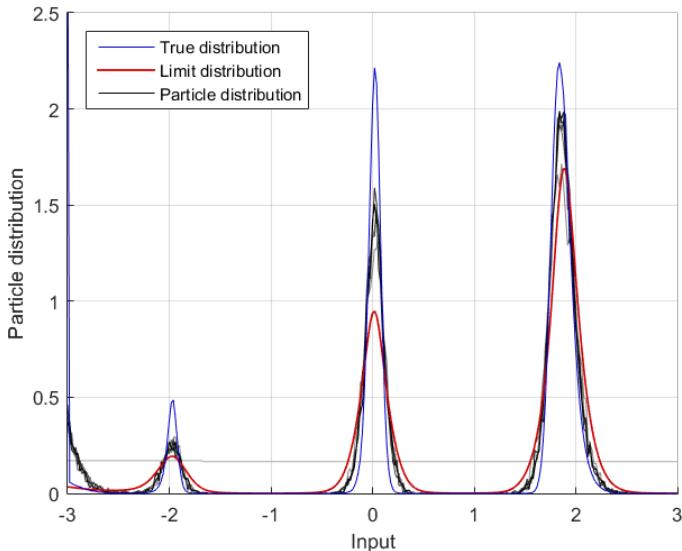


Figure 6.6: The development of the distribution of particles over $n_r = 5$ rounds when we use $n_c = 30$ challengers per round. Shown are the true maximum distribution from Figure 6.2, the limit distribution of the particles if we would use only $n_c = 1$ challenger per round, and the distribution of the particles over each of the rounds. The darker the line, the later the round is. Note that the final distribution of particles converges faster and also converges to a limit distribution closer to the true maximum distribution, as compared to the previous limit distribution which is still shown for comparison. (Calculating the new limit distribution analytically is impossible.)

it to self-normalized importance sampling (Section 6.2.3). We then extend the ideas to multiple time steps through sequential importance sampling (Section 6.2.4). Subsequently, we solve a few problems which might occur by introducing the concepts of resampling (Section 6.2.5), mixture importance sampling (Section 6.2.6) and defensive importance sampling (Section 6.2.7).

6.2.1. THE IDEA BEHIND MONTE CARLO METHODS

Suppose that we have some system with state equation

$$\underline{x}_{k+1} = f(\underline{x}_k). \quad (6.20)$$

Here the *state transition function* $f(\underline{x}_k)$ does not even have to be a deterministic system. It can also be subject to process noise, such that inserting the same input \underline{x}_k does not always give the same output \underline{x}_{k+1} . However, we do assume that we know this state transition function $f(\underline{x}_k)$, including the distribution of the noise.

The main question we now want to ask is, ‘If \underline{x}_k is a random variable, distributed according to $p(\underline{x}_k)$, what is then the distribution $p(\underline{x}_{k+1})$ of \underline{x}_{k+1} ?’

For arbitrary distributions $p(\underline{x}_k)$ and arbitrary functions $f(\underline{x}_k)$, there is no general analytical solution. There is something else we can do though. We can pick a large number n_p of samples $\underline{x}_k^1, \dots, \underline{x}_k^{n_p}$ from the distribution $p(\underline{x}_k)$. We will call these *particles*. For each of them, we calculate $\underline{x}_{k+1}^1, \dots, \underline{x}_{k+1}^{n_p}$ through (6.20). Bluntly put, we just pick lots of random initial states from our known distribution and see what the outcome will be. This type of methods is known as *Monte Carlo*² methods.

Based on the particles $\underline{x}_{k+1}^1, \dots, \underline{x}_{k+1}^{n_p}$, we can now set up the distribution of \underline{x}_{k+1} . Officially the result would be the probability density function

$$\underline{x}_{k+1} \sim p(\underline{x}_{k+1}) = \frac{1}{n_p} \sum_{i=1}^{n_p} \delta(\underline{x}_{k+1} - \underline{x}_{k+1}^i). \quad (6.21)$$

This posterior distribution has a lot of advantages when we want to calculate things with it. For instance, if we want to find the expected value of $h(\underline{x}_{k+1})$ for any transforming function $h(\dots)$, it follows as

$$\begin{aligned} \mathbb{E}[h(\underline{x}_{k+1})] &= \int_X h(\underline{x}_{k+1}) p(\underline{x}_{k+1}) d\underline{x}_{k+1} \\ &= \frac{1}{n_p} \sum_{i=1}^{n_p} \int_X h(\underline{x}_{k+1}) \delta(\underline{x}_{k+1} - \underline{x}_{k+1}^i) d\underline{x}_{k+1} \\ &= \frac{1}{n_p} \sum_{i=1}^{n_p} h(\underline{x}_{k+1}^i). \end{aligned} \quad (6.22)$$

The probability density function (6.21) is not very suitable for making nice plots though. After all, we are plotting very spiky delta functions. We would get a result like that from Figure 6.7 (top right).

²The name ‘Monte Carlo’ originated as a code name for the method during the Manhattan Project in the 1940s, where such methods were central to the simulations that were performed. The name itself refers to the Monte Carlo casino in Monaco.

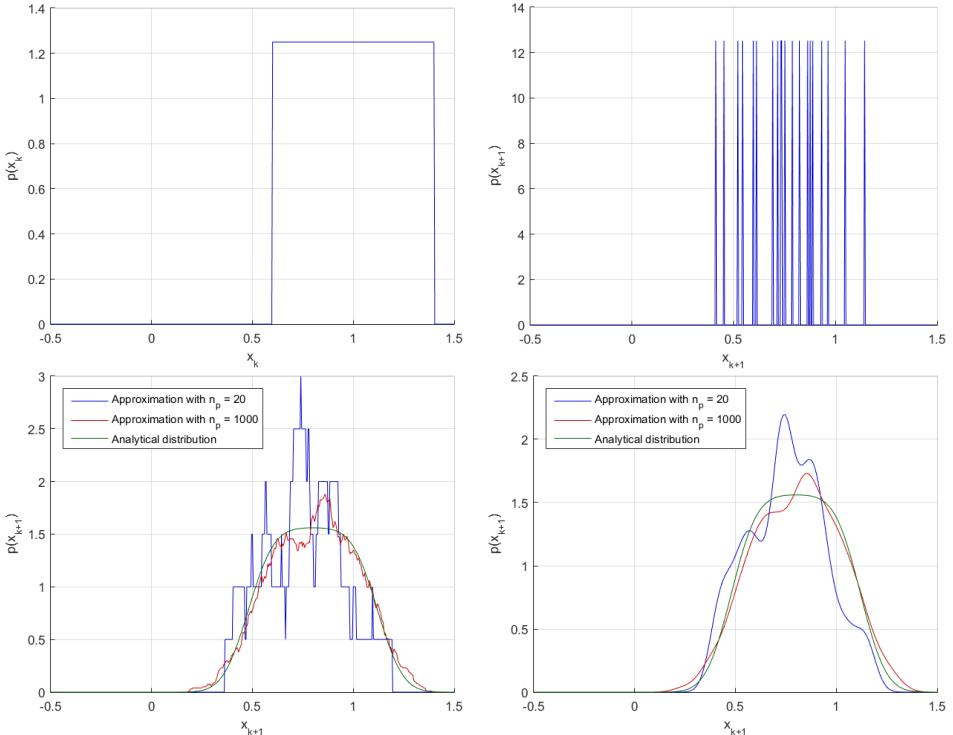


Figure 6.7: An application of calculating the distribution of the next state of a system. We start with the distribution of the initial state x_k (top left), which we (quite arbitrarily) choose to be uniform. We know its properties. We then sample n_p particles from this initial distribution. We use either $n_p = 20$ or $n_p = 1000$ in this example. For all these particles x_k we calculate $x_{k+1} = f(x_k) = 0.8x_k + \underline{v}$, where we have $\underline{v} \sim \mathcal{N}(0, 0.1^2)$. We then use these particles to approximate the distribution of x_{k+1} . We either do this through the delta function (top right), a block kernel function with width $w = 0.1$ (bottom left) or a Gaussian kernel function with width $\sigma_x = 0.05$ (bottom right). Also the analytical distribution of the next state has been calculated and plotted as comparison.

To solve this issue, we can use a technique known as *Kernel Density Estimation* (KDE). In this method we use a kernel³ $k_x(\mathbf{x}_{k+1}, \mathbf{x}_{k+1}^i)$ to estimate the probability density as

$$\underline{\mathbf{x}}_{k+1} \sim p(\mathbf{x}_{k+1}) = \frac{1}{n_p} \sum_{i=1}^{n_p} k_x(\mathbf{x}_{k+1}, \mathbf{x}_{k+1}^i). \quad (6.23)$$

As kernel function, we can use a block function with some width w , a Gaussian exponential function $\mathcal{N}(\mathbf{x}_{k+1} | \mathbf{x}_{k+1}^i, \Sigma_x)$, or something else. Pretty much any kernel can be used, although we do want to use kernel functions whose integral equals one, to get a proper probability density functions. Some possible results are shown in Figure 6.7.

It is worthwhile to note here that, as you get more particles, your approximation becomes more accurate. At the same time, as you get more particles, you can also reduce the width of your kernel function, up to the point where you graph stops being nice and smooth.

6.2.2. IMPORTANCE SAMPLING

Let's take a look at the distribution $p(\mathbf{x}_k)$ of the initial state. It may happen that there is a certain region χ in the state space where the states are very important, while the probability $p(\mathbf{x}_k)$, with $\mathbf{x}_k \in \chi$, is very small. We call these *rare events*. If we sample our particles from $p(\mathbf{x}_k)$, then we are unlikely to get particles representing these rare events at all. This results in an incomplete picture. How can we solve this?

The key is to not draw our samples from the distribution $p(\mathbf{x}_k)$, but use another distribution $q(\mathbf{x}_k)$ to sample our particles from. For instance, we can take a distribution $q(\mathbf{x}_k)$ which gives more particles representing rare events. There is one small problem with this though.

Suppose that we want to find the mean of $\underline{\mathbf{x}}_k$. This mean is formally defined as

$$\mathbb{E}[\underline{\mathbf{x}}_k] = \int_X \mathbf{x}_k p(\mathbf{x}_k) d\mathbf{x}_k. \quad (6.24)$$

We can also find the mean from our particles $\mathbf{x}_k^1, \dots, \mathbf{x}_k^{n_p}$. This is done by applying (6.21), similarly to what we did at (6.22). In our old set-up it would follow that

$$\mathbb{E}[\underline{\mathbf{x}}_k] = \int_X \mathbf{x}_k p(\mathbf{x}_k) d\mathbf{x}_k = \frac{1}{n_p} \sum_{i=1}^{n_p} \int_X \mathbf{x}_k \delta(\mathbf{x}_k - \mathbf{x}_k^i) d\mathbf{x}_k = \frac{1}{n_p} \sum_{i=1}^{n_p} \mathbf{x}_k^i. \quad (6.25)$$

If we use the new set-up, and select our particles from $q(\mathbf{x}_k)$, we wind up with a different result though. The key here is to instead write

$$\mathbb{E}[\underline{\mathbf{x}}_k] = \int_X \mathbf{x}_k \frac{p(\mathbf{x}_k)}{q(\mathbf{x}_k)} q(\mathbf{x}_k) d\mathbf{x}_k = \int_X \mathbf{x}_k w(\mathbf{x}_k) q(\mathbf{x}_k) d\mathbf{x}_k, \quad (6.26)$$

where we have defined the *weight* $w(\mathbf{x}_k)$ to equal $\frac{p(\mathbf{x}_k)}{q(\mathbf{x}_k)}$. From this comes the idea of adding weights to the particles, also known as *importance sampling*.

³We use the notation $k_x(\dots)$ here, instead of $k(\dots)$, to prevent confusion with the Gaussian process covariance function.

Let's take a look at how exactly it works. Suppose that we want to find the posterior distribution of \underline{x}_{k+1} . To find this, we sample particles $\underline{x}_k^1, \dots, \underline{x}_k^{n_p}$ from $q(\underline{x}_k)$. To every particle we attach a weight $w^i = w(\underline{x}_k^i)$. Subsequently, we feed all particles $\underline{x}_k^1, \dots, \underline{x}_k^{n_p}$ through the state transition function $f(\underline{x}_k)$ to get the particles in the next time step $\underline{x}_{k+1}^1, \dots, \underline{x}_{k+1}^{n_p}$. The posterior distribution of \underline{x}_{k+1} can now be approximated through

$$\underline{x}_{k+1} \sim p(\underline{x}_{k+1}) = \frac{1}{n_p} \sum_{i=1}^{n_p} w^i k_x(\underline{x}_{k+1}, \underline{x}_{k+1}^i). \quad (6.27)$$

It is important to note that events which we emphasize through $q(\underline{x}_k)$ (like the rare events) will get more particles than before, but these particles subsequently have a lower weight. As such, we will still find the correct posterior distribution.

6.2.3. SELF-NORMALIZED IMPORTANCE SAMPLING

Let's consider these weights $w(\underline{x}_k)$. What kind of values can they have? Naturally, if $q(\underline{x}_k)$ is very large or very small for certain states \underline{x}_k , these weights can have very small or very large values as well. The average value for these weights, however, is

$$\mathbb{E}[w(\underline{x}_k)] = \int_X w(\underline{x}_k) q(\underline{x}_k) d\underline{x}_k = \int_X \frac{p(\underline{x}_k)}{q(\underline{x}_k)} q(\underline{x}_k) d\underline{x}_k = \int_X p(\underline{x}_k) d\underline{x}_k = 1. \quad (6.28)$$

6

So the sum of all n_p weights on average equals n_p . Note that this is an average. It may happen that we have selected more rare events than expected, and the sum of all weights will be a bit lower, or we have selected less rare events, and the sum will be higher. Nevertheless, the sum of all weights has an expected value of n_p , which is an important insight to keep in mind.

Now let's make our problem a bit harder. Suppose that we do not know the prior distribution $p(\underline{x}_k)$ itself, but we do know it up to a proportionality constant. That is, we can calculate the value of $\tilde{p}(\underline{x}_k) = C p(\underline{x}_k)$, but both C and $p(\underline{x}_k)$ are unknown. In addition, we cannot just integrate over $\tilde{p}(\underline{x}_k)$ to find C , because that would require too many computations. How can we now still get our algorithm working?

The key here is to use our knowledge that the weights on average equal one, and to manually enforce this. To accomplish this, we first redefine the weights $w(\underline{x}_k)$ as

$$w(\underline{x}_k) = \frac{\tilde{p}(\underline{x}_k)}{q(\underline{x}_k)} = C \frac{p(\underline{x}_k)}{q(\underline{x}_k)}. \quad (6.29)$$

So every time we sample a particle \underline{x}_k^i , we use the above equation to calculate its weight w^i . Next, we need to enforce that our weights on average equal one. In other words, we normalize the weights according to⁴

$$\tilde{w}^i \equiv \frac{w^i}{\bar{w}}, \quad (6.30)$$

⁴In some literature the weights are normalized such that their *sum* equals one. We normalize the weights such that their *sum* equals n_p and their *mean value* equals one.

where the \tilde{w} notation means ‘weights after normalization’ and the mean weight \bar{w} equals

$$\bar{w} \equiv \frac{1}{n_p} \sum_{i=1}^{n_p} w^i. \quad (6.31)$$

The approximation of the distribution of \underline{x}_{k+1} now becomes

$$\underline{x}_{k+1} \sim p(\underline{x}_{k+1}) = \frac{1}{n_p} \sum_{i=1}^{n_p} \tilde{w}^i k_x(\underline{x}_{k+1}, \underline{x}_{k+1}^i) = \frac{\sum_{i=1}^{n_p} w^i k_x(\underline{x}_{k+1}, \underline{x}_{k+1}^i)}{\sum_{i=1}^{n_p} w^i}. \quad (6.32)$$

Note that, irrespective of which value C might have, it will drop out of the equations. This method – normalizing the weights by making sure they are on average equal to one – is known as *self-normalized importance sampling*.

6.2.4. SEQUENTIAL IMPORTANCE SAMPLING

We now know how to find the distribution of \underline{x}_{k+1} given the distribution of \underline{x}_k . Or equivalently, we know how to find the distribution of \underline{x}_k given the distribution of \underline{x}_{k-1} . Next, suppose that we perform a state measurement according to

$$\underline{y}_k = g(\underline{x}_k) + \underline{v}, \quad (6.33)$$

with $g(\dots)$ the known *output function* and $\underline{v} \sim \mathcal{N}(\mathbf{0}, \Sigma_y)$ being Gaussian white output noise, where Σ_y is also known. How can we take this measurement into account in our distribution of particles?

The key here is to apply Bayes’ law. The probability that we have some state \underline{x}_k , given that we have obtained a measurement \underline{y}_k , equals

$$p(\underline{x}_k | \underline{y}_k) = \frac{p(\underline{y}_k | \underline{x}_k) p(\underline{x}_k)}{p(\underline{y}_k)}. \quad (6.34)$$

We can apply this law to the distribution of the state \underline{x}_k . The left-hand-side term $p(\underline{x}_k | \underline{y}_k)$ is the *posterior state distribution* which we want to find. We know the *observation likelihood* $p(\underline{y}_k | \underline{x}_k)$ from (6.33) and the *prior state distribution* $p(\underline{x}_k)$ from (6.32). It follows that

$$p(\underline{x}_k | \underline{y}_k) = \frac{p(\underline{y}_k | \underline{x}_k)}{p(\underline{y}_k)} \frac{1}{n_p} \sum_{i=1}^{n_p} \tilde{w}^i k_x(\underline{x}_{k+1}, \underline{x}_{k+1}^i). \quad (6.35)$$

From this we can see that incorporating our measurement comes down to adjusting the weights. Ideally, we would use as weight update law

$$\tilde{w}^i \leftarrow \tilde{w}^i \frac{p(\underline{y}_k | \underline{x}_k)}{p(\underline{y}_k)}. \quad (6.36)$$

However, this is generally not possible, because the *marginal likelihood* $p(\underline{y}_k)$ is not known. It is a constant though, not depending on \underline{x}_k . As such, we can apply the self-normalization trick again, first defining

$$w^i \leftarrow \tilde{w}^i p(\underline{y}_k | \underline{x}_k), \quad (6.37)$$

and then normalizing the weights through

$$\tilde{w}^i \leftarrow \frac{w^i}{\bar{w}} = \frac{w^i}{\sum_{i=1}^{n_p} w^i}. \quad (6.38)$$

By updating the weights in this way, we can take into account measurements of the state.

6.2.5. RESAMPLING

There is a problem with the set-up we have now. To see what it is, we need to do a thought experiment.

Suppose that we have some unknown state \underline{x}_k which we want to find. Initially, we do not have any clue what this state is, so when we choose particles $\underline{x}_0^1, \dots, \underline{x}_0^{n_p}$ to approximate the distribution of \underline{x}_0 , we choose them all over the state space. Over time, the state transforms to $\underline{x}_1, \underline{x}_2, \dots$ and we get the corresponding measurements $\underline{y}_1, \underline{y}_2, \dots$. We use these measurements to adjust the weights of our particles. Particles which do not correspond to the measurements (which are most of them) get a lower weight, ultimately reducing to zero, while particles which do correspond to the measurements (hopefully at least one) will get a higher weight.

The problem is that, after some time, most particles will have a weight of zero and ultimately become useless, while we will have one or (with some luck) a few particles with a very high weight. Naturally, these few particles cannot describe the full state distribution by themselves. This problem is known as *weight degeneracy*. The solution is that we need to get rid of particles with zero weights and add more particles near those with high weights. This idea is called *resampling*.

When applying resampling, we effectively choose a new set of particles $\tilde{\underline{x}}_k^1, \dots, \tilde{\underline{x}}_k^{n_p}$ out of the old particles $\underline{x}_k^1, \dots, \underline{x}_k^{n_p}$. The $\tilde{\underline{x}}$ notation here denotes the resampled particles. Old particles with high weights are likely to be picked multiple times, while old particles with a zero weight will not be picked at all. After doing this resampling, all new particles will have the same weight 1.

But how does this ‘picking’ process work in detail? There are actually many ways to do resampling. To demonstrate them, we consider a simple example problem. Suppose that we have five particles $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$ and $x_5 = 5$ with weights w^i of 0.1, 0.4, 1, 1.5 and 2, respectively. Note that all the weights add up to the number of particles $n_p = 5$. In fact, we can plot the weights in a *cumulative weight chart* as in Figure 6.8 (top left).

The first thing we can do is randomly pick $n_p = 5$ new particles, where the chance that we pick a particle x_i is proportional to the weights w^i . So the chance that we pick (for instance) x_4 equals $\frac{w_4}{n_p} = \frac{1.5}{5} = 30\%$. This process is known as *multinomial resampling* and has been visualized in Figure 6.8 (top right).

The problem with this approach is that there is a large degree of randomness. In the example selection of Figure 6.8 (top right), we wind up with new particles [2, 4, 5, 5, 5]. Note that we have lost the old particle 1 (rightfully so) and the old particle 3 (not very rightfully so).

To reduce the randomness, we could divide the vertical axis from Figure 6.8 into n_p blocks and randomly pick a particle from each block. This idea, called *stratified resam-*

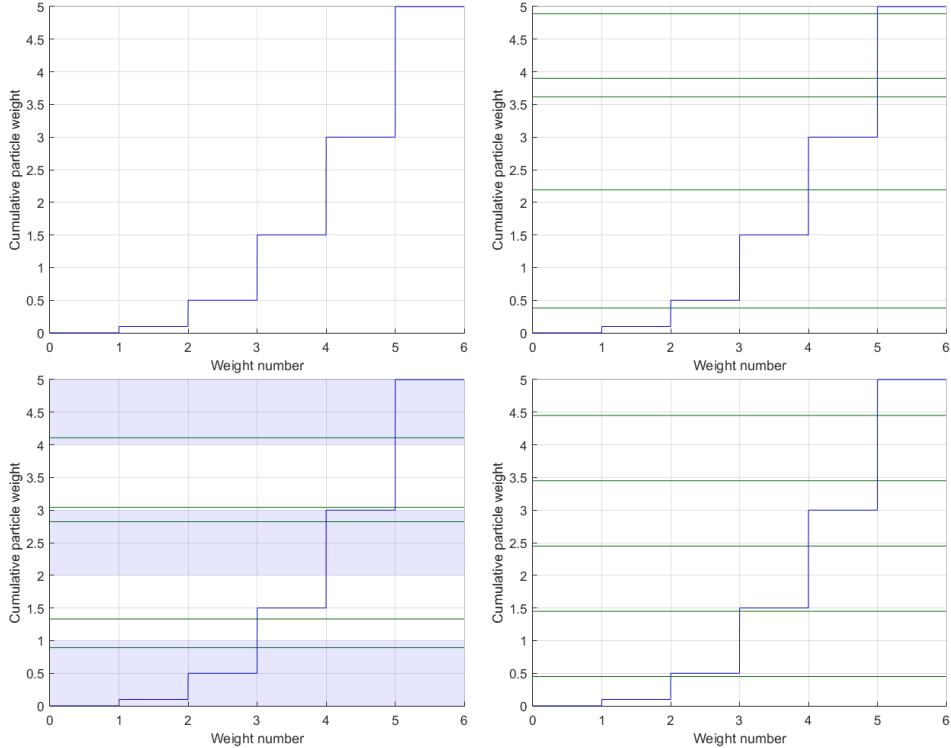


Figure 6.8: An example resampling problem. We have a cumulative weight chart (top left). When we take n_p random points along the vertical axis and choose the corresponding particles, we are applying multinomial resampling (top right). When we split the vertical axis up into n_p equally sized blocks and take random points in each block, we apply stratified resampling. And when we pick the same point in each of the n_p blocks, we are using systematic resampling. Note that in the first two resampling algorithms we have to use n_p random numbers, while the latter algorithm only requires one random number, making the algorithm a lot less random.

pling, is shown in Figure 6.8 (bottom left).

In this new approach, we are (rightfully) guaranteed to get particle 5 twice. However, it may still happen that we get particle 2 twice, while we only get particle 3 once. This seems odd, and it can be solved by adjusting the strategy slightly.

Instead of picking random points from each block, we now only pick a random point from the first block, and put all other points at exactly the same point in their respective blocks. This process is referred to as *systematic resampling* and is visualized in Figure 6.8 (bottom right).

Because systematic resampling seems to be the most fair approach with the least degree of randomness involved, we will use it for the rest of this chapter.

6.2.6. MIXTURE IMPORTANCE SAMPLING

An interesting property of resampling is that we suddenly wind up with multiple particles \tilde{x}_k^j at exactly the same location. We could ask ourselves, ‘Is this a problem?’

The answer actually depends on the state transition function $f(\underline{x}_k)$. If this function has some process noise in it, then this is not much of a problem. At the next time step, when we calculate \underline{x}_{k+1} , the particles will all be different. They automatically spread out across possible state values. If $f(\underline{x}_k)$ is deterministic though, then all these particles will remain the same. In this case it is rather pointless to keep track of a (possibly large) number of particles which all do the exact same thing. As such, we need to change something in our method.

To see how we can fix this problem, we need to look at what resampling actually comes down to. When we apply (multinomial) resampling, we effectively draw each new particle \tilde{x}_k^j from the distribution

$$p(\tilde{x}_k^j) = \frac{1}{n_p} \sum_{i=1}^{n_p} \tilde{w}^i \delta(\tilde{x}_k^j - \underline{x}_k^i). \quad (6.39)$$

However, when we wanted to plot the distribution of the unknown state \underline{x}_k , we did not use the above expression. Instead, we used the kernel density estimate

$$\underline{x}_k \sim p(\underline{x}_k) = \frac{1}{n_p} \sum_{i=1}^{n_p} \tilde{w}^i k_x(\underline{x}_k, \underline{x}_k^i), \quad (6.40)$$

for some properly chosen kernel function $k_x(\dots, \dots)$. As such, we could also apply resampling by picking our new samples from the above distribution. Since the above distribution is a summation, or mixture, of various different distributions/kernels, this technique is known as *mixture importance sampling*.

In practice, when we apply this mixture importance sampling, we perform two steps. Suppose that we are choosing a new particle \tilde{x}_k^j . The first step is essentially the same as what we did before: we pick one of our old particles \underline{x}_k^i , where particles with high weights w^i are more likely to be chosen. We call \underline{x}_k^i the *ancestor* of \tilde{x}_k^j of \underline{x}_k^i . In the second step, we now do *not* set \tilde{x}_k^j equal to \underline{x}_k^i , but instead sample it from $k_x(\tilde{x}_k^j, \underline{x}_k^i)$. This generally causes \tilde{x}_k^j to be close to its ancestor, but not exactly equal to it.

6.2.7. DEFENSIVE IMPORTANCE SAMPLING

We use our set of particles \tilde{x}_k^i to approximate the distribution of the state x_k . Now, let's suppose that we have made some mistake in choosing our initial distribution of particles: the true state x_k is *not* among our particles \tilde{x}_k^i , or even close to one. What happens then?

In the usual version of our algorithm, we are then unlikely to find the true state. It may happen that the process noise within the state transition function $f(x_k)$, or alternatively the kernel $k_x(\tilde{x}_k^j, x_k^i)$ within our mixture importance sampling, accidentally puts a wrong particle in the right place, but that is extremely unlikely. Instead, it is also possible to defend ourselves from this by applying *defensive importance sampling*.

The idea is that most of the time, in a large part α , we still sample our new set of particles from the KDE (6.40) in the usual way. However, in a small part $1 - \alpha$ of the cases, as some sort of 'extra security', we sample from a completely different distribution $q(\tilde{x}_k^j)$ instead, where this distribution is set up to include all possible values of the state x_k . As such, we are now sampling \tilde{x}_k^j from

$$q'(\tilde{x}_k^j) = (1 - \alpha) q(\tilde{x}_k^j) + \alpha \frac{1}{n_p} \sum_{i=1}^{n_p} \tilde{w}^i k_x(\tilde{x}_k^j, x_k^i). \quad (6.41)$$

When we apply this, we do distort our posterior distribution of particles. As such, we need to compensate for this using weights. Previously, resampling caused all weights to be equal to one again. Now this is not the case. Instead, according to the idea of importance sampling, the weight w^j of a new particle \tilde{x}_k^j will be equal to

$$w^j = \frac{p(\tilde{x}_k^j)}{q'(\tilde{x}_k^j)} = \frac{\frac{1}{n_p} \sum_{i=1}^{n_p} \tilde{w}^i k_x(\tilde{x}_k^j, x_k^i)}{(1 - \alpha) q(\tilde{x}_k^j) + \alpha \frac{1}{n_p} \sum_{i=1}^{n_p} \tilde{w}^i k_x(\tilde{x}_k^j, x_k^i)}. \quad (6.42)$$

This idea works well, except for one small issue. To calculate the weight w^j of a new particle, we need to sum over n_p terms. Calculating the weight w^j of a single particle hence takes $\mathcal{O}(n_p)$ time, and resampling all particles takes $\mathcal{O}(n_p^2)$ time. Since we want to use large numbers of particles to get the highest accuracy possible, this is unacceptable.

The solution lies in applying a mindshift. We just noted that resampling consists of two steps: first we select an old particle x_k^i (step 1) and then we sample a new one from the kernel $k_x(\tilde{x}_k^j, x_k^i)$ (step 2). In defensive importance sampling we added an extra rule prior to this whole scheme. 'In $(1 - \alpha)$ part of the cases, sample from $q(\tilde{x}_k^j)$ instead.' Instead of applying this extra rule *before* these two steps, we will now put it *between* them. So, we first select an old particle x_k^i . Then, in a part α of the cases, we select a new particle from the kernel $k_x(\tilde{x}_k^j, x_k^i)$, while in a part $(1 - \alpha)$ of the cases, we ignore this old particle x_k^i and select a new particle from $q(\tilde{x}_k^j)$. According to this mindshift, instead of using expression (6.42) for the weights w^j , we can also use

$$w^j = \frac{p(\tilde{x}_k^j | x_k^i)}{q'(\tilde{x}_k^j | x_k^i)} = \frac{k_x(\tilde{x}_k^j, x_k^i)}{(1 - \alpha) q(\tilde{x}_k^j) + \alpha k_x(\tilde{x}_k^j, x_k^i)}. \quad (6.43)$$

Here, the notation ' $|\mathbf{x}_k^i$ ' can be read as 'Given that we have picked \mathbf{x}_k^i as ancestor.' Note that the above expression can be calculated in $\mathcal{O}(1)$ (constant) time for each particle, solving our problem. As such, our algorithm remains efficient.

This technique of defensive importance sampling guarantees that, if we just iterate long enough, and if the correct state \mathbf{x}_k can be sampled from $q(\mathbf{x}_k)$ with nonzero probability, we are guaranteed to eventually find it. So even if our initial set of particles is bad, we can fix this along the way.

6.3. APPLYING SMC IDEAS TO FIND THE MAXIMUM

In this section we will use the SMC ideas we just picked up to improve the MCMD algorithm from Section 6.1.4. Keep in mind here that we mainly use SMC for inspiration. The resulting method will not actually be an SMC method, because the target distribution cannot be calculated analytically; not even up to a proportionality constant.

We start by getting all our notations and definitions clear (Section 6.3.1). We then improve the convergence rate of the algorithm (Section 6.3.2), fix some issues by adding weights to particles (Section 6.3.3), discuss the resampling of particles (Section 6.3.4), adding a trick to ensure we always eventually find the correct optimum (Section 6.3.5). Finally, we enable the algorithm to not only deal with discrete functions but also with continuous functions (Section 6.3.6), and look at how we can find the corresponding distribution of the optimal function value \underline{f}^* (Section 6.3.7).

6

6.3.1. NOTATIONS AND DEFINITIONS

Before we start, we should make it clear what exact problem we are working on. We are going to start with an easy discrete problem and slowly build it up to the full continuous problem.

We start with the problem where the input \mathbf{x} can only take a finite number n possible values, which we denote by $\mathbf{x}_1, \dots, \mathbf{x}_n$. As a result, we can set up n bins, which we also name $\mathbf{x}_1, \dots, \mathbf{x}_n$, into which we can put particles.

We will use a number of n_p particles in the MCMD algorithm. We write these as $\mathbf{x}^1, \dots, \mathbf{x}^{n_p}$. Each of these particles \mathbf{x}^j will be set equal to some possible input value \mathbf{x}_i .

Each of the existing (champion) particles will be challenged by challenger particles. When we have a challenger particle challenging \mathbf{x}^j , then we will denote the challenger particle as \mathbf{x}_c^j , which means as much as 'the particle challenging \mathbf{x}^j '. If we are talking about a generic challenger particle, we just write it as \mathbf{x}_c .

Eventually our goal is to approximate the maximum distribution. We will write this distribution as $p_m(\mathbf{x}) \equiv p(\mathbf{x} = \mathbf{x}^*)$. It is the chance that a given input point \mathbf{x} equals the optimal input point \mathbf{x}^* . For now, the function $p_m(\mathbf{x})$ equals the number of particles \mathbf{x}^j that are equal to the given input point \mathbf{x} . We can write this as

$$p(\mathbf{x}) = \frac{1}{n} \sum_{j=1}^{n_p} \delta(\mathbf{x}, \mathbf{x}^j), \quad (6.44)$$

where $\delta(\mathbf{x}, \mathbf{x}^j)$ here is the discrete delta function: it equals one when $\mathbf{x} = \mathbf{x}^j$ and zero otherwise. So effectively, the above sum counts the number of particles that equals \mathbf{x} , and the probability $p(\mathbf{x})$ is the part of all particles that are in bin \mathbf{x} .

6.3.2. IMPROVING THE CONVERGENCE RATE

Now let's consider the algorithm from Section 6.1.4. How long does it take for the distribution of particles to converge to the limit distribution? Naturally, the answer to that question depends on this limit distribution.

Let's consider a relatively simple example problem. Suppose that the input x can only be an integer number $1, 2, \dots, 10$. In addition, we know with full certainty that $f(7)$ is the maximum. If we use $n_p = 100$ particles, how long will it take before all particles are in bin $x = 7$?

The problem is that challenger particles are taken randomly from all possible inputs $1, \dots, 10$, so only a tenth of the challenger particles will be at $x = 7$. So only a tenth of the challenges may cause a particle to shift from an incorrect bin to the correct bin. This causes slow convergence. And in many practical applications this problem is far worse, because we have many more bins than just 10.

The solution is to sample our challenger particles based on where we think that the maximum is. In other words, we sample the next challenger particle \mathbf{x}_c from the current belief of the maximum distribution $p_m(\mathbf{x})$. To do so, we could calculate $p_m(\mathbf{x})$, but we don't even need to do so. We could also just randomly select a currently existing champion particle \mathbf{x}^i and use it as the next challenger particle \mathbf{x}_c^j to challenge some (usually different) champion particle \mathbf{x}^j .

Though this new challenger sampling method definitely does speed up convergence, it also changes the limit distribution. In fact, the algorithm stops changing only when all particles are in the same bin. And this bin does not even have to be the right bin! So effectively we have not only sped up the convergence of the algorithm, but also changed its outcome, and that is not what we wanted.

6.3.3. ADDING WEIGHTS TO PARTICLES

The key realization here is that we have changed the distribution from which we are sampling our challengers. We used to sample challenger particles from the flat distribution $q(\mathbf{x}) = \frac{1}{n}$, and now we are sampling them from the distribution $p_m(\mathbf{x})$. We need to compensate for this change.

Identically to Monte Carlo methods (see Section 6.2.2) we will do so by attaching a weight w_c to each new challenger particle \mathbf{x}_c . This weight will equal

$$w_c = \frac{q(\mathbf{x}_c)}{p_m(\mathbf{x}_c)} = \frac{1/n_p}{p_m(\mathbf{x})}. \quad (6.45)$$

So when $p_m(\mathbf{x}_c)$ is large for some input \mathbf{x}_c , we select this input more often than before, but we also give particles from it a lower weight. Similarly, when $p_m(\mathbf{x}_c)$ is small, we select this input less often than before, but we give it particles from it a higher weight. And on average, the weights equal one.

To see how this works out, consider a simple example problem with $n = 2$ bins. Suppose that we have $p_m(1) = 3/4$ of the particles in the bin $x = 1$ and $p_m(2) = 1/4$ of the particles in the bin $x = 2$, and up till now all particles have the same weight. Normally we would select challenger particles randomly from each of the two bins, so $q(1) = q(2) = \frac{1}{2}$. However, because bin $x = 1$ is more likely to be the maximum, we now sample more challenger particles from it. As a result, these challenger particles get a weight $w = \frac{1/2}{3/4} = \frac{2}{3}$,

while challenger particles from the bin $x = 2$ will get a weight $w = \frac{1/2}{1/4} = 2$. Given that we get three times more challenger particles from the bin $x = 1$ than from the bin $x = 2$, this keeps things fair.

Because particles have a weight now, the expression for $p_m(\mathbf{x})$ has changed. We have to take into account the weights. This is done through

$$p_m(\mathbf{x}) = \frac{\sum_{i=1}^{n_p} w^i \delta(\mathbf{x}, \mathbf{x}^i)}{\sum_{i=1}^{n_p} w^i}. \quad (6.46)$$

So to find $p_m(\mathbf{x})$, we look at the sum of the weights of the particles in bin \mathbf{x} and compare it to the total weight of all particles in all bins.

This also means that sampling from $p_m(\mathbf{x})$ is done differently. Previously we could just take a random champion particle and use that as our sample. Now we have to take into account the weights as well: the chance that we pick some particle \mathbf{x}^i is proportional to its weight w^i .

6.3.4. RESAMPLING OF PARTICLES

At this point you may be wondering, ‘If we take into account the weights of particles while selecting a new challenger point, should we then also take into account the weights of particles while selecting a champion to challenge?’

The answer here is ‘No, because this is already being taken into account in another way.’ To see how, consider a small example. We have a bin for $x = 1$ with three particles with weight 1, and a bin for $x = 2$ with one particle with weight 3. Note that in this case, if we ignore weights of particles when finding a champion to challenge, we are three times more likely to challenge a particle from bin $x = 1$ than a particle from bin $x = 2$. However, if we do challenge a particle from bin $x = 1$ and it loses, then the bin loses a weight of 1. On the other hand, if we challenge the particle from bin $x = 2$ and it loses, then the bin loses a weight of 3. So the conclusion is that bin $x = 2$ is three times less likely to be challenged, but when it is challenged and loses, it loses three times more weight, which matches out. As a result, we should still challenge each champion particle just as often, irrespective of its weight.

However, we can also prevent this issue from occurring in the first place by applying resampling. Suppose that we have done a full round of challenges. In other words, we have challenged every single champion particle, and some have gotten replaced while others have survived. In this case, many particles will have different weights. To fix this, we can randomly sample n_p new particles from the distribution $p_{max}(\mathbf{x})$ (or apply any of the other resampling techniques from Section 6.2.5) and give all of these new particles a weight of 1. Because all particles have the same weight now, there is no need to make a distinction between them based on their weight.

6.3.5. ENSURING CORRECT CONVERGENCE

At the start of our algorithm, we randomly (or evenly) divide all our n_p particles over all n bins. Now suppose that the correct bin, which corresponds to the optimal input \mathbf{x}^* , through some unfortunate coincidence does not have any particles in it. With our current set-up, we can only get challengers from bins that already have particles in them. So this guarantees that we will never find the optimal input \mathbf{x}^* .

We can solve this by implementing a variant of defensive importance sampling (see Section 6.2.7). That is, we don't just sample new challenger particles from the maximum distribution $p_m(\mathbf{x})$, and we don't just sample them from the original flat distribution $q(\mathbf{x})$, but we do something in-between. In a part α of the cases we sample from $p_m(\mathbf{x})$, and in the remaining part $(1 - \alpha)$ of the cases we sample from $q(\mathbf{x})$. As a result, we sample from the distribution

$$q'(\mathbf{x}) = \alpha p_m(\mathbf{x}) + (1 - \alpha) q(\mathbf{x}). \quad (6.47)$$

By doing this we ensure that, if we just keep running the algorithm, we are guaranteed to eventually always select a challenger particle \mathbf{x}_c at the optimal input point \mathbf{x}^* .

In practice, to sample from this distribution $q'(\mathbf{x})$, we pick a random champion particle \mathbf{x}^i (taking into account weights). In a part α of the cases we use this as the next challenger \mathbf{x}_c^j , but in a part $(1 - \alpha)$ of the cases we ignore it and use a fully random challenger particle \mathbf{x}_c^j instead.

When we apply this method, we do have to adjust the way we calculate the weight of a challenger particle. We do this according to

$$w_c = \frac{q(\mathbf{x}_c)}{q'(\mathbf{x}_c)} = \frac{q(\mathbf{x}_c)}{\alpha p_m(\mathbf{x}_c) + (1 - \alpha) q(\mathbf{x}_c)}, \quad (6.48)$$

where we use (6.46) to find $p_m(\mathbf{x}_c)$.

There is a problem with the above expression though. If we use (6.46) to calculate $p_m(\mathbf{x}_c)$, we need to consider all particles, only to calculate the weight of a single particle. This is very inefficient and will slow down our algorithm when the number of particles is large.

We have seen this problem before, in Section 6.2.7. We can use the same solution idea here. Suppose that we do not consider the sampling distribution $q'(\mathbf{x})$ from *before* we pick a random champion particle \mathbf{x}^i , but the sampling distribution $q'(\mathbf{x}|\mathbf{x}^i)$ *after* having picked the champion particle \mathbf{x}^i . It equals

$$q'(\mathbf{x}|\mathbf{x}^i) = \alpha \delta(\mathbf{x}, \mathbf{x}^i) + (1 - \alpha) q(\mathbf{x}). \quad (6.49)$$

In other words, in a part α of the cases we set the new challenger particle \mathbf{x}_c^j equal to \mathbf{x}^i , and in a part $(1 - \alpha)$ we pick a fully random challenger. Using this expression, we can set the weight of the resulting challenger particle \mathbf{x}_c equal to

$$w_c = \frac{q(\mathbf{x}_c)}{q'(\mathbf{x}_c|\mathbf{x}^i)} = \frac{q(\mathbf{x}_c)}{\alpha \delta(\mathbf{x}_c, \mathbf{x}^i) + (1 - \alpha) q(\mathbf{x}_c)}. \quad (6.50)$$

By using this expression, we get on average the same result as when we would use (6.48), but we save ourselves a lot of computations.

6.3.6. EXPANDING THE ALGORITHM TO CONTINUOUS FUNCTIONS

The algorithm so far has been set up for functions $f(\mathbf{x})$ with a finite number of possible input points \mathbf{x} . Continuous functions have an infinite number of possible input points. This changes some things, although less than may initially be expected.

The main difference is how we calculate $p_m(\mathbf{x})$. Instead of an actual probability, it is now a probability density function, officially written as $f_{\underline{\mathbf{x}}^*}(\mathbf{x})$, although we still write it as $p_m(\mathbf{x})$ to keep the same notation. Through weighted kernel density estimation, we can approximate it as

$$p_m(\mathbf{x}) = \frac{1}{n_p} \sum_{i=1}^{n_p} w^i k_x(\mathbf{x}, \mathbf{x}^i), \quad (6.51)$$

where $k_x(\mathbf{x}, \mathbf{x}^i)$ is a kernel function we have to choose ourselves. Personally, I prefer a Gaussian kernel with a small length scale.

Sampling from $p_m(\mathbf{x})$ is also done slightly differently. Previously, if we wanted to sample from $p_m(\mathbf{x})$, we just needed to pick a random champion particle \mathbf{x}^i , taking into account their weights. Now, we need to do an extra step. First we still need to pick a random champion particle \mathbf{x}^i , but afterwards we need to generate a sample from the kernel $k_x(\mathbf{x}, \mathbf{x}^i)$, which then is our sample from $p_m(\mathbf{x})$. We need to use this method as well when sampling challenger particles from $p_m(\mathbf{x})$.

Because the way in which we sample challenger particles has changed, we also need to calculate the weight of these particles in a new way. To be precise, equation (6.49) for $q'(\mathbf{x}|\mathbf{x}^i)$ turns into

$$q'(\mathbf{x}|\mathbf{x}^i) = \alpha k_x(\mathbf{x}, \mathbf{x}^i) + (1 - \alpha) q(\mathbf{x}), \quad (6.52)$$

which turns the expression for the weight w_c into

$$w_c = \frac{q(\mathbf{x}_c)}{q'(\mathbf{x}_c|\mathbf{x}^i)} = \frac{q(\mathbf{x}_c)}{\alpha k_x(\mathbf{x}_c, \mathbf{x}^i) + (1 - \alpha) q(\mathbf{x}_c)}. \quad (6.53)$$

When we apply this, the algorithm can safely be applied to continuous functions. This is special, as most algorithms cannot be extended so easily from finitely many possible input points to infinitely many. Through it, we can approximate the maximum distribution of a Gaussian process in a relatively efficient way. The algorithm itself is summarized in pseudo-code in Algorithm 2, while the result of applying the algorithm is shown in Figure 6.9.

6.3.7. THE DISTRIBUTION OF THE MAXIMUM VALUE

So far we have focused on the distribution of the optimal input $\underline{\mathbf{x}}^*$. Another interesting random variable is the corresponding optimal function value \underline{f}^* . Can we find/approximate the distribution of this parameter too?

Naturally, we can. To do so, we should focus on the actual ‘challenging’. Just like we described in Section 6.1.4, this is done by setting up a joint distribution

$$\begin{bmatrix} f(\mathbf{x}^i) \\ f(\mathbf{x}_c^i) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} f^i \\ f_c^i \end{bmatrix} \middle| \begin{bmatrix} \mu(\mathbf{x}^i) \\ \mu(\mathbf{x}_c^i) \end{bmatrix}, \begin{bmatrix} \Sigma(\mathbf{x}^i, \mathbf{x}^i) & \Sigma(\mathbf{x}^i, \mathbf{x}_c^i) \\ \Sigma(\mathbf{x}_c^i, \mathbf{x}^i) & \Sigma(\mathbf{x}_c^i, \mathbf{x}_c^i) \end{bmatrix} \right) \quad (6.54)$$

for the champion particle \mathbf{x}^i and its challenger \mathbf{x}_c^i . We then take a sample $[\hat{f}^i \quad \hat{f}_c^i]^T$ from this distribution. If $\hat{f}_c^i > \hat{f}^i$, we replace the champion particle by its challenger.

Next, we will do something extra. Whenever a challenger particle wins, we remember the value of \hat{f}_c^i through which it won. As a result, every particle \mathbf{x}^i will get a corresponding *victory function value* \hat{f}^i . (For an initial champion particle \mathbf{x}^i , we can sample the initial victory function value \hat{f}^i from $\mathcal{N}(\mu(\mathbf{x}^i), \Sigma(\mathbf{x}^i, \mathbf{x}^i))$.)

Data: A Gaussian process, user-defined parameters n_p , α and a kernel $k'(\mathbf{x}, \mathbf{x}')$.
Result: An approximate distribution $p_m(\mathbf{x})$ of the optimal input \mathbf{x}^* through (6.51).

Initialization:

```

for  $i \leftarrow 1$  to  $n_p$  do
  | Sample  $\mathbf{x}^i$  from the flat distribution  $q(\mathbf{x})$ . Assign  $w^i = 1$ .
end
end

```

Iteration:

```

repeat
  | Apply systematic resampling to all particles.
  | for  $i \leftarrow 1$  to  $n_p$  do
    |   Select a random particle  $\mathbf{x}^j$ .
    |   if we select a challenger based on  $\mathbf{x}^j$  (probability  $\alpha$ ) then
    |     | Sample a challenger particle  $\mathbf{x}_c^i$  from the kernel  $k_x(\mathbf{x}, \mathbf{x}^j)$ .
    |   else
    |     | Sample a challenger particle  $\mathbf{x}_c^i$  from the flat distribution  $q(\mathbf{x})$ .
    |   end
    |   Sample a vector  $[\hat{f}^i \quad \hat{f}_c^i]^T$  based on (6.54).
    |   if  $\hat{f}_c^i > \hat{f}^i$  then
    |     | Replace particle  $\mathbf{x}^i$  by its challenger  $\mathbf{x}_c^i$ .
    |     | Set the new weight  $w^i$  according to (6.53).
    |   end
  | end
until a sufficient number of challenge rounds has passed;
end

```

Algorithm 2: The Monte Carlo maximum distribution algorithm for continuous functions. All the discussed improvements, like self-normalized importance sampling, mixture importance sampling and systematic resampling, have been incorporated.

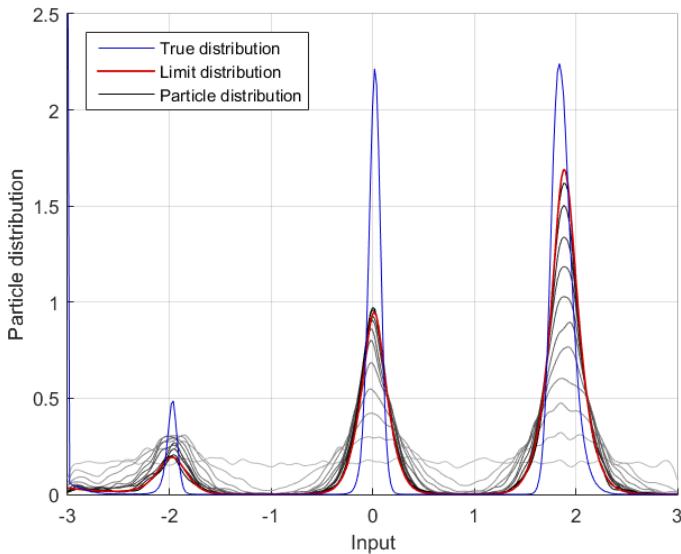


Figure 6.9: The development of the distribution of particles over $n_r = 10$ rounds using the improved version of the MCMD algorithm. Shown are the true maximum distribution from Figure 6.2, the limit distribution of the particles derived in Section 6.1.5 and the distribution of the particles over each of the rounds. The darker the line, the later the round is. Note that convergence for the improved algorithm is much faster than for the basic algorithm (Figure 6.5).

6

These victory function values now describe the distribution of \underline{f}^* . To be precise, we can approximate the distribution of \underline{f}^* using kernel density estimation as

$$\underline{f}^* \sim f_{\underline{f}^*}(f) = \frac{\sum_{i=1}^{n_p} w^i k_f(f, \hat{f}^i)}{\sum_{i=1}^{n_p} w^i}. \quad (6.55)$$

In other words, we just make a histogram of all the victory function values, based on the particle weights and a self-defined kernel function $k_f(f, \hat{f}^i)$. An example distribution resulting from this is shown in Figure 6.10.

6.4. GAUSSIAN PROCESS OPTIMIZATION

We will now look at the actual Gaussian process optimization problem, where we need to find the optimum of a function $f(\mathbf{x})$ approximated by a GP. We start by considering the exact problem set-up (Section 6.4.1). Then we look at some basic solution methods, which generally involve acquisition functions (Section 6.4.2). We continue with a short intermezzo on the concept of entropy (Section 6.4.3) before we apply it to set up a new type of acquisition function (Section 6.4.4). We then look into ways of combining acquisition functions (Section 6.4.5). Finally we consider a very different method, where we sample from the maximum distribution to select new points to try out (Section 6.4.6).

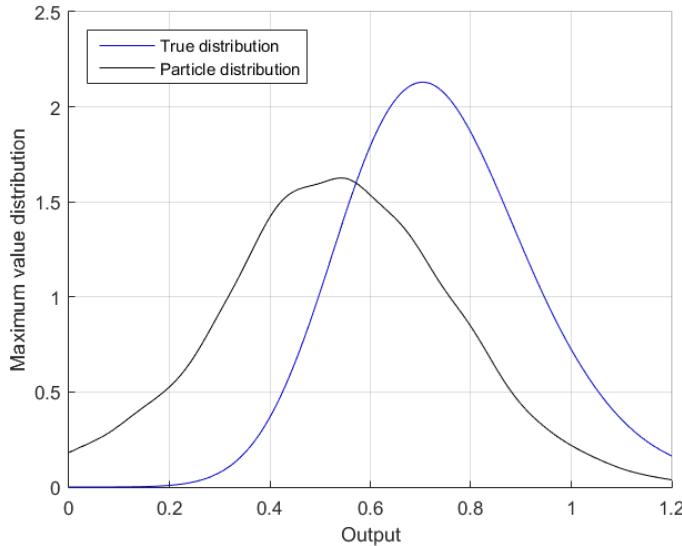


Figure 6.10: The distribution of the maximum value, derived using either brute force methods (the true maximum distribution) or the MCMD algorithm. Note that the MCMD algorithm gives a slightly lower estimate of the maximum than what we can expect in reality. This slight conservatism is inherent to the algorithm and can be expected at any application. The difference becomes smaller when the variance of the Gaussian process decreases.

6

6.4.1. THE GAUSSIAN PROCESS OPTIMIZATION PROBLEM SET-UP

Consider a function $f(\mathbf{x})$. Here, the input \mathbf{x} may represent ‘choices’ we have to make, and the function $f(\mathbf{x})$ specifies how good these choices are. For instance, it may be a cost function that we need to minimize, or a reward function we need to maximize. In this chapter we consider the maximization problem, but maximizing a function $f(\mathbf{x})$ is of course equivalent to minimizing the function $-f(\mathbf{x})$.

We do not know what the function $f(\mathbf{x})$ is though. We can only try certain inputs \mathbf{x} to obtain a (usually noisy) indication of the result $f(\mathbf{x})$. However, evaluating $f(\mathbf{x})$ is expensive. We can only try out a limited number n input points $\mathbf{x}_1, \dots, \mathbf{x}_n$ called *try-out points*. How should we now choose these try-out points to most efficiently find the optimum of $f(\mathbf{x})$?

Part of the solution lies in approximating the function $f(\mathbf{x})$ at every step of the way. We do so using Gaussian processes. As such, this optimization problem is generally known as *Gaussian process optimization* (GPO) or sometimes as *Bayesian optimization*. How can we use a GP approximation to find the optimum of a function?

There are actually two different versions of this problem. In the first set-up, which is the most common, we have to give a recommendation $\hat{\mathbf{x}}^*$ at the end of the algorithm run. This recommendation $\hat{\mathbf{x}}^*$ equals the input point which we believe optimizes the function. Subsequently, we compare the value $f(\hat{\mathbf{x}}^*)$ to the true optimum f^* . The difference is known as the *error*, or sometimes as the *instantaneous regret*. So,

$$\text{error} = f^* - f(\hat{\mathbf{x}}^*). \quad (6.56)$$

Because of this, this set-up is known as the *error minimization set-up*, although it is also known as *probabilistic global optimization*.

The second set-up differs from the first on one important point. In the first set-up we were free to try out any input point \mathbf{x} . In the second set-up different input points \mathbf{x} result in different costs. Specifically, we want to maximize the sum

$$\sum_{i=1}^n f(\mathbf{x}_i). \quad (6.57)$$

In the perfect case, when we choose the optimal input \mathbf{x}^* every single time, we obtain a sum equal to $n f^*$. If the actual sum we obtain is less than this ‘perfect score’, then we call the difference the *regret*. It follows that the regret is defined as the sum of the errors

$$\text{regret} = \sum_{i=1}^n (f^* - f(\mathbf{x}_i)). \quad (6.58)$$

Since we want to minimize this regret, this set-up is known as the *regret minimization set-up*, although it is also known as the *continuous armed-bandit problem*.

These two different set-ups may seem similar, but they require very different strategies. In the error minimization set-up, we want to explore as much as possible, trying out a variety of possible input points. In the regret minimization set-up, we have to trade off *exploration* versus *exploitation*. Initially, we should do some exploring, looking for input points \mathbf{x} resulting in good function values $f(\mathbf{x})$. However, after we have done enough exploring, we should stick with the best input point we found, exploiting it as much as possible. How to transition from exploration to exploitation is known to be a very difficult problem.

6

6.4.2. BASIC GPO METHODS: ACQUISITION FUNCTIONS

We are approximating the function $f(\mathbf{x})$ using a Gaussian process. Suppose that, based on all our current data, we have come up with a mean function $\mu(\mathbf{x})$ and a covariance function $\Sigma(\mathbf{x}, \mathbf{x}')$. Corresponding to this is the *standard deviation function*

$$\sigma(\mathbf{x}) \equiv \sqrt{\Sigma(\mathbf{x}, \mathbf{x})}. \quad (6.59)$$

The main question in GPO is ‘How do we select the next try-out point \mathbf{x}_k ?’

Almost all existing methods try to optimize some criterion. This criterion is known as the *acquisition function* (AF): it specifies which input point to try or ‘acquire’ next. So given an acquisition function $\text{AF}(\mathbf{x})$, we acquire

$$\mathbf{x}_k = \arg \min_{\mathbf{x}} \text{AF}(\mathbf{x}). \quad (6.60)$$

However, which acquisition function should we take? There are numerous possibilities.

THE EXPECTED VALUE ACQUISITION FUNCTION

A simple example of an acquisition function is the *expected value acquisition function* (EV AF). This is defined as

$$\text{EV}(\mathbf{x}) \equiv \mu(\mathbf{x}). \quad (6.61)$$

Picking the input point \mathbf{x} maximizing this acquisition function comes down to picking what we currently believe is the optimal input $\hat{\mathbf{x}}^*$. This is also shown by the example in Figure 6.11. As such, the EV AF is a strategy purely focused on exploitation and not on exploration, making it a pretty bad strategy.

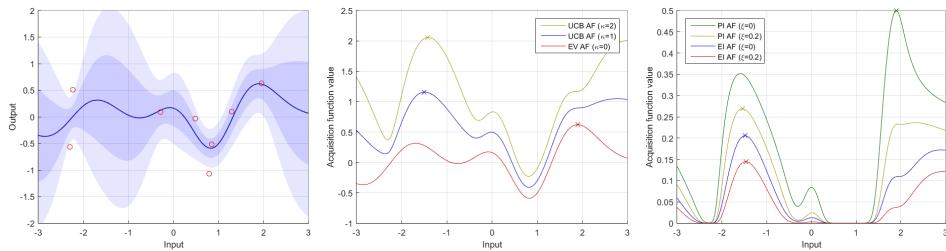


Figure 6.11: Examples of acquisition functions applied to a Gaussian process. As Gaussian process (left) we have used the same set-up as Figure 6.1, except that we only used $n_m = 8$ measurements to create some regions with strong uncertainties. For this Gaussian process, both the UCB and EV acquisition functions are plotted (middle) and the PI and EI acquisition functions (right), for a variety of parameters. The optimums of the AFs are indicated by a cross. Note that the PI and EV acquisition function (with $\kappa = \xi = 0$) both suggest to try a point near $x = 2$ (exploitation) while all other acquisition functions suggest a point near $x = -1.5$ (exploration).

THE PROBABILITY OF IMPROVEMENT ACQUISITION FUNCTION

6

A more suitable acquisition function is the *probability of improvement acquisition function* (PI AF). To implement this, we ask ourselves two questions. First of all, ‘If we had to give a recommendation about the position of the optimum, what would we pick?’ The obvious choice here would be to choose this recommendation $\hat{\mathbf{x}}^*$ as the optimum of the mean function $\mu(\mathbf{x})$. We write the corresponding optimal value as \hat{f}^* .

Naturally, we want to find an input point \mathbf{x} which does better than this current optimum. So the second question is, ‘Which input \mathbf{x} is most likely to give a value $f(\mathbf{x})$ satisfying $f(\mathbf{x}) > \hat{f}^*$?’ In other words, we want to maximize the probability $p(f(\mathbf{x}) > \hat{f}^*)$ that we find an improvement⁵. The corresponding acquisition function equals

$$\text{PI}(\mathbf{x}) = \int_{\hat{f}^*}^{\infty} \mathcal{N}(f | \mu(\mathbf{x}), \sigma(\mathbf{x})^2) df = \Phi\left(\frac{\mu(\mathbf{x}) - \hat{f}^*}{\sigma(\mathbf{x})}\right), \quad (6.62)$$

where $\Phi(z)$ is still the cumulative density function of the standard Gaussian probability distribution, as defined in Section B.4.2.

The problem with the PI AF is that it still does a lot of exploitation: the selected try-out point is often close or equal to the current optimum $\hat{\mathbf{x}}^*$. This is also confirmed by the example in Figure 6.11. To make sure that we only try points that offer a significant improvement, we can add an *exploration parameter* ξ . We now adjust the PI AF to the

⁵In some literature, the possible function value $f(\mathbf{x})$ is not compared to the *current belief* about the maximum \hat{f}^* , but instead to the *best output* $\max_i \hat{f}_{m_i}$ obtained so far. This is computationally easier – we do not have to optimize $\mu(\mathbf{x})$ – but since this best output is affected by noise, this also may result in unexpected results. A single instance of very positive noise can disrupt the entire algorithm.

probability that we get an improvement of at least size ξ . So,

$$\text{PI}(\mathbf{x}) = p(f(\mathbf{x}) > \hat{f}^* + \xi) = \Phi\left(\frac{\mu(\mathbf{x}) - \hat{f}^* - \xi}{\sigma(\mathbf{x})}\right). \quad (6.63)$$

Through this parameter ξ , we can vary between exploration and exploitation. A high value of ξ will give us more exploring try-out points, while a low value will result in more exploitation.

THE EXPECTED IMPROVEMENT ACQUISITION FUNCTION

It might be wiser to not just look at the probability of improvement, but also consider the amount of improvement which we can expect to get. This is done by the *expected improvement acquisition function* (EI AF). It is given by

$$\begin{aligned} \text{EI}(\mathbf{x}) &= \int_{\hat{f}^*}^{\infty} (f - \hat{f}^*) \mathcal{N}(f|\mu(\mathbf{x}), \sigma(\mathbf{x})^2) df \\ &= (\mu(\mathbf{x}) - \hat{f}^*) \Phi\left(\frac{\mu(\mathbf{x}) - \hat{f}^*}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x}) \phi\left(\frac{\mu(\mathbf{x}) - \hat{f}^*}{\sigma(\mathbf{x})}\right), \end{aligned} \quad (6.64)$$

with $\phi(z) = \mathcal{N}(z|0, 1)$ the probability density function of the standard Gaussian distribution. And just like previously, it is possible to add the exploration parameter ξ to adjust the above to

$$\text{EI}(\mathbf{x}) = (\mu(\mathbf{x}) - \hat{f}^*) \Phi\left(\frac{\mu(\mathbf{x}) - \hat{f}^* - \xi}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x}) \phi\left(\frac{\mu(\mathbf{x}) - \hat{f}^* - \xi}{\sigma(\mathbf{x})}\right). \quad (6.65)$$

Again, the higher ξ is, the more this acquisition will focus on exploration. For an example problem the acquisition function is shown in Figure 6.11.

THE UPPER CONFIDENCE BOUND ACQUISITION FUNCTION

In general, both the PI and the EI acquisition functions are quite focused on exploration, and as such are most suitable for the error minimization set-up. An acquisition which is more suitable for the regret minimization set-up is the *upper confidence bound acquisition function* (UCB AF). It is defined as

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa \sigma(\mathbf{x}). \quad (6.66)$$

Often $\kappa = 2$ is used. In this case, this acquisition function simply comes down to picking the input point \mathbf{x} where the grey area from the GP plot reaches the highest, as is also shown in Figure 6.11. In general, high values of κ result in more exploration, while a value of $\kappa = 0$ reduces the UCB AF to the EVAF, causing pure exploitation.

Out of all the acquisition functions, the UCB AF has been analyzed the most thoroughly. Under certain conditions, bounds on the regret that will be obtained are known, as well as the values of κ optimizing these bounds. For further reading, see the work by Srinivas et al. (2012).

6.4.3. INTERMEZZO: THE ENTROPY OF DISTRIBUTIONS

After the basic methods we just discussed, more advanced methods got developed. An important one is entropy search. To understand how it works, you need to know what entropy means in the field of information theory. So we have a short intermezzo on this. If you are familiar with the concept of entropy, feel free to jump to Section 6.4.4 on entropy search.

Suppose that we have a discrete random variable \underline{x} . This variable can take n different values $\underline{x}_1, \dots, \underline{x}_n$, each with corresponding probability $p(\underline{x}_1), \dots, p(\underline{x}_n)$. (For example, imagine that \underline{x} has a 50% chance to equal 1 and a 25% chance to equal either 2 or 3.) For this discrete random variable, the *entropy* $H(\underline{x})$ is defined as

$$H(\underline{x}) \equiv \mathbb{E}[-\log(p(\underline{x}))] = -\sum_{i=1}^n p(\underline{x}_i) \log(p(\underline{x}_i)). \quad (6.67)$$

You can see the entropy as the amount of chaos or uncertainty in a distribution. Suppose that we do not have a clue at all what \underline{x} is. That is, all possible values x_1, \dots, x_n are equally likely, and hence have probability $p(\underline{x}_i) = \frac{1}{n}$. In that case the entropy turns out to be at a maximum and equals

$$\text{maximum entropy} = -\sum_{i=1}^n \frac{1}{n} \log\left(\frac{1}{n}\right) = \log(n). \quad (6.68)$$

If we gain more information, some probabilities $p(\underline{x}_i)$ will rise while others $p(\underline{x}_j)$ will drop. In the extreme case that we know that \underline{x} exactly equals \underline{x}_i , then $p(\underline{x}_i) \rightarrow 1$, which causes $\log(p(\underline{x}_i)) \rightarrow 0$ and hence $p(\underline{x}_i) \log(p(\underline{x}_i)) \rightarrow 0$. Similarly, when $p(\underline{x}_j) \rightarrow 0$, it follows that $p(\underline{x}_j) \log(p(\underline{x}_j)) \rightarrow 0$. As a result, if we are fully certain what the value of \underline{x} is, all terms in the sum become zero and hence the entropy equals zero as well.

We can extend this to continuous distributions. However, for continuous distributions, when \underline{x} can take infinitely many values, the probability $p(\underline{x} = \underline{x})$ generally equals zero for (nearly) all \underline{x} , which means the entropy is also always zero. To fix this problem, we replace $p(\underline{x})$ in the definition by the probability density function $f_{\underline{x}}(\underline{x}) = \frac{p(\underline{x})}{d\underline{x}}$. The result is known as the *continuous entropy* or the *differential entropy* and equals

$$H(\underline{x}) \equiv \mathbb{E}[-\log(f_{\underline{x}}(\underline{x}))] = -\int_X f_{\underline{x}}(\underline{x}) \log(f_{\underline{x}}(\underline{x})) d\underline{x}. \quad (6.69)$$

If \underline{x} has a Gaussian distribution $\underline{x} \sim \mathcal{N}(\underline{x} | \boldsymbol{\mu}, \Sigma)$, then we can rewrite the entropy to

$$\begin{aligned} H(\underline{x}) &= \mathbb{E}\left[-\log\left(\frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2} (\underline{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\underline{x} - \boldsymbol{\mu})\right)\right)\right] \\ &= \mathbb{E}\left[\frac{1}{2} \log(|2\pi\Sigma|) + \frac{1}{2} (\underline{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\underline{x} - \boldsymbol{\mu})\right]. \end{aligned} \quad (6.70)$$

Note that the first term in the above expectation is constant and so can be pulled out. The second term can be solved by applying the trace operator, reordering terms and noticing

that Σ per definition equals $\mathbb{E} \left[(\underline{\mathbf{x}} - \boldsymbol{\mu}) (\underline{\mathbf{x}} - \boldsymbol{\mu})^T \right]$. As a result, we have

$$H(\underline{\mathbf{x}}) = \frac{1}{2} \log \left((2\pi)^{d_x} |\Sigma| \right) + \frac{1}{2} \text{tr} \left(\mathbb{E} \left[(\underline{\mathbf{x}} - \boldsymbol{\mu}) (\underline{\mathbf{x}} - \boldsymbol{\mu})^T \right] \Sigma^{-1} \right) \quad (6.71)$$

$$= \frac{1}{2} \left(\log(|\Sigma|) + d_x \log(2\pi) + d_x \right), \quad (6.72)$$

where d_x is the size of the vector $\underline{\mathbf{x}}$. So when the determinant of the covariance is large, and hence the uncertainty we have about $\underline{\mathbf{x}}$ is large, then the entropy will also be large.

What does the above become in the limit cases? If we do not know anything about $\underline{\mathbf{x}}$, then the probability density function $f_{\underline{\mathbf{x}}}(\underline{\mathbf{x}})$ is constant, which is equivalent to an infinite variance. I call this the *null distribution*. (For background on this, see Sections B.1.4 and B.4.5.) As a result, the entropy becomes $H(\underline{\mathbf{x}}) = \infty$. However, when we are fully certain of the value of $\underline{\mathbf{x}}$, then the PDF turns into a delta function (zero variance) and we obtain an entropy of $H(\underline{\mathbf{x}}) = -\infty$.

This is actually undesirable, since previously the entropy was always positive, unless we were absolutely certain about the value of $\underline{\mathbf{x}}$, in which case the entropy was zero. This problem is solved by considering the *relative entropy* with respect to another PDF $f_{\tilde{\underline{\mathbf{x}}}}(\underline{\mathbf{x}})$. This is also known as the *Kullback-Leibler divergence* and is defined as

$$D(\underline{\mathbf{x}}, \tilde{\underline{\mathbf{x}}}) \equiv \mathbb{E} \left[\log \left(\frac{f_{\underline{\mathbf{x}}}(\underline{\mathbf{x}})}{f_{\tilde{\underline{\mathbf{x}}}}(\underline{\mathbf{x}})} \right) \right] = \int_X f_{\underline{\mathbf{x}}}(\underline{\mathbf{x}}) \log \left(\frac{f_{\underline{\mathbf{x}}}(\underline{\mathbf{x}})}{f_{\tilde{\underline{\mathbf{x}}}}(\underline{\mathbf{x}})} \right) d\underline{\mathbf{x}}. \quad (6.73)$$

Note that we do *not* have $D(\underline{\mathbf{x}}, \tilde{\underline{\mathbf{x}}})$ here. The Kullback-Leibler divergence is not a symmetric operator.

When $\underline{\mathbf{x}} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and $\tilde{\underline{\mathbf{x}}} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$, then the above can be rewritten as

$$D(\underline{\mathbf{x}}, \tilde{\underline{\mathbf{x}}}) = \frac{1}{2} \left(\log \left(\frac{|\tilde{\Sigma}|}{|\Sigma|} \right) + \text{tr} \left(\tilde{\Sigma}^{-1} \Sigma - I \right) + (\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu})^T \tilde{\Sigma}^{-1} (\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}) \right). \quad (6.74)$$

This relative entropy has some useful properties. It cannot be negative, and it is only zero if $\underline{\mathbf{x}}$ and $\tilde{\underline{\mathbf{x}}}$ have exactly the same distribution. The more different these two distributions are though, the bigger the relative entropy becomes.

So what comparison distribution $\tilde{\underline{\mathbf{x}}}$ do we pick? That depends. An idea is to let $\tilde{\underline{\mathbf{x}}}$ have the null distribution, making $f_{\tilde{\underline{\mathbf{x}}}}(\underline{\mathbf{x}})$ a constant. If we are integrating over an infinitely large space of possible input values X , then this will not be possible though. The reason is that the resulting constant $f_{\tilde{\underline{\mathbf{x}}}}(\underline{\mathbf{x}})$ will be zero, and hence the relative entropy will always be infinite (unless $\underline{\mathbf{x}}$ also has the null distribution). If, however, the region X is bounded, then $f_{\tilde{\underline{\mathbf{x}}}}(\underline{\mathbf{x}})$ will be a non-zero constant and this trick will work.

In this case, the more we know about $\underline{\mathbf{x}}$, the further it will lie away from $\tilde{\underline{\mathbf{x}}}$, and hence the larger the relative entropy becomes. So in this case a high *relative* entropy means we know *more* about $\underline{\mathbf{x}}$, while a high entropy means we know *less* about $\underline{\mathbf{x}}$.

6.4.4. ENTROPY SEARCH

Using the ideas of entropy which we just discovered, it is now possible to set up a completely new acquisition function.

Consider the optimal input point \underline{x}^* . This is a random variable, and as such it has a probability density function $f_{\underline{x}^*}(\underline{x}^*)$. We want to obtain as much information as possible about where the optimal input point will be. As a result, we want to minimize the entropy $H(\underline{x}^*)$, or alternatively we want to maximize the relative entropy $D(\underline{x}^*, \tilde{\underline{x}}^*)$, where $\tilde{\underline{x}}^*$ is a random variable with the null distribution. This mindset gives rise to the idea of entropy search.

In *entropy search* we use an acquisition function describing the change in entropy. We want to find the input point \underline{x} which maximizes the change in relative entropy. As such, we could try to use the *entropy search acquisition function*

$$\text{ES}(\underline{x}) = \Delta D(\underline{x}^*, \tilde{\underline{x}}^*). \quad (6.75)$$

The question remains how to calculate this change in relative entropy. Calculating the current distribution of \underline{x}^* and subsequently the current relative entropy is already a challenging problem. However, picking an input point \underline{x} , and then calculating the new relative entropy is even harder. This is because the new relative entropy actually depends on the measurement \hat{f}_m which we make of $f(\underline{x})$.

So to find the expected change in entropy, we need to integrate over all possible values of \hat{f}_m . As a result, we should have actually written (6.75) as

$$\text{ES}(\underline{x}) = \mathbb{E} [\Delta D(\underline{x}^*, \tilde{\underline{x}}^*)], \quad (6.76)$$

where the expectation is taken with respect to the possible measurement \hat{f}_m . Solving this expectation is a very challenging problem which is outside the scope of this text, so for more information you can read the work of [Hennig and Schuler \(2012\)](#). In addition, further improvements were made by [Hernández-Lobato et al. \(2014a\)](#), who also added derivative data into the algorithm and integrated over different possible hyperparameters. However, both groups of people struggled in finding a method to efficiently optimize the change in relative entropy without making too many approximations.

6.4.5. PORTFOLIO METHODS

So far we have seen many different kinds of acquisition functions. However, it is not the case that one acquisition function is always better than another. Each works well for specific kinds of functions. Additionally, some work well at the start of the learning process, when we need exploration, while others work well later on. This gives rise to an idea, resulting in the so-called *portfolio methods*.

The idea is that we take a whole portfolio $\text{AF}_1(\underline{x}), \dots, \text{AF}_{n_a}(\underline{x})$ of n_a different acquisition functions. Just think of all the acquisition functions we have seen so far, and we can even include an acquisition function multiple times if we change its parameters a bit. When we need to decide which input \underline{x}_k to try at time k , each acquisition function $\text{AF}_i(\underline{x})$ in our portfolio first makes a *recommendation* \underline{x}_k^i . We then set up an encompassing algorithm that decides which of these recommendations is the most suitable in the current situation. This is the fundamental idea behind portfolio methods.

The main question is, ‘How do we decide which recommendation is suitable?’ A first idea is to take into account the previous performance of each acquisition function. That is, we keep track of all previous recommendations $\underline{x}_1^i, \underline{x}_2^i, \dots$ of every acquisition function

$\text{AF}_i(\mathbf{x})$ and look at the means $\mu(\mathbf{x}_1^i), \mu(\mathbf{x}_2^i), \dots$ for these recommendations given all the data that we have now. If these means are all very high for some acquisition function $\text{AF}_i(\mathbf{x})$, then this acquisition function apparently gives decent recommendations. Right?

Actually, it may also be an acquisition function that is fully focused on exploiting and not on exploring. As a result, this method works reasonably well, but it is not the best method developed so far. If you want to read more about it, see the work by [Hoffman et al. \(2011\)](#).

A very different idea is to ignore past recommendations and instead only look at the current recommendations $\mathbf{x}_k^1, \dots, \mathbf{x}_k^{n_a}$ from the various acquisition functions. We have already seen that the input point \mathbf{x} maximizing the expected change $\mathbb{E}[\Delta D]$ of the relative entropy is probably a good choice. So we could hence simply check the n_a different recommendations we have been given and see which one results in the largest change in entropy. This means that we do not have to optimize $\text{ES}(\mathbf{x})$ over a large input space, but we only need to check which of the recommendations $\mathbf{x}_k^1, \dots, \mathbf{x}_k^{n_a}$ results in the largest value of $\text{ES}(\mathbf{x})$. For more background on this idea, you can look into the work by [Shahriari et al. \(2014\)](#).

All the methods we have discussed so far here (with the exception of the UCB AF) are methods designed for the error minimization set-up. If you would apply these methods to the regret minimization set-up, they would not perform too well, because they keep on exploring. However, the number of methods applicable to the regret minimization set-up is still very scarce. So let's consider a new method which can tackle this problem.

6

6.4.6. THOMPSON SAMPLING

The last method we discuss is called *Thompson sampling*, or sometimes also *probability matching* or *posterior sampling*. This method is special in that it does not use an acquisition function. Instead, it randomly selects its try-out points, where the probability that a certain point \mathbf{x} is selected as the next try-out point \mathbf{x}_k equals (matches) the probability that \mathbf{x} is the optimal input point \mathbf{x}^* , given all the data that we have so far.

Effectively, with this method we sample our next try-out point \mathbf{x}_k from the maximum distribution $f_{\underline{\mathbf{x}}^*}(\mathbf{x})$. However, the problem is that we generally do not know this maximum distribution. Luckily, several ways have been invented to work around this.

Suppose that we only have a finite number of possible input points. We can then take a trial input set X_* that contains all these input points, and subsequently take a sample \hat{f}_* of the posterior distribution of \underline{f}_* . Such a sample can be visualized as a function sample, like we did in Figure 6.1. For this sample, we then find the input point \mathbf{x}_* for which this sample is at a maximum and use it as the next try-out point \mathbf{x}_k . By doing this, we indeed select our next try-out point according to the maximum distribution.

This trick is great, but it does not work when there are infinitely many possible input points. We could try to optimize a sample function then, through a recursive process. So pick a first input point \mathbf{x}_1 , sample the output \hat{f}_1 , then pick a second input point \mathbf{x}_2 , sample the output \hat{f}_2 given the value of \hat{f}_1 , and so on. By picking the input points in a smart way, trying to find the optimum of the sample, we can optimize the sample. However, sampling \hat{f}_k takes $\mathcal{O}k^2$ time, resulting in a runtime of $\mathcal{O}n_{opt}^3$ for the full optimization algorithm, with n_{opt} the number of points we need to check to optimize the sample. Of course this runtime is far too high.

There are methods to work around this. It is for instance possible to approximate function samples using a finite number of basis functions. (See the supplement written by Hernández-Lobato et al. (2014b) to learn how to do this.) Since these basis functions can be approximated at any input point x_* this solves the problem that we could not generate samples on infinitely many points. However, it does not solve the problem that we need to optimize a nonlinear function. As a result, this method is effective but not very efficient.

The problem is solved by the MCMD algorithm we developed in Section 6.1.4 and further improved in Section 6.3. It allows us to efficiently sample input points from the maximum distribution. We should keep in mind here that the MCMD algorithm provides only an approximation of the maximum distribution. But with the approximation being reasonably accurate, this does mean that it is now possible to apply Thompson sampling to problems with a continuous input space.

In short, Thompson sampling now works as follows. We use the MCMD algorithm to approximate the maximum distribution. When we need to pick the next try-out point x_k , we just take a random sample from the maximum distribution and use it.

6.5. EXPERIMENTS

It is time to do some experiments with our newly developed methods. We start off with simple one-dimensional (Section 6.5.1) and two-dimensional (Section 6.5.2) example problems. We then see if we can apply the methods to tune the controller of a wind turbine. For this, we first check out the wind turbine simulation that we will use (Section 6.5.3). We then study some necessary concepts, like the Coleman transformation (Section 6.5.4) and the damage equivalent load (Section 6.5.5), before we evaluate the experiment results (Section 6.5.6). Afterwards, we apply the same methods to an actual wind turbine in a wind tunnel (Section 6.5.7). Finally, we look at some lessons that we learned from the experiments (Section 6.5.8).

6

6.5.1. OPTIMIZING A ONE-DIMENSIONAL FUNCTION

We start our experiment section off with a familiar one-dimensional experiment. We will apply our Gaussian process optimization algorithms towards finding the optimum of the unknown function

$$f(x) = \cos(3x) - \frac{1}{9}x^2 + \frac{1}{6}x, \quad (6.77)$$

which we also drew measurements from in Figure 6.1. Once more we use $\sigma_n = 0.3$. As parameters for the various algorithms we use $\kappa = 2$ (for the UCB AF) and $\xi = 0.1$ (for the PI and EI AFs), which have shown to work better than significantly larger or smaller parameters. For Thompson sampling through the MCMD algorithm, we use a value of α that decreases from $\frac{2}{3}$ to $\frac{1}{6}$ as more measurements have been made. This has shown to have a slightly beneficial effect on the performance, compared to using a constant α .

Whenever we perform a new measurement, we update the Gaussian process. To do so, we apply the online FITC method of Section 4.2.3. We also set up the set of inducing input points in an online way: we add an inducing input point (according to the ideas of Section 4.3.3) whenever the new measurement is not within a distance d_u (decreasing from 0.3 to 0.02 during the execution of the algorithm) from an already existing inducing

input point. This does result in a small data loss; early measurements do not provide their data directly to inducing input points that are added later on. However, this loss is relatively small, because our strategy implies that there is always an inducing input point close to any chosen measurement point.

An example of the various strategies at work is shown in Figure 6.12. Here we see that all algorithms are quite capable of finding the optimum. However, some require more exploration, and some continue to check other local optima just to make sure that they do not happen to give better results.

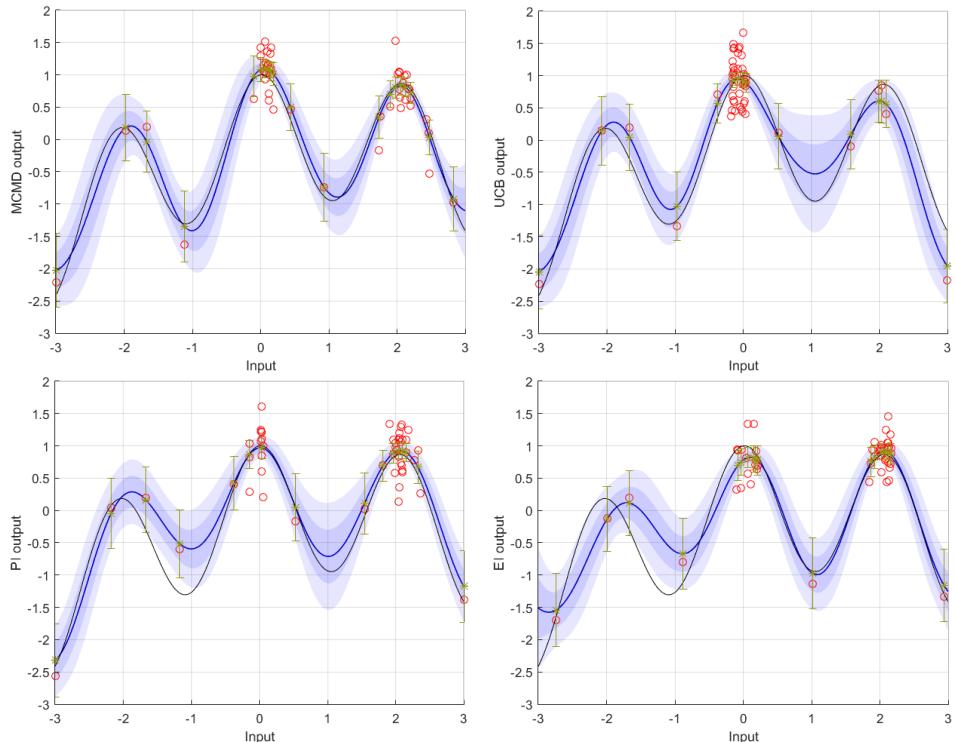


Figure 6.12: Execution runs of various Gaussian process optimization strategies for the problem described in Figure 6.1 for the first $n = 50$ measurements. The real function $f(x) = \cos(3x) - \frac{1}{9}x^2 + \frac{1}{6}x$ is shown, as well as the posterior Gaussian process with its inducing input points. We have used Thompson sampling through the MCMC algorithm (top left), the upper confidence bound AF (top right), the probability of improvement AF (bottom left) and the expected improvement AF (bottom right). Note that the algorithms all explore in a different way. In particular, many acquisition functions insist on trying out the boundary of the input interval, while Thompson sampling does not.

Every now and then it happens that an algorithm initially finds the wrong optimum. An example of this is shown in Figure 6.13. An interesting question is whether an algorithm can manage to ‘escape’ from this wrong belief, as it is given more and more measurements. For traditional acquisition functions, the answer may in some cases be a definite ‘no’. For instance for the UCB AF, if the value of $\mu(\mathbf{x}) + \kappa\sigma(\mathbf{x})$ is smaller everywhere than the mean value $\mu(\hat{\mathbf{x}}^*)$ at the believed optimum, as is the case in Figure 6.13,

then no more exploration will ever take place. A similar thing can take place for other acquisition functions. Thompson sampling here is different, because it uses randomness when selecting try-out points. It is hence guaranteed, when given infinitely many measurements, to eventually find the global optimum. However, in certain unfortunate cases this of course may take a while.

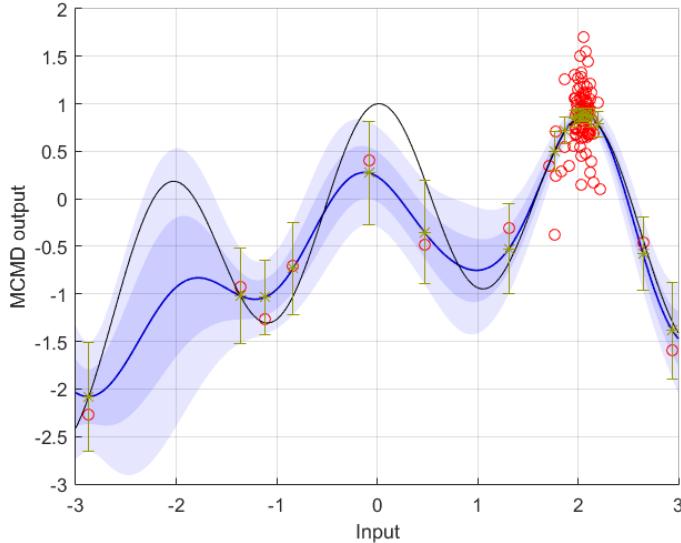


Figure 6.13: An execution run, similar to the ones of Figure 6.12, where initially a wrong optimum is found. This often takes place due to an unfortunate case of noise when a point near the true optimum is tried out.

This is confirmed when we examine how the regret evolves as we run the various algorithms. This regret is shown in Figure 6.14. We see that, in this particular problem, Thompson sampling through the MCMD algorithm does slightly worse than other algorithms. This is caused exactly by cases like those shown in Figure 6.13, where initially a wrong belief of the maximum is obtained. Because Thompson sampling is partly focused on exploiting this (incorrect) maximum, it stays around there for a little while. On the other hand, acquisition functions like the EI and PI AFs are focused on getting as much information as possible. As a result, in the cases where these acquisition functions escape their wrong belief of the optimum, they do so relatively soon.

It may seem that the EI and PI AFs are hence better optimization algorithms. However, this is certainly not always the case, as the next example will show us.

6.5.2. OPTIMIZING A TWO-DIMENSIONAL FUNCTION

The next function we try to optimize is the two-dimensional Branin function. It is used often as test function in optimization literature, for instance by Dixon and Szegő (1978). The Branin function is defined as

$$f(x_1, x_2) = \left(x_2 - \frac{51x_1^2}{40\pi^2} + \frac{5x_1}{\pi} - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos(x_1) + 10, \quad (6.78)$$

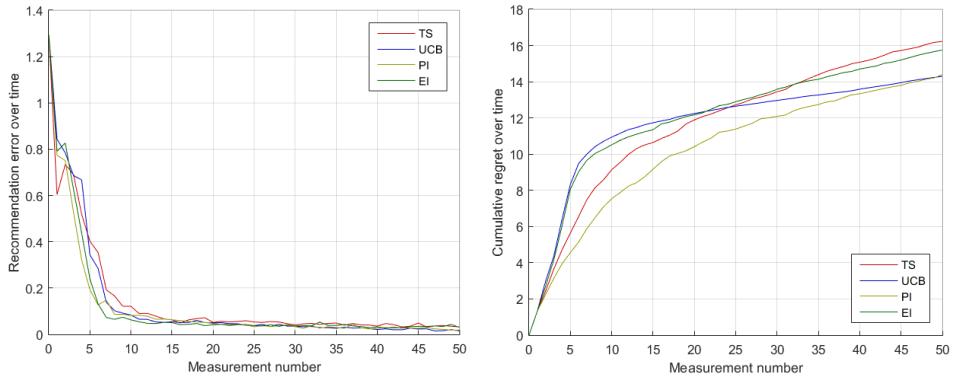


Figure 6.14: The instantaneous (left) and cumulative (right) regret, of various GPO algorithms, as more measurements are added. To reduce the effect of ‘unfortunate events’, a total of 50 full executions of the algorithms was run, and their average was plotted. For the error minimization set-up, the endpoint of the left graph is crucial, while for the (cumulative) regret minimization set-up, the endpoint of the right graph is the quantity that needs to be optimized.

where $x_1 \in [-5, 10]$ and $x_2 \in [0, 15]$. The challenge is to *minimize* this function. We are dealing with maximizing functions here, so instead we will *maximize* minus this function. This function is shown in Figure 6.15.

6

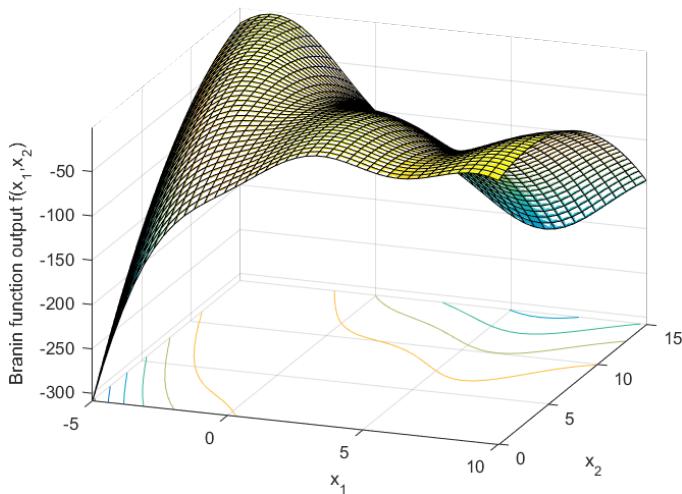


Figure 6.15: A plot of minus the Branin function (6.78). Note the three optima at $(-\pi, \frac{491}{40})$, $(\pi, \frac{91}{40})$ and $(3\pi, \frac{99}{40})$, all at a value of $-\frac{5}{4\pi}$.

A special property of the Branin function is that it has three identical global optima. Their locations can be found analytically as $(-\pi, \frac{491}{40})$, $(\pi, \frac{91}{40})$ and $(3\pi, \frac{99}{40})$, and these optima all have a value of $-\frac{5}{4\pi}$. (Or minus this value for minus the Branin function.)

When optimizing the function, we use the same set-up as in our previous experiment. For the measurement noise, we use $\hat{\sigma}_{f_m} = 5$. As parameters for the various optimization methods we use $\kappa = 2$ and $\xi = 2$, with α once more decreasing from $\frac{2}{3}$ to $\frac{1}{6}$ as we go. When we run the algorithm, we get regret developments like those shown in Figure 6.16, and here we see that this time Thompson sampling performs much better than the alternatives.

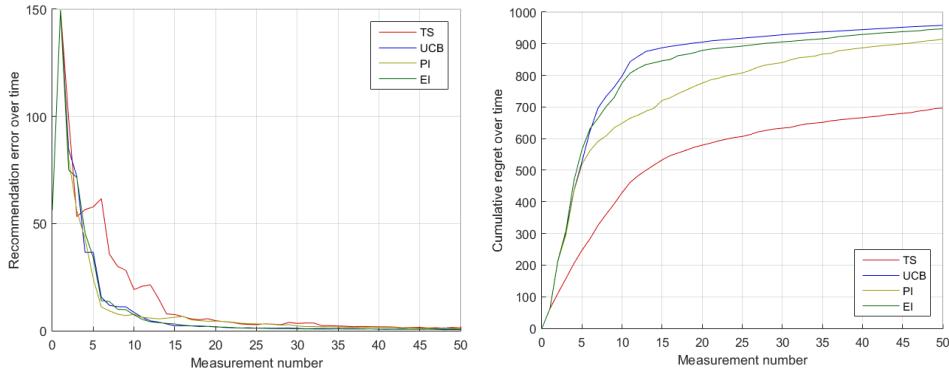


Figure 6.16: The instantaneous (left) and cumulative (right) regret, of various GPO algorithms, as more measurements are added. Again a total of 50 full executions of the algorithms was run, and their average was plotted. While information-based acquisition functions manage to find the optimum sooner, Thompson sampling appears to do so with less regret.

It is interesting to study why Thompson sampling is doing so much better than the other algorithms. We can do so by looking at which try-out points the various algorithms pick. These are shown in Figure 6.17. Here we see that the three acquisition functions often wind up picking try-out points at the border of the input space. In particular, they always try the four corners of the input space. Because one of these corners at input $(-5, 0)$ has a significantly low value (even less than -300), this results in a large loss. Because Thompson sampling is more random in choosing its try-out points, it avoids this bad point and hence performs better.

From this analysis, we see that we *cannot* conclude that Thompson sampling works significantly better than the other algorithms. In fact, any conclusion stating that an optimization always works better than other optimization methods is unlikely to be true. The exact problem has a very strong effect. We *can* conclude that Thompson sampling works better on this specific problem. It may still work less well on other problems though.

One upside of using Thompson sampling together with the MCMD algorithm is that we can study the maximum distribution of the resulting Gaussian process. (Even though you can also apply the MCMD algorithm to the resulting GP from the other acquisition functions.) Such a maximum distribution is shown in Figure 6.18.

Note that in the case of Figure 6.18 the algorithm has managed to find all three optimums. This is not always the case. All optimization methods usually find two of the three maximums, and regularly they find all three, but this is by no means guaranteed given the limited number of measurements.

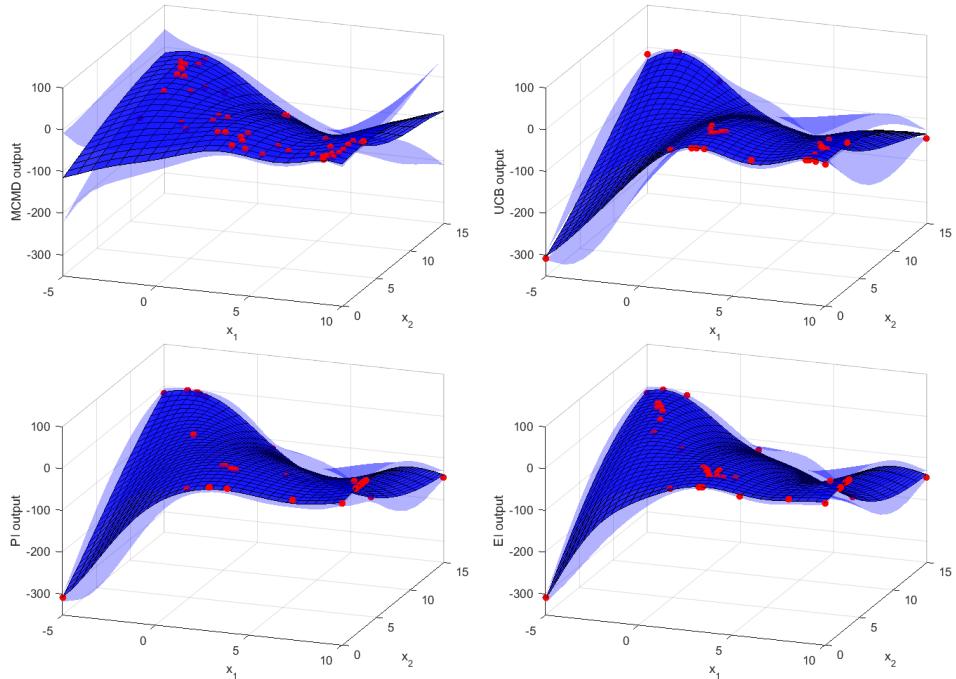


Figure 6.17: The measurements that were obtained, and the subsequent function approximation, for the various optimization methods. Note that all acquisition functions try input points on the edges/corners of the input space, while Thompson sampling does not.

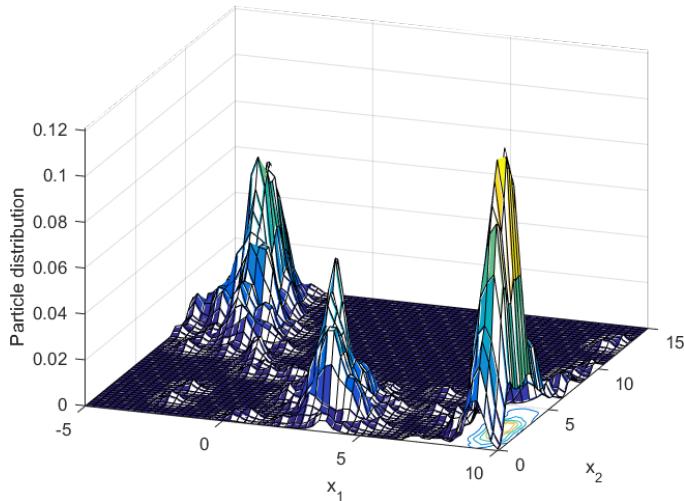


Figure 6.18: The maximum distribution corresponding to the GP of Figure 6.17 (top left) as approximated by the MCMD algorithm. The algorithm has managed to find all three optima, some with more confidence than others. Some stray particles still reside in the lesser explored regions, where the variance is high. In the part of the input space which has been explored but found suboptimal, there are no more particles remaining.

6.5.3. A WIND TURBINE SIMULATION SYSTEM

It is time to apply our methods to a more practical problem: a wind turbine simulation. More specifically, we use a linearized version of the so-called TURBU model, developed 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 use a linearized version of the full TURBU model, to keep the computational time required to run all the optimization methods multiple times within reasonable bounds. This linearized model has only ten states. First there is the angular position ψ (the *azimuth*) of the turbine, and its derivative, the rotor speed Ω . Secondly, there is the forward/backward position of the nacelle due to bending of the tower, as well as its derivative. Then a single bending mode is also incorporated for each wind turbine blade, resulting in the remaining six states.

The turbine simulation also has several inputs. Some of these cannot be controlled. For instance, there is a wind speed for each of the blades, in which both the rotation of the wind turbine and a representative turbulence spectrum are taken into account. Others can be controlled, like the torque applied by the turbine generator. Through a simple proportional controller, this is used to keep the rotor speed constant. Finally, there are also flaps installed on the turbine blades, and it is through the control of these flaps that we should reduce the loads within the wind turbine.

As output of the model, we first of all have basic data like the angular position and velocity of the turbine, and the forward/backward motion of the nacelle. However, we also measure the wind speed encountered by each of the blades and, more importantly, the bending moment at the root of each of the blades, around each of the three axes. Of these three moments, the in-plane bending and the twisting (torsion) of the blades are of lesser importance. It is mainly the out-of-plane flapping moment of the blades which is problematic, because it causes significant bending of the thin wind turbine blades, with high stresses as a result. We will refer to this as the *Root Bending Moment* (RBM). An example of such RBMs can be seen in Figure 6.19 (left).

6.5.4. MULTIPLE REFERENCE FRAMES: THE COLEMAN TRANSFORMATION

For all the parameters related to the blades, we can choose in which reference frame we want them. We can either use these parameters as they are in real life, which is known as the *rotating reference frame*. So then we would know the (for instance) bending loads of each individual blade.

However, we could also apply the *Coleman transformation*, also known as the *multi-blade coordinate transformation*. Let's assume we are dealing with a three-bladed wind turbine. We write the azimuth angle of each of the blades as ψ_1 , ψ_2 and ψ_3 . So if the first blade is pointing upwards, then $\psi_1 = 0^\circ$, $\psi_2 = 120^\circ$ and $\psi_3 = 240^\circ$. We also write the corresponding RBMs (or any other blade parameters) as q_1 , q_2 and q_3 . We can now

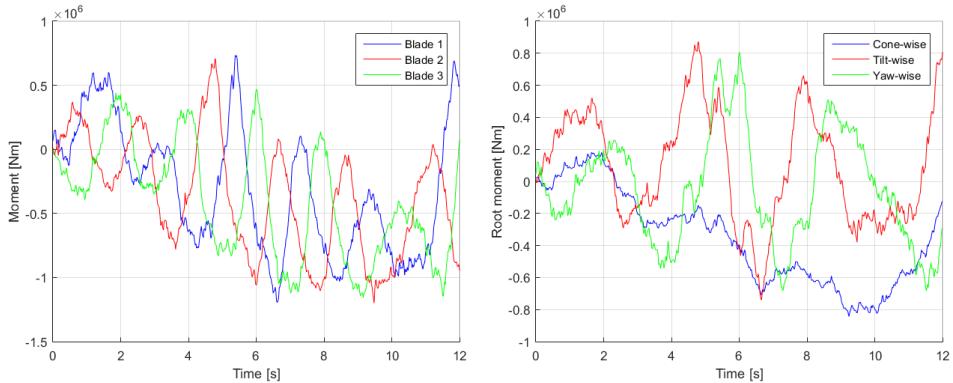


Figure 6.19: The root bending moments of the wind turbine in flapping direction. This is plotted in the rotating reference frame (left) and the fixed reference frame (right). In the rotating reference frame you can clearly see the oscillation of the RBMs of the various blades as the turbine rotates. In the fixed reference frame the signals do not vary with the turbine azimuth, but only due to other processes like turbulence.

transform these parameters into the so-called *fixed reference frame* through⁶

$$\begin{bmatrix} q_0 \\ q_t \\ q_y \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 2\cos(\psi_1) & 2\cos(\psi_2) & 2\cos(\psi_3) \\ 2\sin(\psi_1) & 2\sin(\psi_2) & 2\sin(\psi_3) \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}. \quad (6.79)$$

The first of these parameters is known as the *collective* or *cone-wise* RBM (or whatever blade parameter we transform). It reflects the amount in which the wind turbine blades are bending all together. The second parameter is known as the *tilt-wise* or *pitch-wise* RBM. It is large if blades that are pointing up are bending one way and blades that are pointing down are bending the other way. The third parameter is called the *yaw-wise* or *flap-wise* RBM, and it is large if blades pointing to the right are bending one way and blades pointing to the left are bending the other way. Ideally, in a perfectly symmetric world, it is zero. An example of this transformation being applied is shown in Figure 6.19 (right).

We can also apply the *inverse Coleman transformation* to go from the fixed to the rotating reference frame. This follows as

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} 1 & \cos(\psi_1) & \sin(\psi_1) \\ 1 & \cos(\psi_2) & \sin(\psi_2) \\ 1 & \cos(\psi_3) & \sin(\psi_3) \end{bmatrix} \begin{bmatrix} q_0 \\ q_t \\ q_y \end{bmatrix}, \quad (6.80)$$

where the above matrix is the inverse of the Coleman transformation matrix.

When working with wind turbines, it is generally more convenient to work within the fixed reference frame. That is, we keep track of the RBMs within the fixed reference frame, and then use these to determine how much we should deflect the flaps, also within the fixed reference frame.

⁶As always, there is a multitude of notation conventions in literature. Also the order in which the elements appear within the vector may differ. So when applying this transformation, always check which element represents which mode.

When we do this, we should ignore the collective RBM. One reason for this is that the collective RBM does not cause any fatigue damage. After all, while the other two RBMs represent cyclic loads, the collective RBM represents constant loads, and constant loads do not cause fatigue damage. At the same time, the main way to reduce the collective RBM is to slow the wind turbine down. In other words, reducing the collective RBM gets in the way of the generator controller, which we do not want. As a result, we will only use the tilt-wise and yaw-wise RBMs to determine how much to deflect the flaps in a tilt-wise and yaw-wise fashion.

The way in which we do this is rather simple: we use a proportional controller. In other words, we take the RBMs in the fixed reference frame, multiply them by a constant gain, and feed those signals to the blade flaps. The reason behind this overt simplicity mainly lies in the conservativeness of the wind turbine industry. Many wind turbine manufacturers prefer proven technologies above new and potentially risky algorithms. And if there is any control method which is simple and effective (though not necessarily optimal) then it is the proportional controller. The main question now is: which controller gains should we use?

6.5.5. A QUALITY CRITERION: THE DAMAGE EQUIVALENT LOAD

To know which controller gains are optimal, we need a quality criterion. What are we actually trying to optimize?

In short, we are trying to maximize the lifetime of the wind turbine by reducing the fatigue damage. But measuring how much ‘fatigue’ a blade has absorbed is not as easy as it may seem, because the blade may be subject to lots of small oscillations, a few big ones, and anything in-between, as is shown in Figure 6.19. How do we compare all this?

The first step is to analyze how many oscillations of which magnitude our blades have been subject to. This is done through a method called *rainflow counting*. A proper explanation is given by Nieslony (2009), but we briefly take a look at it anyway with the help of Figure 6.20.

To start the rainflow counting procedure, we take the RBM plot and only consider the peaks (the turning points) in it. We then rotate the plot so it resembles a ‘roof’ with potential rain running down. The idea is to introduce a ‘flow’ of rain at every point where the graph turns to the right. This rain slides down the roof, until the plot turns back to the left, at which point the rain falls down to the ‘roof’ directly beneath it. When it lands on this roof, it encounters an already existing flow of rain. The shorter one of these two is cut off, and the longer one continues. At the end, when we have set up all the rain flows in this way, we look at the sizes of all the rain flows. These are the magnitudes of the oscillations. It hence gives us an overview of all the various oscillation stress magnitudes S_1, S_2, \dots, S_{n_s} we have encountered, with n_s the number of oscillations.

We now know the magnitudes of all the individual oscillations we encountered. The next step is to compare them with each other. To do this, we ideally need a so-called *S-N Curve*. Here, S is the stress magnitude of the oscillations which the blade encounters, and N is the number of such oscillations which the material can survive before failing, also known as the *life cycles*. Naturally, if we subject a piece of material to stronger oscillations (higher stress S) it fails after fewer of these oscillations (lower lifetime N). The speed at which these oscillations is applied generally has little influence.

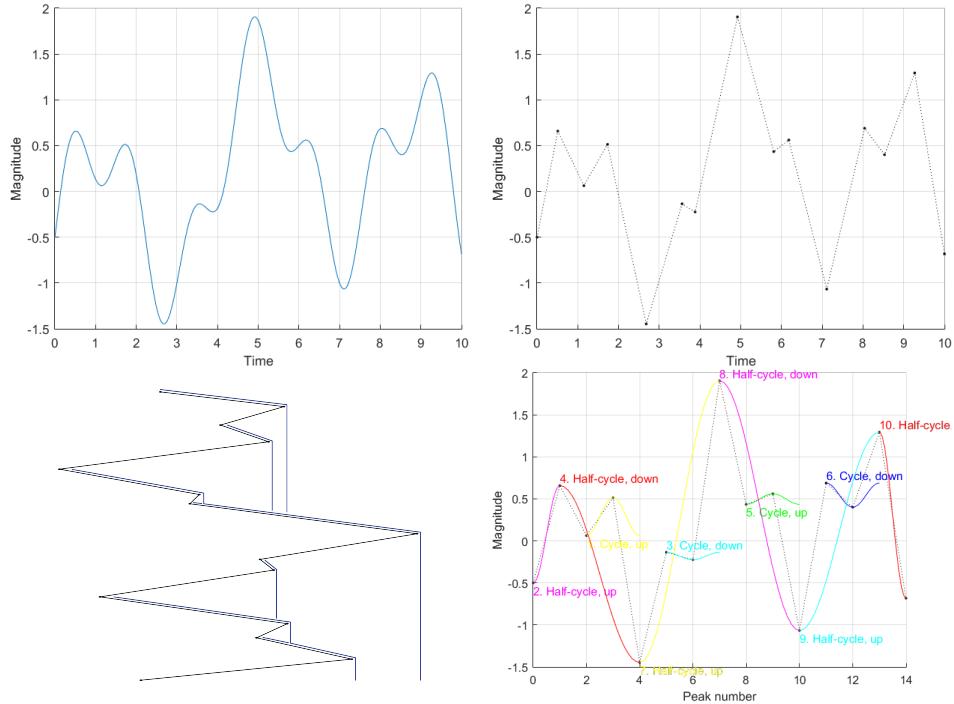


Figure 6.20: The rainflow procedure visualized. We start with a signal to analyze (top left). We then extract the peaks of this signal (top right). Everything else is irrelevant. We turn the plot sideways and let rain flow down from every rightward turning point (bottom left). When two flows meet, the 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 \(2009\)](#).

The S-N curve is generally plotted using a logarithmic scale for N . In this case, it often takes a mostly linear shape. This means that N and S approximately satisfy a relation of the form $SN^m = \text{constant}$. This means that, if the oscillation stress S becomes twice as big, the number of life cycles is reduced by a factor 2^m .

The factor m here is known as the *Wöhler exponent*. It depends on the material used. According to [Savenije and Peeringa \(2009\)](#), our glass fiber composite blades have a Wöhler exponent of $m = 11$. This is a very significant number. After all, if we get twice as much stress, the number of life cycles is reduced by a factor $2^{11} = 2048$. Or in other words, a single oscillation of magnitude $2S$ is equivalent to 2048 oscillations of magnitude S . This idea of ‘equivalence’ is also known as *Miner’s rule*. Using it, we can calculate a so-called *Damage Equivalent Load* (DEL).

Suppose that we have subjected our blade to a time t of oscillations. During this time it has encountered all sorts of oscillations S_1, S_2, \dots, S_{n_s} , as evaluated by our rainflow counting algorithm. Using this, we can calculate the number of equivalent oscillations N_{eq} at a given reference stress amplitude S_r . This is done through⁷

$$N_{eq} = \left(\frac{S_1}{S_r} \right)^m + \left(\frac{S_2}{S_r} \right)^m + \dots + \left(\frac{S_{n_s}}{S_r} \right)^m. \quad (6.81)$$

In practice, we often look at this in a different way. Instead of looking at the number of equivalent oscillations for a given reference stress magnitude, we look at the equivalent stress magnitude for a given number of oscillations. In other words, we ask ourselves, ‘If we had encountered N_r oscillations, all of the same magnitude S_{eq} , what would this equivalent stress S_{eq} be that results in exactly the same fatigue damage as our whole assortment of oscillations?’ We refer to S_{eq} as the damage equivalent load and we can calculate it through

$$S_{eq} = S_r \left(\frac{N_{eq}}{N_r} \right)^{\frac{1}{m}} = \left(\frac{S_1^m + S_2^m + \dots + S_{n_s}^m}{N_r} \right)^{\frac{1}{m}}. \quad (6.82)$$

How do we choose N_r though? To do so, we usually look at the time t over which we analyzed the stress oscillations. We then pick a reference frequency f_r (often $f_r = 1 \text{ Hz}$) and use $N_r = f_r t$. As a result of this, we can now say that all oscillations put together give the same fatigue damage as a sinusoidal stress signal with frequency f_r and magnitude S_{eq} . This is irrespective of the duration of our experiment.

So in our application, we want to minimize this damage equivalent load, which we now know how to calculate. However, there is a second side of the story, which is that more aggressive controllers will almost always do better at reducing the DEL. We do not want very aggressive controllers though, because they provide too many highly varying input signals to the flaps. This will wear out the bearings on the flaps.

The lifetime of bearings is often measured in the amount of angular distance traveled. So we should also take into account this distance traveled. Or, normalizing per unit of time, we should take into account the mean rate of change of the input signal provided to the flaps. By setting up a weighted combination of the DEL and the mean

⁷If the rainflow counting algorithm indicates that only half of an oscillation has taken place, instead of a full oscillation, we should add a factor $\frac{1}{2}$ to the corresponding oscillation term.

rate of change of the input signal, we can come up with a performance (or damage) score. The goal of the algorithm now is to minimize this performance score, or equivalently to maximize its reciprocal.

6.5.6. OPTIMIZING THE CONTROLLER SETTINGS OF A WIND TURBINE

It is time to discuss the actual experiment. Sadly, code for this experiment is not available online, since TURBU is third-party software and would also take up much more memory space than is available in the online repository. However, the method that is used is the same as for the two-dimensional problem of Section 6.5.2.

For our simulation, we will use a wind speed of 10m/s, with a representative turbulence spectrum. For every experiment, we put the wind turbine in the zero initial state, which it has been linearized about. We then simulate for a relatively long time of $T = 200$ s. This minimizes the influence of random effects like turbulence, reducing the noise in the resulting parameters, like the damage equivalent load.

As was mentioned at the end of Section 6.5.4, we will use a proportional controller for both the tilt-wise and the yaw-wise flap input signals. The signal that is fed to this proportional controller are the tilt-wise and yaw-wise RBMs. That gives us two gains to tune. 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. While the first will result in a large fatigue damage, the latter will result in a high bearing damage. Since our performance score is a weighted combination of these two parameters, the optimum should lie somewhere in-between.

To get an idea of what the performance score plot will look like, we can apply a brute force method. That is, we simply apply 500 random control settings and apply GP regression to the outcome. This results in Figure 6.21. In real life we can of course not apply such an approach. Trying out random control laws like this, without good reason, would result in an unacceptable amount of fatigue damage, if not anything worse.

Figure 6.21 shows us that the performance score is a mostly convex function with respect to the controller gains. There does not seem to exist any local optima. As a result, after some basic manual tuning of the parameters of the acquisition functions, we wound up with the low values of $\kappa = 1$ and $\xi = 0.005$. This shows that this problem is not so much about exploration – finding the optimum – but mainly one of exploitation – sticking with the first optimum found.

When we apply the various optimization methods to this problem, we get the results shown in Figure 6.22. They are very similar to the the results of the one-dimensional problem shown in Figure 6.14. Once more, it seems that the UCB and the EI acquisition functions still do quite some exploring, starting off with the corners of the input space, while both Thompson sampling and the PI acquisition function are more focused on exploiting. This gets them better results in the short term, but in the long run the performance of the various algorithms is once more similar.

It should be noted here that the various optimization methods directly optimize (a function of) the DEL of the wind turbine. Many controller tuning algorithms cannot do so, because of the highly nonlinear nature of the DEL. Instead, they optimize other related quantities, assuming that this would have a beneficial effect on the DEL. This may

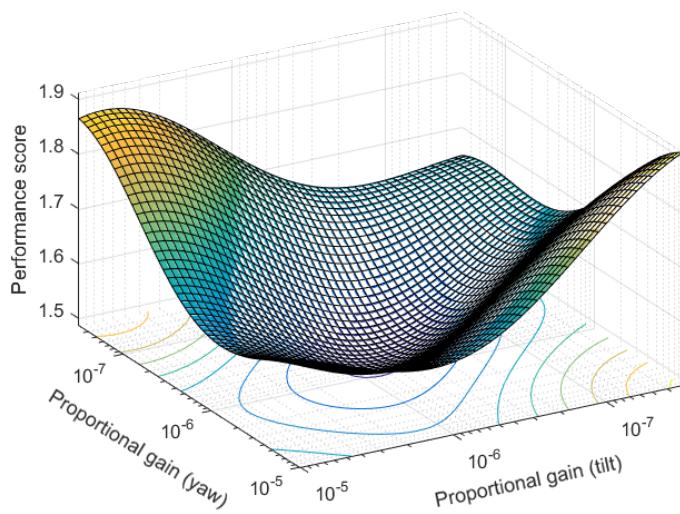


Figure 6.21: An approximation of the wind turbine performance score, with respect to the controller gains. The approximation was made by taking 500 random points and applying a GP regression algorithm to the outcome.

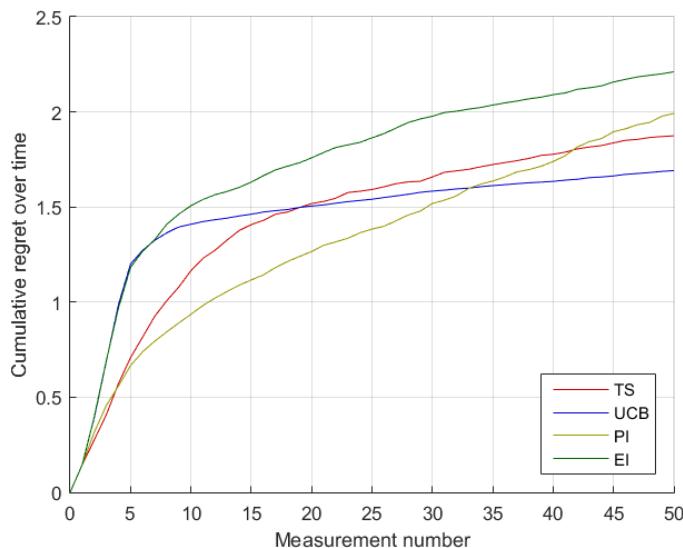


Figure 6.22: The cumulative regret of the various GPO algorithms when applied to the wind turbine problem. Again a total of 50 full executions of the algorithms was run, and their average was plotted.

not always be the case though, which makes Gaussian process optimization, definitely for this application, all the more powerful.

6.5.7. 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 6.23. A two-bladed wind turbine works slightly different from a three-bladed turbine, but the differences are not significant to how GPO is applied.



6

Figure 6.23: The wind turbine used in the experiment, placed in the open-jet wind tunnel of the Delft University of Technology.

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

$$u(t) = A \sin(\psi + \phi), \quad (6.83)$$

where A is the amplitude and ϕ is the phase shift of the input signal. Equivalently, we can also rewrite the above (see Theorem A.40) to the form

$$u(t) = \theta_1 \sin(\psi) + \theta_2 \cos(\psi). \quad (6.84)$$

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, be-

cause restricting the control signal to a sinusoidal form already removes the risk of winding up with an overly aggressive controller.

For this experiment only $n = 15$ measurements were performed, and only for Thompson sampling. These measurements, together with the resulting approximation of the damage equivalent load, is shown in Figure 6.24 (left). The corresponding belief about where the optimal parameters are is shown in Figure 6.24 (right).

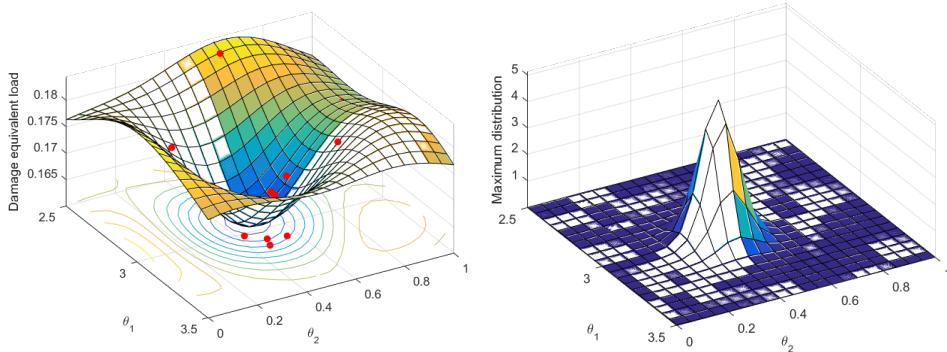


Figure 6.24: The results of the wind turbine experiment, with the approximated damage equivalent load mean (left) and the belief of where the optimal controller settings would be (right). A total of $n = 15$ measurements were performed using Thompson sampling. During these experiments, the Thompson sampling 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

There are two questions that arise from this experiment. First of all, are the resulting parameters correct? Naturally, this is hard to know for sure. However, the controller parameters that were found 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* (IFT) just before the experiment. IFT fundamentally works very differently from GPO. For instance, IFT is not capable of using the highly nonlinear quality criterion of the DEL, while GPO is able to deal with this. But the fact that both IFT and GPO come up with similar controller parameters is at least an indication that these parameters are somewhat sensible.

The second question is why only $n = 15$ experiments were run. These are practical reasons. Changing the controller settings required shutting off the wind turbine controller which, for safety reasons, required shutting off the entire wind tunnel. As a result, running each experiment took quite some time.

Still, more than fifteen experiments were planned. But after the fifteenth experiment, which involved a slightly longer wind tunnel downtime (read: a coffee break⁸), something had mysteriously changed in the wind turbine dynamics, and the optimal controller parameters were suddenly very different, closer to $\theta_1 = 2.5$ and $\theta_2 = 0$. Later on it turned out that the clamp fixing the yaw angle of the turbine had sprung loose, turning it into a freely yawing wind turbine, thus altering the dynamics. Though the algorithm

⁸To be honest, a water break. I don't drink coffee.

subsequently did mostly converge to the new optimum, albeit with a rather strange DEL approximation, this did defeat the purpose of the experiment and so the experiment was aborted.

However, this does raise the question of whether the algorithm can adapt to such changing circumstances. One of the main assumptions behind the algorithm is that the objective function, albeit noisy, is at least constant in time. If it is not, we need to look into ways of taking this into account. The final two suggestions of Section 4.5.2 may provide a starting point for this.

6.5.8. LESSONS LEARNED FROM THE OPTIMIZATION EXPERIMENTS

There are various lessons that were learned from the optimization experiments. We will discuss a few of them here.

A first lesson, not noted previously, was learned from actually implementing all the optimization methods. It seems that implementing an acquisition function is easy, but this is not always the case. Especially when implementing the PI and EI acquisition functions, there were problems in the optimization algorithm.

These problems were initially caused because the acquisition function was so small as to (numerically) be zero for most of the input space. This was solved by instead using the logarithm of the acquisition function. But then still often only local optimums of the acquisition function were found. This was in turn solved by optimizing the acquisition function through a multi-start approach specifically tuned to this problem.

The lesson from this is that using any optimization method always requires some actual fine-tuning of that method to the specific problem we are applying it to. And the more we fine-tune our optimization method to the problem, the better it performs. It is hence nearly impossible to conclude that one optimization method works better than another, because it all depends on how much we fine-tune it to the problem.

We can learn the second lesson from comparing the one-dimensional experiment of Section 6.5.1 to the two-dimensional experiment of Section 6.5.2. While Thompson sampling performed worst in the first experiment, it performed best in the second experiment. However, this was mainly caused by the function that was being optimized, together with certain habits of the optimization method used. Sometimes an optimization method appears to work well on a certain function and sometimes it does not. But since the function that is optimized is in advance unknown, it is very hard to predict which method will work best.

As a result, we can again conclude that it is impossible to say that one optimization method is better than another. It all depends on the problem used. Although this does provide an extra argument in favor of using the portfolio methods discussed in Section 6.4.5. To be precise, it is always wise to keep track of how any optimization method is doing while it is optimizing a function, and whether or not it might be wiser to use another optimization method.

6.6. OVERVIEW OF LITERATURE AND CONTRIBUTIONS

As usual, we close this chapter off with a literature overview (Section 6.6.1) and some suggestions for further research (Section 6.6.2) for people eager to expand on this excit-

ing field.

6.6.1. LITERATURE OVERVIEW

In this chapter we considered two separate topics. The first is finding the maximum distribution of a Gaussian process and the second is the Gaussian process optimization problem.

THE MAXIMUM DISTRIBUTION

For the maximum distribution, relatively little literature is available. The idea of a maximum distribution was noted by [Lizotte \(2008\)](#), though he did not actually calculate it. It was calculated by [Villemonteix et al. \(2009\)](#) through the same brute force method we also applied in Figure 6.2. An expansion to this was developed by [Hennig and Schuler \(2012\)](#), who used the expectation propagation method from [Minka \(2001\)](#) to approximate the minimum distribution. Though their approximation method was reasonably accurate, it had a runtime of $\mathcal{O}(n^4)$, making it infeasible to apply to most problems. An alternative method was described by [Hernández-Lobato et al. \(2014a\)](#), with further elaborations in the supplement [Hernández-Lobato et al. \(2014b\)](#), where function samples were approximated through a finite number of basis functions and then optimized to find samples from the maximum distribution. Other than that, I have not managed to find any mentions in literature of the maximum distribution of a Gaussian process.

GAUSSIAN PROCESS OPTIMIZATION

The second matter we discussed, Gaussian process optimization, has received a lot more attention in literature. Naturally there is a lot of literature on function optimization in general, but most methods assume that the function can be easily evaluated, that derivative data is known and/or that the function is concave (for maximization) or convex (for minimization). A good overview of such optimization methods is given by [Boyd and Vandenberghe \(2004\)](#). However, we considered the optimization of a function of which every function evaluation is expensive. An overview of this problem is given by [Jones et al. \(1998\)](#). In particular, we used a Bayesian approach to approximate the function during the optimization process, resulting in the Gaussian process optimization problem.

We noted in Section 6.4.1 that there are two different set-ups for the GPO problem. Most of the literature focuses on the error minimization set-up. Proper introductions into this problem are given by [Brochu et al. \(2010\)](#), [Shahriari et al. \(2016\)](#). In short, the PI acquisition function was first suggested by [Kushner \(1964\)](#) and later expanded through contributions by [Torn and Zilinskas \(1989\)](#), [Jones \(2001\)](#), adding the exploration/exploitation parameter ξ . The EI acquisition function was suggested by [Mockus et al. \(1978\)](#). Improvements were then made by [Osborne \(2010\)](#), who added multi-step lookahead, [Park and Law \(2015\)](#), who added a trust region to ensure small changes to the belief of the optimal input \hat{x}^* , and [Brochu et al. \(2010\)](#), who introduced an additional exploration/exploitation parameter ξ similar to the one used in the PI acquisition function. An analysis was performed by [Vazquez and Bect \(2010\)](#) who established convergence bounds for certain specific cases.

There has been slightly less (though still a significant amount) of literature on the regret minimization set-up. The problem itself has been analyzed by for instance [Kleinberg \(2004\)](#), [Grünewälder et al. \(2010\)](#), [de Freitas et al. \(2012\)](#), deriving bounds on the

regret for specific cases. The most well-known technique is the UCB algorithm, which was proposed by [Cox and John \(1997\)](#) and analyzed by [Srinivas et al. \(2012\)](#).

A more advanced method for the error minimization set-up was entropy search. The main idea 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. The method was subsequently developed further as *predictive entropy search* by [Hernández-Lobato et al. \(2014a\)](#).

Portfolio methods were first introduced by [Hoffman et al. \(2011\)](#), who used results from [Auer et al. \(1995\)](#), [Chaudhuri et al. \(2009\)](#). The idea of using a portfolio of acquisition functions was then expanded on by [Shahriari et al. \(2014\)](#), who suggested to use the change in entropy as criterion to select recommendations.

Thompson sampling was first suggested by [Thompson \(1933\)](#), but it has been mostly ignored afterwards. This changed in the late 1990s when it was independently rediscovered several times within the machine learning community. It has been the subject of a lot of analyses, with recent results given by [Chapelle and Li \(2011\)](#), [Agrawal and Goyal \(2012\)](#). However, it has generally been applied to problems with a finite number of possible input points (the armed bandit problem) and not to a problem with infinitely many different inputs. That is, until I wrote about it through [Bijl et al. \(2017b\)](#).

Finally, there is also an enormous list of applications of GPO methods. A few recent examples include the work by [Johan Dahlin \(2015\)](#), [Gutmann and Corander \(2015\)](#), [Marco et al. \(2016\)](#), but this list can pretty much be made as long as desired. If you want to learn of more applications, then read some of the above references. They are bound to mention several.

6.6.2. SUGGESTIONS FOR FURTHER RESEARCH

Gaussian process optimization is still a very active research field. There are plenty of open problems which are worthwhile to look into. I will mention a few which I am personally very curious about.

- **Using less particles for Thompson sampling**

What is the effect of the number of particles used? When we want to approximate the maximum distribution, using more particles will give us a more accurate approximation. But if we are only using the maximum distribution to get samples to use in Thompson sampling, then what happens when we use less particles? For instance, if we only need ten samples from the maximum distribution, does it suffice to use only ten champion particles? Are the samples still adequate then? What problems occur when we really use too few particles? And at how many (or how few) particles do these problems start to occur?

- **Using varying hyperparameters during the optimization**

So far we have assumed that the hyperparameters of the Gaussian process are known and constant. In reality, as was also discussed in Section 3.1.2, the hyperparameters are also random variables, with their own posterior distributions. And ideally we need to take all possible hyperparameters into account. This should be possible to implement within the MCMD algorithm. Now, every time we challenge a champion particle through (6.8) (or through (6.19) if we use multiple challengers per cham-

pion), we should also randomly pick our hyperparameters from the hyper-posterior and use those during the challenge. This should then give us the maximum distribution of the Gaussian process taking into account all possible hyperparameters. But how actually can we do this efficiently, so that we don't have to wait hours for every challenge round?

- **A Gaussian process gradient ascent algorithm**

We have learned in Section 2.5 that the derivative of a Gaussian process is also a Gaussian process. This raises the question of whether it is possible to use Gaussian process regression to implement some sort of gradient ascent algorithm. In particular, can we use our knowledge about the derivative of the Gaussian process to quickly track down a local optimum of the function? And what role does the uncertainty (the variance) of the derivative play in the step size that we use?

- **Low-risk Gaussian process optimization**

In wind turbine applications, and in many other applications, there may be regions of the input space which we simply should never try out. That is, if we pick an input \mathbf{x} (a set of controller parameters) from such a region, the wind turbine becomes unstable and breaks down, with a large amount of damage as a result. Assuming that the cost function $f(\mathbf{x})$ we are approximating is continuous, and that we can potentially see such no-go-regions coming, how do we prevent ourselves from entering such parts of the input space?

In this problem, it is not only important what the upper bound $\mu(\mathbf{x}) + \kappa\sigma(\mathbf{x})$ – the potential gain – of the GP is, but also what the lower bound $\mu(\mathbf{x}) - \kappa\sigma(\mathbf{x})$ – the potential loss – of the GP is. As a result, we can only try input points \mathbf{x}_k close to earlier input points we have tried, so that we are reasonably certain we do not accidentally enter such a very-negative-value region.

Since finding the global optimum is in this case nearly impossible – we cannot risk trying a fully unexplored part of the input space – we can be satisfied with a local optimum for this problem. So possibly Gaussian gradient ascent algorithms (see the previous suggestion) can also be used? Similar ideas have already been tried out, after I first wrote this, by [Sui et al. \(2015\)](#), [Berkenkamp et al. \(2016\)](#).

7

CONCLUSION AND RECOMMENDATIONS

At the start of this thesis, in Section 1.1.2, we asked ourselves four questions related to Gaussian process regression. We will repeat them here one by one to see if we have found an answer to them. In addition, we also consider the next questions that inevitably follow from the answers that we found.

1. *How can GP regression be applied to a big and constantly growing data set?*

We looked into this in Chapter 4. Methods for Gaussian process regression to be applied to large data sets already existed. They made use of inducing input points. We have extended these methods so they can be applied in an online way. That is, both new measurement data and new inducing input points can be added on the fly. These methods are powerful enough that we are now able to deal with big and constantly growing data sets.

Nevertheless, the number of inducing input points required still scales exponentially with the dimension of the input space. As a result, when dealing with problems that both have a large number of input parameters (say, 6 or more) and require a high accuracy, we still run into computational problems because of too many (more than a thousand) inducing input points. Apart from the multitude of small ways in which the methods can be improved, I think this is the most important follow-up problem to tackle. Can we devise a way such that not all inducing input points are required when making predictions?

2. *How can GP regression be applied subject to uncertainty in all measurements?*

In Chapter 5 we developed the SONIG algorithm which allows Gaussian process regression to be applied subject to noisy input points. This has solved the problem that we were facing. The algorithm has shown to have better performance than related algorithms and is computationally more efficient as well.

The next challenge for the development of the SONIG algorithm is to properly deal with various special cases. For instance, for some measurements the algorithm is not

capable of properly finding the posterior distribution of the measurement input. In other cases measurements cannot be incorporated due to numerical problems. Currently such measurements are ignored. By detecting such measurements and dealing with them in the proper way, an extra gain can still be obtained, both concerning performance and concerning the user-friendliness of the SONIG toolbox.

3. How can Gaussian processes be optimized with respect to various parameters?

In Chapter 6 we looked at what the ‘optimum’ of a Gaussian process actually means. It turns out to be a random variable with its own distribution. This distribution cannot be calculated analytically, so we have developed the Monte Carlo Maximum Distribution (MCMD) algorithm to approximate it. Using this algorithm, it is possible to set up a Thompson sampling optimization method which finds the optimum of an unknown nonlinear function with little cost (regret). The performance of this Gaussian process optimization method was comparable to or slightly better than other optimization methods, depending on the exact function used.

The next step in the development of the MCMD algorithm is to see if any performance bounds can be obtained. When applying Thompson sampling to problems with a discrete input space, bounds on the cumulative regret can be derived. The question now is whether such bounds also exist for the continuous-input problem.

4. How can other people apply GP regression algorithms to wind turbine problems?

Applying Gaussian process regression is not always as easy as we may like. Nevertheless, in this thesis I have tried to make it as easy as possible. The past six chapters provide master students, or anyone with a similar background, with an intuitive view on Gaussian processes. It also lists the equations through which people can quickly set up their own Gaussian process regression methods. In addition, all the corresponding mathematics is available in the appendices and all the source code is online (see [Bijl \(2016a\)](#)). This allows people to easily find examples of working GP regression applications; especially applications related to wind energy.

That does not mean that we are done. There is still a long way to go. One step in the right direction has been made by developing the SONIG toolbox. (See [Bijl \(2016b\)](#).) However, no toolbox is ever finished. If we want Gaussian process regression to be applied more, then I believe the challenge is *not* to develop new advanced regression tricks. Instead, I think we should focus on improving toolboxes like the SONIG toolbox, making them more powerful and especially more user-friendly, so the power of Gaussian process regression can be harnessed by nearly anyone.

Concluding, we have answered the questions posed in the introduction, but several more new challenges have sprung up. There is still a long way to go before Gaussian process regression can be easily applied to various applications, in wind energy and beyond.

A

MATRIX ALGEBRA

Summary — Matrices can be subject to a variety of operations. Of course it is possible to add up and multiply matrices, but we can also take the trace of a matrix, the derivative, the vectorization, or multiply matrices using the Kronecker product. All these operators have various potentially useful properties.

We can also take the inverse of a matrix. This is usually a computationally demanding process. But if we split the matrix up into blocks, and we already know the inverse of some of these blocks, we might have an easier time finding the inverse of the full matrix.

Matrices play a fundamental role in Gaussian exponentials as well. When multiplying Gaussian exponentials, we can often find the outcome in relatively easy way by evaluating a few matrix expressions.

Finally we consider Lyapunov equations. Their solutions, which can be found analytically, satisfy a variety of interesting properties. For instance, Lyapunov solutions generally equal integrals over matrix exponentials. These integrals can also be solved by evaluating matrix exponential expressions. Often this latter method is easier to apply. However, if the time over which we integrate is large, then it will result in numerical inaccuracies, so then it will still be better to solve the respective Lyapunov equations.

A

This appendix covers a couple of important theorems related to matrix algebra. To follow this appendix, it is important that you are familiar with matrices and their basic properties. So you should be able to do matrix multiplications, find inverses and determinants, and know what eigenvalues are. No other prior knowledge is required.

We start by examining some basic matrix operations, like the derivative of a matrix, the trace and the vectorization (Section A.1). Then we look at various ways of efficiently calculating matrix inverses, based on what data we already have (Section A.2). We continue by studying Gaussian exponentials, which use covariance matrices in their exponents (Section A.3). Afterwards we study Lyapunov equations and how we can also find their solutions through infinite integrals over matrix exponentials (Section A.4). We also look at an alternative way to solve integrals over matrix exponentials (Section A.5). Finally, we have some miscellaneous theorems which did not fit in anywhere else (Section A.6).

A.1. MATRIX OPERATIONS

Various operations can be applied to matrices. Here we look at a few of them, and what properties result from these operations. We start by examining the trace operator (Section A.1.1), continue with matrix derivatives (Section A.1.2) and then consider both matrix vectorization and the Kronecker product (Section A.1.3).

A.1.1. THE TRACE OPERATOR

The *trace operator* $\text{tr}(P)$ is defined as the sum of the diagonal elements of the square matrix P . In other words,

$$\text{tr}(P) = \text{tr} \left(\begin{bmatrix} P_{11} & P_{12} & \cdots \\ P_{21} & P_{22} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \right) = P_{11} + P_{22} + \dots \quad (\text{A.1})$$

This trace operator has a couple of very convenient properties. First of all, it is a linear operator. That is, $\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)$ and $\text{tr}(cA) = c\text{tr}(A)$ for square matrices A and B and scalar c . (You can proof this directly from the definition.) Secondly, we always have $\text{tr}(P) = \text{tr}(P^T)$, while for a scalar we even have $\text{tr}(c) = c$. And thirdly, the trace operator has the *cyclic property*, as explained by the following theorem.

Theorem A.1. *For any matrices A , B and C for which ABC is square, we have*

$$\text{tr}(ABC) = \text{tr}(CAB) = \text{tr}(BCA). \quad (\text{A.2})$$

Proof. We will first prove the relation $\text{tr}(PQ) = \text{tr}(QP)$ for some $m \times n$ matrix P and $n \times m$ matrix Q . Let's denote the element of P in row i and column j as P_{ij} and similarly for Q or any matrix [...]. The definition of matrix multiplication tells us that

$$[PQ]_{ij} = \sum_{k=1}^n P_{ik}Q_{kj}. \quad (\text{A.3})$$

From this, it follows that

$$\text{tr}(PQ) = \sum_{i=1}^m [PQ]_{ii} = \sum_{i=1}^m \sum_{j=1}^n P_{ij}Q_{ji} = \sum_{j=1}^n \sum_{i=1}^m Q_{ji}P_{ij} = \sum_{j=1}^n [QP]_{jj} = \text{tr}(QP). \quad (\text{A.4})$$

This relation directly implies (A.2). To be precise, substituting $P = AB$ and $Q = C$ gives the first relation, and substituting $P = CA$ and $Q = B$ gives the second. \square

Note that we can only cycle the elements within the trace function. We generally do *not* have $\text{tr}(ABC) = \text{tr}(CBA)$.

A.1.2. MATRIX DERIVATIVES

The *matrix derivative* is defined element-wise. So, if an $m \times n$ matrix P depends on the parameter x , then per definition

$$\frac{dP}{dx} = \begin{bmatrix} \frac{dP_{11}}{dx} & \dots & \frac{dP_{1n}}{dx} \\ \vdots & \ddots & \vdots \\ \frac{dP_{m1}}{dx} & \dots & \frac{dP_{mn}}{dx} \end{bmatrix}. \quad (\text{A.5})$$

If this is the derivative of a matrix, then what is the derivative of a matrix inverse? That question is answered by the following theorem.

Theorem A.2. *For any invertible matrix P and any parameter x , the derivative of P^{-1} with respect to x equals*

$$\frac{dP^{-1}}{dx} = -P^{-1} \frac{dP}{dx} P^{-1}. \quad (\text{A.6})$$

Proof. Consider the relation $PP^{-1} = I$. If we take the derivative of both sides, applying the chain rule, we get

$$\frac{dP}{dx} P^{-1} + P \frac{dP^{-1}}{dx} = 0. \quad (\text{A.7})$$

Solving for dP^{-1}/dx directly proves the theorem. \square

Similarly, what is the derivative of a matrix determinant?

Theorem A.3. *For any invertible matrix P , the derivative of $|P|$ is given by*

$$\frac{d|P|}{dx} = |P| \text{tr}\left(P^{-1} \frac{dP}{dx}\right). \quad (\text{A.8})$$

Proof. This proof is a bit too lengthy for this thesis. I will only note that this theorem is a special case of Jacobi's formula, and for details refer to the work by Bellman (1997) or Magnus and Neudecker (1999). \square

A nice consequence of the above theorem is that

$$\frac{d \log |P|}{dx} = \text{tr}\left(P^{-1} \frac{dP}{dx}\right), \quad (\text{A.9})$$

which somewhat surprisingly is an easier expression than A.8.

A

A.1.3. VECTORIZATION AND THE KRONECKER PRODUCT

The *vectorization* $\text{vec}(P)$ of some matrix P is defined as the concatenation of the columns of P into one big vector. For instance,

$$\text{vec}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = \begin{bmatrix} a \\ c \\ b \\ d \end{bmatrix}. \quad (\text{A.10})$$

The *Kronecker product* $P \otimes Q$ of an $m_P \times n_P$ matrix P and an $m_Q \times n_Q$ matrix Q is defined as the $m_P m_Q \times n_P n_Q$ matrix

$$P \otimes Q = \begin{bmatrix} P_{11}Q & \cdots & P_{1n}Q \\ \vdots & \ddots & \vdots \\ P_{m1}Q & \cdots & P_{mn}Q \end{bmatrix}. \quad (\text{A.11})$$

These two operators are often used together, mainly because of the following theorem.

Theorem A.4. *For any matrices P of size $k \times l$ and Q of size $l \times m$, it holds that*

$$\text{vec}(PQ) = (I_m \otimes P) \text{vec}(Q) = (Q^T \otimes I_k) \text{vec}(P), \quad (\text{A.12})$$

where I_k denotes the identity matrix of size k .

Proof. This can be proven by expanding the matrix equations. Let's write the matrix Q as

$$Q = [\mathbf{q}_1 \quad \mathbf{q}_2 \quad \cdots \quad \mathbf{q}_m] = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{k1} & q_{k2} & \cdots & q_{km} \end{bmatrix}. \quad (\text{A.13})$$

We can first note that $\text{vec}(PQ)$ equals

$$\text{vec}(PQ) = \text{vec}([P\mathbf{q}_1 \quad P\mathbf{q}_2 \quad \cdots \quad P\mathbf{q}_m]) = \begin{bmatrix} P\mathbf{q}_1 \\ P\mathbf{q}_2 \\ \vdots \\ P\mathbf{q}_m \end{bmatrix}. \quad (\text{A.14})$$

Next, consider $(I_m \otimes P) \text{vec}(Q)$. This equals

$$(I_m \otimes P) \text{vec}(Q) = \begin{bmatrix} P & 0 & \cdots & 0 \\ 0 & P & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_m \end{bmatrix} = \begin{bmatrix} P\mathbf{q}_1 \\ P\mathbf{q}_2 \\ \vdots \\ P\mathbf{q}_m \end{bmatrix}. \quad (\text{A.15})$$

So these two quantities are equal. Finally we examine $(Q^T \otimes I_k) \text{vec}(P)$. This equals

$$\begin{aligned} (Q^T \otimes I_k) \text{vec}(P) &= \begin{bmatrix} q_{11}I_k & q_{21}I_k & \cdots & q_{l1}I_k \\ q_{12}I_k & q_{22}I_k & \cdots & q_{l2}I_k \\ \vdots & \vdots & \ddots & \vdots \\ q_{1m}I_k & q_{2m}I_k & \cdots & q_{lm}I_k \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_l \end{bmatrix} \\ &= \begin{bmatrix} q_{11}\mathbf{p}_1 + q_{21}\mathbf{p}_2 + \dots + q_{l1}\mathbf{p}_l \\ q_{12}\mathbf{p}_1 + q_{22}\mathbf{p}_2 + \dots + q_{l2}\mathbf{p}_l \\ \vdots \\ q_{1m}\mathbf{p}_1 + q_{2m}\mathbf{p}_2 + \dots + q_{lm}\mathbf{p}_l \end{bmatrix} = \begin{bmatrix} P\mathbf{q}_1 \\ P\mathbf{q}_2 \\ \vdots \\ P\mathbf{q}_m \end{bmatrix}. \end{aligned} \quad (\text{A.16})$$

So this quantity is also equal to the previous one. \square

It often occurs that we want to find the trace $\text{tr}(PQ)$ of a product of matrices. In that case, we could also use the following theorem.

Theorem A.5. Any matrices P of size $n \times m$ and Q of size $m \times n$ satisfy

$$\text{tr}(PQ) = \text{vec}(P^T)^T \text{vec}(Q) \quad (\text{A.17})$$

Proof. Let's define $R = P^T$. We can write Q and R as

$$Q = [\mathbf{q}_1 \ \mathbf{q}_2 \ \cdots \ \mathbf{q}_n], \quad (\text{A.18})$$

$$R = [\mathbf{r}_1 \ \mathbf{r}_2 \ \cdots \ \mathbf{r}_n]. \quad (\text{A.19})$$

It follows that

$$\begin{aligned} \text{tr}(R^T Q) &= \text{tr} \left(\begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \vdots \\ \mathbf{r}_n^T \end{bmatrix} [\mathbf{q}_1 \ \mathbf{q}_2 \ \cdots \ \mathbf{q}_n] \right) = \text{tr} \left(\begin{bmatrix} \mathbf{r}_1^T \mathbf{q}_1 & \mathbf{r}_1^T \mathbf{q}_2 & \cdots & \mathbf{r}_1^T \mathbf{q}_n \\ \mathbf{r}_2^T \mathbf{q}_1 & \mathbf{r}_2^T \mathbf{q}_2 & \cdots & \mathbf{r}_2^T \mathbf{q}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{r}_n^T \mathbf{q}_1 & \mathbf{r}_n^T \mathbf{q}_2 & \cdots & \mathbf{r}_n^T \mathbf{q}_n \end{bmatrix} \right) \\ &= \mathbf{r}_1^T \mathbf{q}_1 + \mathbf{r}_2^T \mathbf{q}_2 + \dots + \mathbf{r}_n^T \mathbf{q}_n = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_n \end{bmatrix}^T \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_n \end{bmatrix} = \text{vec}(R)^T \text{vec}(Q). \end{aligned} \quad (\text{A.20})$$

Replacing R by P^T will complete the proof. \square

A.2. MATRIX INVERSES

In this section we will consider various ways of inverting various matrices. We will often use the notation Δ_A and Δ_D . These are *Schur complements*, defined respectively as

$$\Delta_A = D - CA^{-1}B, \quad (\text{A.21})$$

$$\Delta_D = A - BD^{-1}C. \quad (\text{A.22})$$

This notation will come in handy when we invert blockwise matrices (Section A.2.1) and when we invert sums of matrices (Section A.2.2).

A

A.2.1. BLOCKWISE MATRIX INVERSES

Our first theorem now shows how we can invert a blockwise matrix. The theory behind this is far from new, with a good overview given by Hager (1989).

Theorem A.6. *Assume that A , D , Δ_A and Δ_D are invertible. For a matrix P written in blockwise form, we can find the inverse P^{-1} through one of two equations,*

$$\begin{aligned} P^{-1} &= \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B\Delta_A^{-1}CA^{-1} & -A^{-1}B\Delta_A^{-1} \\ -\Delta_A^{-1}CA^{-1} & \Delta_A^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \Delta_D^{-1} & -\Delta_D^{-1}BD^{-1} \\ -D^{-1}C\Delta_D^{-1} & D^{-1} + D^{-1}C\Delta_D^{-1}BD^{-1} \end{bmatrix}. \end{aligned} \quad (\text{A.23})$$

Proof. The trick to prove this theorem is to transform the matrix P into something with a diagonal form. To start, we can transform it into something with an upper triangular form using a left-multiplication

$$\begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix}. \quad (\text{A.24})$$

Alternatively, we can transform it into something with a lower triangular form using a right-multiplication

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & D - CA^{-1}B \end{bmatrix}. \quad (\text{A.25})$$

If we apply both transformations, then we get a diagonal matrix

$$\begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix}. \quad (\text{A.26})$$

If we instead find the inverse transformations, then we can write our matrix P as LDU , with L a lower (block-)triangular matrix, D a (block-)diagonal matrix and U an upper (block-)triangular matrix. The resulting decomposition of P is hence called an *LDU decomposition*. It is given by

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}^{-1} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix}^{-1}. \quad (\text{A.27})$$

If we invert both sides, making use of $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$, we find that

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & (D - CA^{-1}B)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}. \quad (\text{A.28})$$

By expanding this, we find that P^{-1} equals

$$P^{-1} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}. \quad (\text{A.29})$$

This proves the first half of the theorem.

The proof of the second half is nearly identical, except for one small difference. Previously we used a left-multiplication to put the matrix in an upper triangular form, and a right-multiplication to put it in a lower triangular form. Now we do the opposite. That is, we use a left-multiplication to put the matrix in a lower triangular form, and a right-multiplication to put it in an upper triangular form. This gives us

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} A - BD^{-1}C & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ -D^{-1}C & I \end{bmatrix}^{-1}. \quad (\text{A.30})$$

The rest of the steps are the same. So we find that

$$\begin{aligned} \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} I & 0 \\ -D^{-1}C & I \end{bmatrix} \begin{bmatrix} (A - BD^{-1}C)^{-1} & 0 \\ 0 & D^{-1} \end{bmatrix} \begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} (A - BD^{-1}C)^{-1} & -(A - BD^{-1}C)^{-1}BD^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & D^{-1} + D^{-1}C(A - BD^{-1}C)^{-1}BD^{-1} \end{bmatrix}. \end{aligned} \quad (\text{A.31})$$

This also proves the second half of the theorem. \square

The next theorem we look at is the *matrix inversion lemma*, which is also known as the *Woodbury matrix identity* or one of various other names. It is a familiar theorem, found for instance in the work by Hager (1989) and Higham (2002), but we repeat it here so we have a complete overview of important theorems.

Theorem A.7. *Assume that A, D, Δ_A and Δ_D are invertible. Then we have*

$$\begin{aligned} \Delta_D^{-1} &= (A - BD^{-1}C)^{-1} \\ &= A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} \\ &= A^{-1} + A^{-1}B\Delta_A^{-1}CA^{-1}. \end{aligned} \quad (\text{A.32})$$

Proof. This follows directly from Theorem A.6. If we look at the top left block of the matrices in this theorem, we directly find that

$$(A - BD^{-1}C)^{-1} = A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}. \quad (\text{A.33})$$

Combining this with the definitions for Δ_A and Δ_D proves the theorem. \square

In literature the matrix inversion lemma has lots of different notations. Often U is used instead of B , V is used instead of C and $-C^{-1}$ is used instead of D . Sometimes B is turned into $-B$, resulting in

$$(A + BD^{-1}C)^{-1} = A^{-1} - A^{-1}B(D + CA^{-1}B)^{-1}CA^{-1}. \quad (\text{A.34})$$

The essence of the theorem remains the same though: to replace a possibly difficult matrix inverse with one that is potentially a lot easier to compute.

A similar theorem to the matrix inversion lemma is the following one.

Theorem A.8. *Assume that A, D, Δ_A and Δ_D are invertible. Then we have*

$$A^{-1}B(D - CA^{-1}B)^{-1} = A^{-1}B\Delta_A^{-1} = \Delta_D^{-1}BD^{-1} = (A - BD^{-1}C)^{-1}BD^{-1}. \quad (\text{A.35})$$

A

Proof. This theorem also follows directly from Theorem A.6. If we look at the top right block of the matrices in this theorem, we directly find the equation above. \square

Finally, using the results so far, we can derive another related theorem. This one concerns the determinant of a blockwise matrix.

Theorem A.9. *The determinant of a blockwise matrix equals*

$$\begin{aligned} \begin{vmatrix} A & B \\ C & D \end{vmatrix} &= |A||D - CA^{-1}B| = |A||\Delta_A| \\ &= |D||A - BD^{-1}C| = |D||\Delta_D|. \end{aligned} \quad (\text{A.36})$$

Proof. We should first note, from the definition of the determinant, that

$$\begin{vmatrix} X & 0 \\ 0 & Y \end{vmatrix} = |X||Y|, \quad (\text{A.37})$$

$$\begin{vmatrix} I & 0 \\ Z & I \end{vmatrix} = 1, \quad (\text{A.38})$$

and $|XYZ| = |X||Y||Z|$ for any (appropriately sized) matrices X , Y and Z . Using this, and using the LDU decomposition (A.27), the above theorem directly follows. \square

A.2.2. INVERTING SUMS OF MATRICES

It often occurs that we need to find the inverse of a sum of inverses. In that case the following theorem can come in handy.

Theorem A.10. *Assume that all matrices with inverses in the below equation are invertible. Also define P and Q as in the equation below. We then have*

$$\begin{aligned} (P^{-1} + Q^{-1})^{-1} &= \left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} + \begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} \right)^{-1} \\ &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \tilde{A}^{-1} + \tilde{A}^{-1}\tilde{B}\Delta_{\tilde{A}}^{-1}\tilde{C}\tilde{A}^{-1} & -\tilde{A}^{-1}\tilde{B}\Delta_{\tilde{A}}^{-1} \\ -\Delta_{\tilde{A}}^{-1}\tilde{C}\tilde{A}^{-1} & \Delta_{\tilde{A}}^{-1} \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} \\ &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \Delta_{\tilde{D}}^{-1} & -\Delta_{\tilde{D}}^{-1}\tilde{B}\tilde{D}^{-1} \\ -\tilde{D}^{-1}\tilde{C}\Delta_{\tilde{D}}^{-1} & \tilde{D}^{-1} + \tilde{D}^{-1}\tilde{C}\Delta_{\tilde{D}}^{-1}\tilde{B}\tilde{D}^{-1} \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}, \end{aligned} \quad (\text{A.39})$$

where we have defined

$$\begin{bmatrix} \tilde{A} & \tilde{B} \\ \tilde{C} & \tilde{D} \end{bmatrix} = \begin{bmatrix} A+E & B+F \\ C+G & D+H \end{bmatrix}, \quad (\text{A.40})$$

$$\Delta_{\tilde{A}} = \tilde{D} - \tilde{C}\tilde{A}^{-1}\tilde{B} = (D+H) - (C+G)(A+E)^{-1}(B+F), \quad (\text{A.41})$$

$$\Delta_{\tilde{D}} = \tilde{A} - \tilde{B}\tilde{D}^{-1}\tilde{C} = (A+E) - (B+F)(D+H)^{-1}(C+G). \quad (\text{A.42})$$

Proof. The easiest way to prove the above theorem is through

$$\begin{aligned} (P^{-1} + Q^{-1})^{-1} &= P(P+Q)^{-1}Q \\ &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} A+E & B+F \\ C+G & D+H \end{bmatrix}^{-1} \begin{bmatrix} E & F \\ G & H \end{bmatrix} \\ &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \tilde{A} & \tilde{B} \\ \tilde{C} & \tilde{D} \end{bmatrix}^{-1} \begin{bmatrix} E & F \\ G & H \end{bmatrix}. \end{aligned} \quad (\text{A.43})$$

We can apply Theorem A.6 to this. Making use of definitions (A.21) and (A.22) for $\Delta_{\tilde{A}}$ and $\Delta_{\tilde{D}}$, we now immediately find that

$$\begin{aligned} (P^{-1} + Q^{-1})^{-1} &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \tilde{A}^{-1} + \tilde{A}^{-1}\tilde{B}\Delta_{\tilde{A}}^{-1}\tilde{C}\tilde{A}^{-1} & -\tilde{A}^{-1}\tilde{B}\Delta_{\tilde{A}}^{-1} \\ -\Delta_{\tilde{A}}^{-1}\tilde{C}\tilde{A}^{-1} & \Delta_{\tilde{A}}^{-1} \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} \\ &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \Delta_{\tilde{D}}^{-1} & -\Delta_{\tilde{D}}^{-1}\tilde{B}\tilde{D}^{-1} \\ -\tilde{D}^{-1}\tilde{C}\Delta_{\tilde{D}}^{-1} & \tilde{D}^{-1} + \tilde{D}^{-1}\tilde{C}\Delta_{\tilde{D}}^{-1}\tilde{B}\tilde{D}^{-1} \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}. \end{aligned} \quad (\text{A.44})$$

This proves the theorem. \square

We can take the limit of $H \rightarrow \infty$ for the above theorem. Having a matrix limit may sound complicated, but what it basically means is that all eigenvalues of H go to infinity. So one way of looking at this is as if $H = hI$ with $h \rightarrow \infty$.

Theorem A.11. *Consider Theorem A.10. Under the same assumptions and definitions, in the limit of $H \rightarrow \infty$, this theorem reduces to*

$$\begin{aligned} \left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} + \begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} \right)^{-1} &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} - \begin{bmatrix} A \\ C \end{bmatrix} (A+E)^{-1} [A \ B] \\ &= \begin{bmatrix} A(A+E)^{-1}E & E(A+E)^{-1}B \\ C(A+E)^{-1}E & D - C(A+E)^{-1}B \end{bmatrix}. \end{aligned} \quad (\text{A.45})$$

Proof. We could prove this by indeed considering Theorem A.10 as $H \rightarrow \infty$. A somewhat faster approach would be to rewrite $(P^{-1} + Q^{-1})^{-1}$ into

$$P(P+Q)^{-1}Q = P(P+Q)^{-1}(P+Q) - P(P+Q)^{-1}Q = P - P(P+Q)^{-1}P. \quad (\text{A.46})$$

In this case we have

$$(P+Q)^{-1} = \begin{bmatrix} A+E & B+F \\ C+G & D+H \end{bmatrix} = \begin{bmatrix} A+E & * \\ * & \infty \end{bmatrix}^{-1} = \begin{bmatrix} (A+E)^{-1} & 0 \\ 0 & 0 \end{bmatrix}, \quad (\text{A.47})$$

where a * denotes an immaterial value: it will drop out of the equations anyway. From this, we directly find that, as $H \rightarrow \infty$,

$$\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} + \begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} \right)^{-1} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} - \begin{bmatrix} A \\ C \end{bmatrix} (A+E)^{-1} [A \ B]. \quad (\text{A.48})$$

A

This proves the first part of (A.45). To prove the second part, we need to rewrite the above. We can rewrite the top left term through

$$A - A(A+E)^{-1}A = A(A+E)^{-1}(A+E) - A(A+E)^{-1}A = A(A+E)^{-1}E. \quad (\text{A.49})$$

Rewriting the other terms goes in an identical way, after which we also prove the second result of (A.45). \square

A.3. GAUSSIAN EXPONENTIALS

In this section we examine Gaussian exponentials and Gaussian exponential functions. We start with the exponentials (Section A.3.1) which are exponential whose exponent has the form $-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})$. If we then also add a normalization constant $\frac{1}{\sqrt{|2\pi\Sigma|}}$ we wind up with Gaussian exponential functions, whose properties we also study (Section A.3.2). Finally we also look at joint Gaussian exponential functions of multiple variables \mathbf{x}_a and \mathbf{x}_b (Section A.3.3).

A.3.1. MULTIPLYING GAUSSIAN EXPONENTIALS

The first property we examine concerns the product of Gaussian exponentials. It is a familiar expression in literature, given for instance by Deisenroth (2010).

Theorem A.12. *Assume that $\Sigma_a > 0$ and $\Sigma_b > 0$ are symmetric. The product of two Gaussian exponentials (without multiplying constants) equals*

$$\begin{aligned} & \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1}(\mathbf{x} - \boldsymbol{\mu}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_b)^T \Sigma_b^{-1}(\mathbf{x} - \boldsymbol{\mu}_b)\right) \\ &= \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)^T (\Sigma_a + \Sigma_b)^{-1}(\boldsymbol{\mu}_a + \boldsymbol{\mu}_b)\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T (\Sigma_a^{-1} + \Sigma_b^{-1})(\mathbf{x} - \boldsymbol{\mu})\right), \end{aligned} \quad (\text{A.50})$$

where we define

$$\boldsymbol{\mu} = (\Sigma_a^{-1} + \Sigma_b^{-1})^{-1}(\Sigma_a^{-1}\boldsymbol{\mu}_a + \Sigma_b^{-1}\boldsymbol{\mu}_b). \quad (\text{A.51})$$

Proof. To prove this, we start with merging the exponentials and expanding the brackets.

$$\begin{aligned} & \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1}(\mathbf{x} - \boldsymbol{\mu}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_b)^T \Sigma_b^{-1}(\mathbf{x} - \boldsymbol{\mu}_b)\right) \\ &= \exp\left(-\frac{1}{2}(\mathbf{x}^T \Sigma_a^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma_a^{-1} \boldsymbol{\mu}_a + \boldsymbol{\mu}_a^T \Sigma_a^{-1} \boldsymbol{\mu}_a + \mathbf{x}^T \Sigma_b^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma_b^{-1} \boldsymbol{\mu}_b + \boldsymbol{\mu}_b^T \Sigma_b^{-1} \boldsymbol{\mu}_b)\right). \end{aligned} \quad (\text{A.52})$$

We want to complete the squares with respect to \mathbf{x} . That is, within the exponential we want to find something of the form

$$-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}) + c, \quad (\text{A.53})$$

for some parameters $\boldsymbol{\mu}$, Σ^{-1} and c . By looking at the exponential, and checking out all the terms with $\mathbf{x}^T(\dots)\mathbf{x}$, we can right away see that

$$\Sigma_a^{-1} + \Sigma_b^{-1} = \Sigma^{-1}. \quad (\text{A.54})$$

Next, if we look at all the terms of the form $\mathbf{x}^T(\dots)^{-1}(\dots)$, we find that we should have

$$-2\mathbf{x}^T \Sigma_a^{-1} \boldsymbol{\mu}_a - 2\mathbf{x}^T \Sigma_b^{-1} \boldsymbol{\mu}_b = -2\mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}. \quad (\text{A.55})$$

This should hold for all \mathbf{x} . We find that this is indeed the case when

$$\boldsymbol{\mu} = \Sigma(\Sigma_a^{-1} \boldsymbol{\mu}_a + \Sigma_b^{-1} \boldsymbol{\mu}_b). \quad (\text{A.56})$$

To find c , we look at all remaining terms. That is,

$$-\frac{1}{2} (\boldsymbol{\mu}_a^T \Sigma_a^{-1} \boldsymbol{\mu}_a + \boldsymbol{\mu}_b^T \Sigma_b^{-1} \boldsymbol{\mu}_b) = -\frac{1}{2} \boldsymbol{\mu}^T \Sigma^{-1} \boldsymbol{\mu} + c. \quad (\text{A.57})$$

Working this out will require quite some steps. The initial steps are

$$\begin{aligned} -2c &= \boldsymbol{\mu}_a^T \Sigma_a^{-1} \boldsymbol{\mu}_a + \boldsymbol{\mu}_b^T \Sigma_b^{-1} \boldsymbol{\mu}_b - \boldsymbol{\mu}^T \Sigma^{-1} \boldsymbol{\mu} \\ &= \boldsymbol{\mu}_a^T \Sigma_a^{-1} \boldsymbol{\mu}_a + \boldsymbol{\mu}_b^T \Sigma_b^{-1} \boldsymbol{\mu}_b - (\boldsymbol{\mu}_a^T \Sigma_a^{-1} + \boldsymbol{\mu}_b^T \Sigma_b^{-1}) \Sigma \Sigma^{-1} \Sigma (\Sigma_a^{-1} \boldsymbol{\mu}_a + \Sigma_b^{-1} \boldsymbol{\mu}_b) \\ &= \boldsymbol{\mu}_a^T (\Sigma_a^{-1} - \Sigma_a^{-1} \Sigma \Sigma_a^{-1}) \boldsymbol{\mu}_a - 2\boldsymbol{\mu}_a^T (\Sigma_a^{-1} \Sigma \Sigma_b^{-1}) \boldsymbol{\mu}_b + \boldsymbol{\mu}_b^T (\Sigma_b^{-1} - \Sigma_b^{-1} \Sigma \Sigma_b^{-1}) \boldsymbol{\mu}_b. \end{aligned} \quad (\text{A.58})$$

It is now interesting to note that the terms within brackets are all equal. Don't believe me? To see what they are all equal to, we consider

$$\begin{aligned} \Sigma_b^{-1} - \Sigma_b^{-1} \Sigma \Sigma_b^{-1} &= \Sigma^{-1} \Sigma \Sigma_b^{-1} - \Sigma_b^{-1} \Sigma \Sigma_b^{-1} \\ &= (\Sigma^{-1} - \Sigma_b^{-1}) \Sigma \Sigma_b^{-1} \\ &= (\Sigma_a^{-1} + \Sigma_b^{-1} - \Sigma_b^{-1})(\Sigma_a^{-1} + \Sigma_b^{-1})^{-1} \Sigma_b^{-1} \\ &= \Sigma_a^{-1} (\Sigma_a^{-1} + \Sigma_b^{-1})^{-1} \Sigma_b^{-1} \\ &= (\Sigma_a + \Sigma_b)^{-1}. \end{aligned} \quad (\text{A.59})$$

We can show similarly that the other terms within brackets equal $(\Sigma_a + \Sigma_b)^{-1}$. As a result, we have

$$-2c = \boldsymbol{\mu}_a^T (\Sigma_a + \Sigma_b)^{-1} \boldsymbol{\mu}_a - 2\boldsymbol{\mu}_a^T (\Sigma_a + \Sigma_b)^{-1} \boldsymbol{\mu}_b + \boldsymbol{\mu}_b^T (\Sigma_a + \Sigma_b)^{-1} \boldsymbol{\mu}_b \quad (\text{A.60})$$

$$c = -\frac{1}{2} (\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)^T (\Sigma_a + \Sigma_b)^{-1} (\boldsymbol{\mu}_a + \boldsymbol{\mu}_b). \quad (\text{A.61})$$

Now we know Σ , $\boldsymbol{\mu}$ and c . By merging our results, we may write

$$\begin{aligned} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\mathbf{x} - \boldsymbol{\mu}_a)\right) \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_b)^T \Sigma_b^{-1} (\mathbf{x} - \boldsymbol{\mu}_b)\right) \\ = \exp\left(-\frac{1}{2} (\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)^T (\Sigma_a + \Sigma_b)^{-1} (\boldsymbol{\mu}_a + \boldsymbol{\mu}_b)\right) \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right). \end{aligned} \quad (\text{A.62})$$

This proves what we wanted to prove. □

A

A.3.2. MULTIPLYING/DIVIDING GAUSSIAN EXPONENTIAL FUNCTIONS

From multiplying exponentials, it is only a small step further to multiplying Gaussian exponential functions $\mathcal{N}(\dots)$. We have defined the Gaussian exponential function $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ (see (2.12)) as

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) \equiv \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right). \quad (\text{A.63})$$

If we now multiply two Gaussian exponentials, then the result is again a Gaussian exponential function $\mathcal{N}(\dots)$, multiplied by a constant.

Theorem A.13. *The product $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_b, \Sigma_b)$ equals*

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_b, \Sigma_b) = C\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma), \quad (\text{A.64})$$

where C , $\boldsymbol{\mu}$ and Σ are defined through

$$\Sigma = (\Sigma_a^{-1} + \Sigma_b^{-1})^{-1}, \quad (\text{A.65})$$

$$\boldsymbol{\mu} = \Sigma(\Sigma_a^{-1}\boldsymbol{\mu}_a + \Sigma_b^{-1}\boldsymbol{\mu}_b), \quad (\text{A.66})$$

$$C = \mathcal{N}(\boldsymbol{\mu}_a|\boldsymbol{\mu}_b, \Sigma_a + \Sigma_b) = \mathcal{N}(\boldsymbol{\mu}_b|\boldsymbol{\mu}_a, \Sigma_a + \Sigma_b). \quad (\text{A.67})$$

Proof. Making use of Theorem A.12, we find that

$$\begin{aligned} & \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_b, \Sigma_b) \\ &= \frac{1}{\sqrt{|2\pi\Sigma_a|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\mathbf{x}-\boldsymbol{\mu}_a)\right) \frac{1}{\sqrt{|2\pi\Sigma_b|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_b)^T \Sigma_b^{-1} (\mathbf{x}-\boldsymbol{\mu}_b)\right) \\ &= \frac{1}{(2\pi)^{d_x}} \frac{1}{\sqrt{|\Sigma_a||\Sigma_b|}} \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)^T (\Sigma_a + \Sigma_b)^{-1} (\boldsymbol{\mu}_a + \boldsymbol{\mu}_b)\right) \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right). \end{aligned} \quad (\text{A.68})$$

Here we made use of the determinant property that $|cA| = c^d|A|$, with d the dimension of the square matrix A . We have the exponentials sorted out now, but we still need to fix the constants, including the matrix determinants. We will make use of the rules $|A||B| = |AB|$ and $|A^{-1}| = |A|^{-1}$, which allow us to derive

$$\begin{aligned} \frac{1}{(2\pi)^{d_x}} \frac{1}{\sqrt{|\Sigma_a||\Sigma_b|}} &= \frac{1}{(2\pi)^{d_x}} \sqrt{\frac{|\Sigma_a^{-1} + \Sigma_b^{-1}|}{|\Sigma_a||\Sigma_a^{-1} + \Sigma_b^{-1}||\Sigma_b|}} \\ &= \frac{1}{(2\pi)^{d_x}} \sqrt{\frac{|\Sigma_a^{-1} + \Sigma_b^{-1}|}{|\Sigma_a + \Sigma_b|}} \\ &= \frac{1}{\sqrt{(2\pi)^{d_x}|\Sigma_a + \Sigma_b|}} \frac{1}{\sqrt{(2\pi)^{d_x}|(\Sigma_a^{-1} + \Sigma_b^{-1})^{-1}|}}. \end{aligned} \quad (\text{A.69})$$

Using this result, we find that

$$\begin{aligned} & \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_b, \Sigma_b) \\ &= \frac{1}{\sqrt{|2\pi(\Sigma_a + \Sigma_b)|}} \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)^T (\Sigma_a + \Sigma_b)^{-1} (\boldsymbol{\mu}_a + \boldsymbol{\mu}_b)\right) \\ &\quad \frac{1}{\sqrt{|2\pi(\Sigma_a^{-1} + \Sigma_b^{-1})^{-1}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right) \\ &= \mathcal{N}(\boldsymbol{\mu}_a|\boldsymbol{\mu}_b, \Sigma_a + \Sigma_b) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma). \end{aligned} \quad (\text{A.70})$$

This completes the proof of this theorem. \square

Something similar happens when we divide Gaussian functions. The following theorem shows how that works.

Theorem A.14. *The division of $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ by $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a)$ equals*

$$\frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)}{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a)} = \frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_b, \Sigma_b)}{\mathcal{N}(\boldsymbol{\mu}_a|\boldsymbol{\mu}_b, \Sigma_a + \Sigma_b)}, \quad (\text{A.71})$$

where $\boldsymbol{\mu}_b$ and Σ_b are defined through

$$\Sigma_b = (\Sigma^{-1} - \Sigma_a^{-1})^{-1}, \quad (\text{A.72})$$

$$\boldsymbol{\mu}_b = \Sigma_b(\Sigma^{-1}\boldsymbol{\mu} - \Sigma_a^{-1}\boldsymbol{\mu}_a). \quad (\text{A.73})$$

Proof. This theorem directly follows from Theorem A.13. In fact, this theorem states that

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_b, \Sigma_b) = \mathcal{N}(\boldsymbol{\mu}_a|\boldsymbol{\mu}_b, \Sigma_a + \Sigma_b) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma), \quad (\text{A.74})$$

where we have

$$\Sigma = (\Sigma_a^{-1} + \Sigma_b^{-1})^{-1}, \quad (\text{A.75})$$

$$\boldsymbol{\mu} = \Sigma(\Sigma_a^{-1}\boldsymbol{\mu}_a + \Sigma_b^{-1}\boldsymbol{\mu}_b). \quad (\text{A.76})$$

If we rewrite this to

$$\Sigma_b = (\Sigma^{-1} - \Sigma_a^{-1})^{-1}, \quad (\text{A.77})$$

$$\boldsymbol{\mu}_b = \Sigma_b(\Sigma^{-1}\boldsymbol{\mu} - \Sigma_a^{-1}\boldsymbol{\mu}_a), \quad (\text{A.78})$$

then it directly follows that

$$\frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)}{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \Sigma_a)} = \frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_b, \Sigma_b)}{\mathcal{N}(\boldsymbol{\mu}_a|\boldsymbol{\mu}_b, \Sigma_a + \Sigma_b)}. \quad (\text{A.79})$$

\square

A

A.3.3. JOINT GAUSSIAN EXPONENTIAL FUNCTIONS

Suppose that we have a joint Gaussian exponential function

$$\mathcal{N}\left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix}, \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}\right). \quad (\text{A.80})$$

It is actually possible to split this joint exponential function up into two separate Gaussian exponential functions. How that works is explained by the following theorem.

Theorem A.15. *The joint Gaussian exponential function can be rewritten to*

$$\begin{aligned} & \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix}, \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}\right) \\ &= \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \Sigma_{aa}) \mathcal{N}(\mathbf{x}_b | \boldsymbol{\mu}_b + \Sigma_{ba} \Sigma_{aa}^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_a), \Sigma_{bb} - \Sigma_{ba} \Sigma_{aa}^{-1} \Sigma_{ab}) \\ &= \mathcal{N}(\mathbf{x}_b | \boldsymbol{\mu}_b, \Sigma_{bb}) \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a + \Sigma_{ab} \Sigma_{bb}^{-1} (\mathbf{x}_b - \boldsymbol{\mu}_b), \Sigma_{aa} - \Sigma_{ab} \Sigma_{bb}^{-1} \Sigma_{ba}). \end{aligned} \quad (\text{A.81})$$

Proof. Per definition, the joint Gaussian exponential function equals

$$\frac{1}{\sqrt{\left|2\pi \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}\right|}} \exp\left(-\frac{1}{2} \left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} \right)^T \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}^{-1} \left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} \right)\right). \quad (\text{A.82})$$

For the determinant we will apply Theorem A.9 and for the matrix inverse we will use Theorem A.6. For simplicity of notation, we will also write $\Delta_a = \Sigma_{bb} - \Sigma_{ba} \Sigma_{aa}^{-1} \Sigma_{ab}$ and define $\mathbf{y}_a = \mathbf{x}_a - \boldsymbol{\mu}_a$ and $\mathbf{y}_b = \mathbf{x}_b - \boldsymbol{\mu}_b$. We can now rewrite the above to

$$\frac{1}{\sqrt{|2\pi \Sigma_{aa}| |2\pi \Delta_a|}} \exp\left(-\frac{1}{2} \begin{bmatrix} \mathbf{y}_a \\ \mathbf{y}_b \end{bmatrix}^T \begin{bmatrix} \Sigma_{aa}^{-1} + \Sigma_{aa}^{-1} \Sigma_{ab} \Delta_a^{-1} \Sigma_{ba} \Sigma_{aa}^{-1} & \Sigma_{aa}^{-1} \Sigma_{ab} \Delta_a^{-1} \\ \Delta_a^{-1} \Sigma_{ba} \Sigma_{aa}^{-1} & \Delta_a^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{y}_a \\ \mathbf{y}_b \end{bmatrix}\right).$$

We can already split up the above into two separate exponents, through

$$\begin{aligned} \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix}, \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}\right) &= \frac{1}{\sqrt{|2\pi \Sigma_{aa}|}} \exp\left(-\frac{1}{2} \mathbf{y}_a^T \Sigma_{aa}^{-1} \mathbf{y}_a\right) \\ &\quad \frac{1}{\sqrt{|2\pi \Delta_a|}} \exp\left(-\frac{1}{2} \begin{bmatrix} \mathbf{y}_a \\ \mathbf{y}_b \end{bmatrix}^T \begin{bmatrix} -\Sigma_{aa}^{-1} \Sigma_{ab} & I \end{bmatrix} \Delta_a^{-1} \begin{bmatrix} -\Sigma_{ba} \Sigma_{aa}^{-1} \\ I \end{bmatrix} \begin{bmatrix} \mathbf{y}_a \\ \mathbf{y}_b \end{bmatrix}\right). \end{aligned} \quad (\text{A.83})$$

The left half already is in the form of the Gaussian exponential function $\mathcal{N}(\mathbf{y}_a | \mathbf{0}, \Sigma_{aa})$. The right half still needs a bit of work. In particular, by noting that $\Sigma_{ab} = \Sigma_{ba}^T$, we can find that the above equals

$$\mathcal{N}(\mathbf{y}_a | \mathbf{0}, \Sigma_{aa}) \frac{1}{\sqrt{|2\pi \Delta_a|}} \exp\left(-\frac{1}{2} (\mathbf{y}_b - \Sigma_{ba} \Sigma_{aa}^{-1} \mathbf{y}_a)^T \Delta_a^{-1} (\mathbf{y}_b - \Sigma_{ba} \Sigma_{aa}^{-1} \mathbf{y}_a)\right). \quad (\text{A.84})$$

And if we then substitute the \mathbf{y} parameters back to $\mathbf{x} - \boldsymbol{\mu}$ (with the corresponding subscript), and also fill in Δ_a , then we wind up with

$$\mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \Sigma_{aa}) \mathcal{N}(\mathbf{x}_b | \boldsymbol{\mu}_b + \Sigma_{ba} \Sigma_{aa}^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_a), \Sigma_{bb} - \Sigma_{ba} \Sigma_{aa}^{-1} \Sigma_{ab}). \quad (\text{A.85})$$

This proves the first half of (A.81). The second half is proven identically, but then by using the Schur complements of Σ_{bb} instead of that of Σ_{aa} . \square

It is also possible to integrate over Gaussian exponential functions. We should first note here that

$$\int_X \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) d\mathbf{x} = \int_X \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) d\mathbf{x} = 1, \quad (\text{A.86})$$

because the above function is a probability density function. (Also see Section B.4.1.) But what happens when we integrate over only a part of the vector \mathbf{x} ? That is explained by the following theorem.

Theorem A.16. *The joint Gaussian exponential function satisfies the integral*

$$\int_{X_b} \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix}, \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}\right) d\mathbf{x}_b = \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \Sigma_{aa}). \quad (\text{A.87})$$

Proof. If we use Theorem A.15, then we can immediately find that the above integral equals

$$\int_{X_b} \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \Sigma_{aa}) \mathcal{N}(\mathbf{x}_b | \boldsymbol{\mu}_b + \Sigma_{ba} \Sigma_{aa}^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_a), \Sigma_{bb} - \Sigma_{ba} \Sigma_{aa}^{-1} \Sigma_{ab}) d\mathbf{x}_b. \quad (\text{A.88})$$

The first term within this integral does not depend on \mathbf{x}_b . It is hence a constant and can be taken out of the integral. We then remain with

$$\mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \Sigma_{aa}) \int_{X_b} \mathcal{N}(\mathbf{x}_b | \boldsymbol{\mu}_b + \Sigma_{ba} \Sigma_{aa}^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_a), \Sigma_{bb} - \Sigma_{ba} \Sigma_{aa}^{-1} \Sigma_{ab}) d\mathbf{x}_b. \quad (\text{A.89})$$

According to (A.86) this integral equals one, irrespective of the value of \mathbf{x}_a . This therefore completes the proof. \square

A.3.4. OTHER GAUSSIAN EXPONENTIAL RELATIONS

There are many more products, integrals and such of Gaussian exponentials which could be useful. In fact, we need the solutions to a few of them in the main text, and those solutions are derived here.

We start by examining the product of two Gaussian exponential functions, where one of them does not have \mathbf{x} as its main parameter.

Theorem A.17. *For known parameters $a, \mathbf{b}, \Sigma_a, \boldsymbol{\mu}_x$ and Σ_x , it holds that*

$$\begin{aligned} \mathcal{N}(a | \mathbf{b}^T \mathbf{x}, \Sigma_a) \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_x, \Sigma_x) &= \frac{1}{\sqrt{|2\pi\Sigma_a||2\pi\Sigma_x|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) \\ &\quad \exp\left(-\frac{1}{2}(a \Sigma_a^{-1} a + \boldsymbol{\mu}_x^T \Sigma_x^{-1} \boldsymbol{\mu}_x - \boldsymbol{\mu}^T \Sigma \boldsymbol{\mu})\right), \end{aligned} \quad (\text{A.90})$$

where we have defined

$$\Sigma = (\mathbf{b} \Sigma_a^{-1} \mathbf{b}^T + \Sigma_x^{-1})^{-1}, \quad (\text{A.91})$$

$$\boldsymbol{\mu} = \Sigma(\mathbf{b} \Sigma_a^{-1} a + \Sigma_x^{-1} \boldsymbol{\mu}_x) = \boldsymbol{\mu}_x + \Sigma \mathbf{b} \Sigma_a^{-1} (a - \mathbf{b}^T \boldsymbol{\mu}_x). \quad (\text{A.92})$$

A

Proof. Expanding the Gaussian exponentials gives us

$$\begin{aligned} \mathcal{N}(a|\mathbf{b}^T \mathbf{x}, \Sigma_a) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_x, \Sigma_x) &= \frac{1}{\sqrt{|2\pi\Sigma_a||2\pi\Sigma_x|}} \exp\left(-\frac{1}{2} (\mathbf{a} - \mathbf{b}^T \mathbf{x})^T \Sigma_a^{-1} (\mathbf{a} - \mathbf{b}^T \mathbf{x})\right) \\ &\quad \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_x)^T \Sigma_x (\mathbf{x} - \boldsymbol{\mu}_x)\right). \end{aligned} \quad (\text{A.93})$$

Our main focus will be on the exponentials. Currently, we can write them as

$$\begin{aligned} &\exp\left(-\frac{1}{2} (\mathbf{a}\Sigma_a^{-1}\mathbf{a} - \mathbf{a}\Sigma_a^{-1}\mathbf{b}^T \mathbf{x} - \mathbf{x}^T \mathbf{b}\Sigma_a^{-1}\mathbf{a} + \mathbf{x}^T \mathbf{b}\Sigma_a^{-1}\mathbf{b}^T \mathbf{x}\right. \\ &\quad \left.+ \mathbf{x}^T \Sigma_x^{-1} \mathbf{x} - \mathbf{x}^T \Sigma_x^{-1} \boldsymbol{\mu}_x - \boldsymbol{\mu}_x^T \Sigma_x^{-1} \mathbf{x} + \boldsymbol{\mu}_x^T \Sigma_x^{-1} \boldsymbol{\mu}_x\right)). \end{aligned} \quad (\text{A.94})$$

We would like to write them as

$$\begin{aligned} &\exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) + \text{remaining terms}\right) \\ &= \exp\left(-\frac{1}{2} (\mathbf{x}^T \Sigma^{-1} \mathbf{x} - \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu} - \boldsymbol{\mu}^T \Sigma^{-1} \mathbf{x} + \boldsymbol{\mu}^T \Sigma^{-1} \boldsymbol{\mu}) + \text{remaining terms}\right). \end{aligned} \quad (\text{A.95})$$

This tells us that we should use $\Sigma^{-1} = \mathbf{b}\Sigma_a^{-1}\mathbf{b}^T + \Sigma_x^{-1}$. It follows that

$$\mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu} = \mathbf{x}^T (\mathbf{b}\Sigma_a^{-1}\mathbf{a} + \Sigma_x^{-1} \boldsymbol{\mu}_x), \quad (\text{A.96})$$

which gives us our first relation for $\boldsymbol{\mu}$. We can prove that the second relation for $\boldsymbol{\mu}$ also holds through

$$\begin{aligned} \boldsymbol{\mu} &= \Sigma (\mathbf{b}\Sigma_a^{-1}\mathbf{a} + \Sigma_x^{-1} \boldsymbol{\mu}_x) \\ &= \Sigma (\mathbf{b}\Sigma_a^{-1}\mathbf{a} - \mathbf{b}\Sigma_a^{-1}\mathbf{b}^T \boldsymbol{\mu}_x + \mathbf{b}\Sigma_a^{-1}\mathbf{b}^T \boldsymbol{\mu}_x + \Sigma_x^{-1} \boldsymbol{\mu}_x) \\ &= \Sigma (\mathbf{b}\Sigma_a^{-1} (\mathbf{a} - \mathbf{b}^T \boldsymbol{\mu}_x) + \Sigma^{-1} \boldsymbol{\mu}_x) \\ &= \boldsymbol{\mu}_x + \Sigma \mathbf{b}\Sigma_a^{-1} (\mathbf{a} - \mathbf{b}^T \boldsymbol{\mu}_x). \end{aligned} \quad (\text{A.97})$$

Bringing all the remaining terms (the ones without \mathbf{x}) together in one final exponential will complete the proof. \square

It may also happen that we need to integrate over the product over a Gaussian exponential and a Gaussian exponential function. Let's see what that results in.

Theorem A.18. *For known parameters \mathbf{x}_a , Λ , $\boldsymbol{\mu}$ and Σ , it holds that*

$$\begin{aligned} \int_X \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}_a)^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}_a)\right) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) d\mathbf{x} \\ = \sqrt{\frac{|\Lambda|}{|\Lambda + \Sigma|}} \exp\left(-\frac{1}{2} (\mathbf{x}_a - \boldsymbol{\mu})^T (\Lambda + \Sigma)^{-1} (\mathbf{x}_a - \boldsymbol{\mu})\right). \end{aligned} \quad (\text{A.98})$$

Proof. We will prove this using Theorem A.12. This theorem allows us to rewrite exponents such that one of them does not depend on \mathbf{x} anymore. Specifically, it tells us that

$$\begin{aligned} & \int_X \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_a)^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}_a)\right) \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \Sigma) d\mathbf{x} \\ &= \int_X \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_a)^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right) d\mathbf{x} \\ &= \int_X \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}_a - \boldsymbol{\mu})^T (\Lambda + \Sigma)^{-1} (\mathbf{x}_a - \boldsymbol{\mu})\right) \\ & \quad \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}')^T (\Lambda^{-1} + \Sigma^{-1}) (\mathbf{x} - \boldsymbol{\mu}')\right) d\mathbf{x}, \end{aligned} \quad (\text{A.99})$$

where we have defined the totally irrelevant parameter $\boldsymbol{\mu}'$ as

$$\boldsymbol{\mu}' = (\Lambda^{-1} + \Sigma^{-1})^{-1} (\Lambda^{-1} \mathbf{x}_a + \Sigma^{-1} \boldsymbol{\mu}). \quad (\text{A.100})$$

In our integral, the first exponent does not depend on \mathbf{x} anymore, so we can pull it out. That leaves us with the second integral. Because the integral over a Gaussian exponential equals one, as explained by (A.86), we know that

$$\begin{aligned} \int_X \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}')^T (\Lambda^{-1} + \Sigma^{-1}) (\mathbf{x} - \boldsymbol{\mu}')\right) d\mathbf{x} &= \sqrt{|2\pi(\Lambda^{-1} + \Sigma^{-1})^{-1}|} \\ &= \sqrt{|2\pi\Sigma(\Lambda + \Sigma)^{-1}\Lambda|} \\ &= \sqrt{\frac{|2\pi\Sigma||\Lambda|}{|\Lambda + \Sigma|}}. \end{aligned} \quad (\text{A.101})$$

By using this result, we directly wind up with (A.98). \square

The next two theorems look at integrals over three Gaussian exponentials.

Theorem A.19. *For known parameters \mathbf{x}_a , \mathbf{x}_b , Λ , $\boldsymbol{\mu}$ and Σ , if we define $\bar{\mathbf{x}} \equiv \frac{1}{2}(\mathbf{x}_a + \mathbf{x}_b)$, then it holds that*

$$\begin{aligned} & \int_X \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_a)^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_b)^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}_b)\right) \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \Sigma) d\mathbf{x} \\ &= \sqrt{\frac{|\Lambda|}{|\Lambda + 2\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}_a - \mathbf{x}_b)^T (2\Lambda)^{-1} (\mathbf{x}_a - \mathbf{x}_b)\right) \exp\left(-\frac{1}{2}(\bar{\mathbf{x}} - \boldsymbol{\mu})^T \left(\frac{1}{2}\Lambda + \Sigma\right)^{-1} (\bar{\mathbf{x}} - \boldsymbol{\mu})\right). \end{aligned} \quad (\text{A.102})$$

Proof. We will prove this using Theorem A.12. This theorem allows us to rewrite the three matrix exponentials that all depend on \mathbf{x} into three matrix exponentials of which only one depends on \mathbf{x} .

First we rewrite the product of the first two exponentials. This equals

$$\begin{aligned} & \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_a)^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_b)^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}_b)\right) \\ &= \exp\left(-\frac{1}{2}(\mathbf{x}_a - \mathbf{x}_b)^T (2\Lambda)^{-1} (\mathbf{x}_a - \mathbf{x}_b)\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \bar{\mathbf{x}})^T (2\Lambda^{-1}) (\mathbf{x} - \bar{\mathbf{x}})\right). \end{aligned} \quad (\text{A.103})$$

A

Note that the matrix inverse in the two above exponents is different. While the first becomes $(2\Lambda)^{-1}$, the second becomes $(2\Lambda^{-1}) = (\frac{1}{2}\Lambda)^{-1}$. The first of the two above exponentials does not depend on \mathbf{x} , so we can pull it out of the integral. We combine the second exponential with $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$, again using Theorem A.12. It becomes

$$\begin{aligned} & \exp\left(-\frac{1}{2}(\mathbf{x} - \bar{\mathbf{x}})^T(2\Lambda^{-1})(\mathbf{x} - \bar{\mathbf{x}})\right) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) \\ &= \exp\left(-\frac{1}{2}(\mathbf{x} - \bar{\mathbf{x}})^T\left(\frac{1}{2}\Lambda\right)^{-1}(\mathbf{x} - \bar{\mathbf{x}})\right) \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \\ &= \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\bar{\mathbf{x}} - \boldsymbol{\mu})^T\left(\frac{1}{2}\Lambda + \Sigma\right)^{-1}(\bar{\mathbf{x}} - \boldsymbol{\mu})\right) \exp\left(-\frac{1}{2}(\mathbf{x} - \bar{\boldsymbol{\mu}})^T\left(\left(\frac{1}{2}\Lambda\right)^{-1} + \Sigma^{-1}\right)(\mathbf{x} - \bar{\boldsymbol{\mu}})\right), \end{aligned} \quad (\text{A.104})$$

where the value of $\bar{\boldsymbol{\mu}}$ is totally irrelevant. Though if you're interested, it equals

$$\bar{\boldsymbol{\mu}} = \left(\left(\frac{1}{2}\Lambda\right)^{-1} + \Sigma^{-1}\right)^{-1} \left(\left(\frac{1}{2}\Lambda\right)^{-1} \bar{\mathbf{x}} + \Sigma^{-1} \boldsymbol{\mu}\right). \quad (\text{A.105})$$

The most important realization here is that only the latter exponential depends on \mathbf{x} , and hence we can evaluate the integral. Using (A.86), we can find that

$$\int_X \exp\left(-\frac{1}{2}(\mathbf{x} - \bar{\boldsymbol{\mu}})^T\left(\left(\frac{1}{2}\Lambda\right)^{-1} + \Sigma^{-1}\right)(\mathbf{x} - \bar{\boldsymbol{\mu}})\right) d\mathbf{x} = \sqrt{\left|2\pi\left(\left(\frac{1}{2}\Lambda\right)^{-1} + \Sigma^{-1}\right)^{-1}\right|}, \quad (\text{A.106})$$

which can be rewritten to

$$\sqrt{\left|2\pi\left(\left(\frac{1}{2}\Lambda\right)^{-1} + \Sigma^{-1}\right)^{-1}\right|} = \sqrt{\left|2\pi\left(\frac{1}{2}\Lambda\right)\left(\frac{1}{2}\Lambda + \Sigma\right)^{-1}\Sigma\right|} = \sqrt{\frac{\left|\frac{1}{2}\Lambda\right| |2\pi\Sigma|}{\left|\frac{1}{2}\Lambda + \Sigma\right|}}. \quad (\text{A.107})$$

Note that, because the matrices Σ and Λ are of equal size, we could have put the term 2π in any determinant in the above expression. Though if we would have pulled 2π out of the determinant, we would have had to write $(2\pi)^{d_x}$, with d_x the dimension of the vector \mathbf{x} .

If we now put all the results that we have obtained together, we directly find (A.102). \square

Theorem A.20. *For known parameters Λ , $\boldsymbol{\mu}_a$, Σ_a , $\boldsymbol{\mu}_b$ and Σ_b , it holds that*

$$\begin{aligned} \int_X \int_X \exp\left(-\frac{1}{2}(\mathbf{x}_a - \mathbf{x}_b)^T\Lambda^{-1}(\mathbf{x}_a - \mathbf{x}_b)\right) \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_a, \Sigma_a) \mathcal{N}(\mathbf{x}_b|\boldsymbol{\mu}_b, \Sigma_b) d\mathbf{x}_b d\mathbf{x}_a \\ = \sqrt{\frac{|\Lambda|}{|\Lambda + \Sigma_a + \Sigma_b|}} \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)^T(\Lambda + \Sigma_a + \Sigma_b)^{-1}(\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)\right). \end{aligned} \quad (\text{A.108})$$

Proof. We will once more prove this with the help of Theorem A.12. If we expand the Gaussian exponential terms in the double integral, we can write the double integral as

$$\begin{aligned} \int_X \int_X \frac{1}{\sqrt{|2\pi\Sigma_a||2\pi\Sigma_b|}} \exp\left(-\frac{1}{2}(\mathbf{x}_a - \mathbf{x}_b)^T\Lambda^{-1}(\mathbf{x}_a - \mathbf{x}_b)\right) \\ \exp\left(-\frac{1}{2}(\mathbf{x}_a - \boldsymbol{\mu}_a)^T\Sigma_a^{-1}(\mathbf{x}_a - \boldsymbol{\mu}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x}_b - \boldsymbol{\mu}_b)^T\Sigma_b^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)\right) d\mathbf{x}_b d\mathbf{x}_a. \end{aligned} \quad (\text{A.109})$$

There are now two exponentials containing \mathbf{x}_a and two exponentials containing \mathbf{x}_b . To start, we want only one exponential to contain \mathbf{x}_b . To accomplish this, we will merge the first and third exponential through Theorem A.12. This results in

$$\begin{aligned} & \exp\left(-\frac{1}{2}(\mathbf{x}_b - \mathbf{x}_a)^T \Lambda^{-1} (\mathbf{x}_b - \mathbf{x}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x}_b - \boldsymbol{\mu}_b)^T \Sigma_b^{-1} (\mathbf{x}_b - \boldsymbol{\mu}_b)\right) \\ &= \exp\left(-\frac{1}{2}(\mathbf{x}_a - \boldsymbol{\mu}_b)^T (\Lambda + \Sigma_b)^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_b)\right) \exp\left(-\frac{1}{2}(\mathbf{x}_b - \boldsymbol{\mu})^T (\Lambda^{-1} + \Sigma_b^{-1}) (\mathbf{x}_b - \boldsymbol{\mu})\right), \end{aligned} \quad (\text{A.110})$$

where we have defined the irrelevant parameter $\boldsymbol{\mu}$ as $(\Lambda^{-1} + \Sigma_b^{-1})^{-1} (\Lambda^{-1} \mathbf{x}_a + \Sigma_b^{-1} \boldsymbol{\mu}_b)$. Note that only the last exponential now depends on \mathbf{x}_b , which means we can integrate it over \mathbf{x}_b . We find through (A.86) that

$$\begin{aligned} \int_X \exp\left(-\frac{1}{2}(\mathbf{x}_b - \boldsymbol{\mu})^T (\Lambda^{-1} + \Sigma_b^{-1}) (\mathbf{x}_b - \boldsymbol{\mu})\right) d\mathbf{x}_b &= \sqrt{|2\pi(\Lambda^{-1} + \Sigma_b^{-1})^{-1}|} \\ &= \sqrt{\frac{|2\pi\Sigma_b||\Lambda|}{|\Lambda + \Sigma_b|}}. \end{aligned} \quad (\text{A.111})$$

We are left with the second exponential from (A.109) and the first exponential from the result of (A.110). If we merge these together too, we find

$$\begin{aligned} & \exp\left(-\frac{1}{2}(\mathbf{x}_a - \boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_a)\right) \exp\left(-\frac{1}{2}(\mathbf{x}_a - \boldsymbol{\mu}_b)^T (\Lambda + \Sigma_b)^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_b)\right) \\ &= \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)^T (\Lambda + \Sigma_a + \Sigma_b)^{-1} (\boldsymbol{\mu}_a - \boldsymbol{\mu}_b)\right) \\ & \quad \exp\left(-\frac{1}{2}(\mathbf{x}_a - \boldsymbol{\mu}')^T ((\Lambda + \Sigma_b)^{-1} + \Sigma_a^{-1}) (\mathbf{x}_a - \boldsymbol{\mu}')\right), \end{aligned} \quad (\text{A.112})$$

where $\boldsymbol{\mu}'$ is again a totally irrelevant parameter which I will not even bother defining this time. The more important part is that the last exponential from the above expression is the only one depending on \mathbf{x}_a , so we can integrate over \mathbf{x}_a . That will turn this exponential into

$$\int_X \exp\left(-\frac{1}{2}(\mathbf{x}_a - \boldsymbol{\mu}')^T ((\Lambda + \Sigma_b)^{-1} + \Sigma_a^{-1}) (\mathbf{x}_a - \boldsymbol{\mu}')\right) d\mathbf{x}_a = \sqrt{\frac{|2\pi\Sigma_a||\Lambda + \Sigma_b|}{|\Lambda + \Sigma_a + \Sigma_b|}}. \quad (\text{A.113})$$

If we now put all our results together, then we directly find (A.108). \square

A.4. LYAPUNOV EQUATIONS

A *Lyapunov equation* is a matrix equation of the form $AX + XA^T + Q = 0$. Its solution X can be found analytically (through a process similar to matrix sweeping) and this solution can be useful for a variety of applications. The solutions to Lyapunov equations also have interesting properties. In this section we look at various such properties.

There is a lot of literature available on solving Lyapunov equations. Methods to find when a Lyapunov equation has a unique solution, as well as methods to find this unique solution, are mentioned in a variety of publications, among which those by Bartels and

A

[Stewart \(1972\)](#), [Antoulas \(2005\)](#). On the flip side, there is no literature available, as far as I know, that specifically looks at the properties of the resulting solutions, apart from my own publication [Bijl et al. \(2016\)](#). So that is what this chapter is mainly about.

We start this section off by looking at notations and definitions (Section A.4.1). Then we examine when a Lyapunov equation has a unique solution and how to find it (Section A.4.2). Afterwards, we look at some basic properties of Lyapunov solutions (Section A.4.3) as well as how sums and products of Lyapunov solutions work (Section A.4.4). Finally we challenge ourselves with a few difficult integrals which can also be solved using Lyapunov solutions (Section A.4.5).

A.4.1. NOTATIONS AND DEFINITIONS

Before we start, we should get our notations and definitions in order. These notations and definitions may seem haphazard at first, but we will use every one of them.

We start with matrix terminology. Consider a square matrix A with its eigenvalues written as $\lambda_1, \dots, \lambda_n$. This matrix may satisfy certain properties.

- A is called *stable* (or *Hurwitz*) if and only if it has no eigenvalue λ_i with a real part equal to or larger than zero.
- A is called *Sylvester* if and only if it has no two eigenvalues λ_i and λ_j (with possibly $i = j$) satisfying $\lambda_i = -\lambda_j$.
- A is called *invertible* if and only if it has no eigenvalue λ_i equal to zero.

While the first and third concepts are well-known in literature, the second one is new, first posed by [Bijl et al. \(2016\)](#). It will prove crucial when dealing with Lyapunov solutions though, as you will see at Theorem A.23. Also note that a stable matrix is always a Sylvester matrix, and a Sylvester matrix is always invertible, but the converse does not always hold.

Next, let's define some notation conventions. We define the *Lyapunov solution*¹ X^Q to be the solution of the Lyapunov equation

$$AX^Q + X^Q A^T + Q \equiv 0, \quad (\text{A.114})$$

where A is known as the *multiplication matrix* and Q as the *Lyapunov constant*. They both have to be square matrices of the appropriate size. So in our notation X^Q we explicitly mention which Lyapunov constant we have used. The fact that the multiplication matrix A was used is assumed default and is hence not included in the notation.

In a similar way do we define the *alternate Lyapunov solution* \bar{X}^Q as the solution of the *alternate Lyapunov equation*

$$A^T \bar{X}^Q + \bar{X}^Q A + Q \equiv 0. \quad (\text{A.115})$$

Note that we have transposed A here.

¹In this thesis we only work with the *continuous Lyapunov equation* $AX + XA^T + Q = 0$. We do not consider the *discrete Lyapunov equation* $AXA^T - X + Q = 0$. Additionally, we do not consider complex-valued matrices. As a result, we can take the regular transpose A^T of matrices instead of the conjugate transpose A^H .

Instead of A , we may sometimes also use a different matrix B . If we are not using A within the Lyapunov equation, we specifically mention we are using B through a subscript. We hence have

$$BX_B^Q + X_B^Q B^T + Q \equiv 0. \quad (\text{A.116})$$

We sometimes also use a matrix A_α , which is defined as $A_\alpha \equiv A + \alpha I$ for some scalar α . Similarly, we define $A_{k\alpha} \equiv A + k\alpha I$, where k is often an integer. We can write the corresponding Lyapunov solution as $X_{A_{k\alpha}}^Q$. However, we often shorten this notation to $X_{k\alpha}^Q$. So we per definition have

$$A_{k\alpha} X_{k\alpha}^Q + X_{k\alpha}^Q A_{k\alpha}^T + Q \equiv 0 \quad (\text{A.117})$$

and similarly for $\bar{X}_{k\alpha}^Q$. Note that $A_0 = A$ and as a result also $X_0^Q = X^Q$. Also note that, though we have defined $A_\alpha = A + \alpha I$, we do *not* have $X_\alpha = X + \alpha I$. The notation only holds for A .

Lyapunov solutions may also stack. As a result, X^{X^Q} is the solution of the equation

$$AX^{X^Q} + X^{X^Q} A^T + X^Q \equiv 0. \quad (\text{A.118})$$

So effectively X^{X^Q} equals $X^{(X^Q)}$.

Next, let's look at a few integrals. We define the time-dependent parameters $X_{k\alpha}^Q(t_1, t_2)$ and $\bar{X}_{k\alpha}^Q(t_1, t_2)$ as the respective integrals

$$X_{k\alpha}^Q(t_1, t_2) \equiv \int_{t_1}^{t_2} e^{A_{k\alpha} t} Q e^{A_{k\alpha}^T t} dt, \quad (\text{A.119})$$

$$\bar{X}_{k\alpha}^Q(t_1, t_2) \equiv \int_{t_1}^{t_2} e^{A_{k\alpha}^T t} Q e^{A_{k\alpha} t} dt. \quad (\text{A.120})$$

This notation may seem strange at first, but it will make sense later on at Theorem A.26. Additionally, we often have $t_1 = 0$. In that case we define the shorter notation $X_{k\alpha}^Q(t) \equiv X_{k\alpha}^Q(0, t)$.

Finally, we define $\tilde{X}_{k\alpha}^Q(t)$ as the integral

$$\tilde{X}_{k\alpha}^Q(t) \equiv \int_0^t e^{A_{k\alpha}(t-s)} Q e^{As} ds = e^{-n\alpha t} \int_0^t e^{A_{(k+n)\alpha}(t-s)} Q e^{A_{n\alpha}s} ds, \quad (\text{A.121})$$

where the last part holds for any n . Again we write $\tilde{X}_0^Q(t)$ as $\tilde{X}^Q(t)$, omitting any zero subscript.

A.4.2. FINDING THE LYAPUNOV SOLUTION

Before we look at how to solve the Lyapunov equation, we first look at a more general case: how to solve the *Sylvester equation*

$$AX + XB = C. \quad (\text{A.122})$$

In particular, we start by looking at when this equation has a unique solution.

A

Theorem A.21. *The Sylvester equation (A.122) has a unique solution if and only if A and $-B$ do not have any common eigenvalues.*

Proof. The proof of this can be found in the work of [Bartels and Stewart \(1972\)](#), [Antoulas \(2005\)](#). It requires a few relatively advanced mathematical techniques, which I do not want to study in-depth here, so you either have to trust me or look up the references. \square

The next question is, ‘If there is a unique solution, how do we find it?’

Theorem A.22. *If the Sylvester equation (A.122) has a unique solution X , then this solution can be found through*

$$\text{vec}(X) = (I \otimes A + B^T \otimes I)^{-1} \text{vec}(C). \quad (\text{A.123})$$

Proof. Using Theorem A.4 we can rewrite the Sylvester equation to

$$(I \otimes A + B^T \otimes I) \text{vec}(X) = \text{vec}(C). \quad (\text{A.124})$$

We have assumed that there is a unique solution for X . This is only the case when the above matrix $(I \otimes A + B^T \otimes I)$ is invertible. As such, left-multiplying the above by the inverse of this matrix results in (A.123). \square

Now we know how to solve the Sylvester equation, we can use our results to solve the Lyapunov equation.

Theorem A.23. *The Lyapunov equation (A.114) has a unique solution X^Q if and only if the matrix A is Sylvester. In this case the solution can be found through*

$$\text{vec}(X^Q) = (I \otimes A + A \otimes I)^{-1} \text{vec}(Q). \quad (\text{A.125})$$

Proof. To prove the first claim, we will apply Theorem A.21 with $B = A^T$. It follows that the Lyapunov equation has a unique solution if and only if A and $-A^T$ do not have any common eigenvalues.

Let’s denote the eigenvalues of A by $\lambda_1, \dots, \lambda_n$. The eigenvalues of $-A^T$ now equal $-\lambda_1, \dots, -\lambda_n$. After all, transposing a (square) matrix does not alter its eigenvalues, and if λ is an eigenvalue of A , then $-\lambda$ is an eigenvalue of $-A$. We can hence see that A and $-A^T$ have a common eigenvalue if and only if A has two eigenvalues λ_i and λ_j (with possibly $i = j$) which satisfy $\lambda_i = -\lambda_j$. This is (per definition) the case if and only if A is Sylvester. So we can conclude that the Lyapunov equation has a unique solution if and only if A is Sylvester.

That proves the first claim of the theorem. To prove (A.125) we can apply Theorem A.22 with again $B = A^T$. This directly completes the proof. \square

A.4.3. BASIC PROPERTIES OF LYAPUNOV SOLUTIONS

Lyapunov solutions have many interesting properties. The following theorems will outline the most important ones. They all concern the Lyapunov solutions X^Q and not the alternate Lyapunov solutions \bar{X}^Q . However, all theorems also hold for the alternate Lyapunov solutions if we replace A by A^T .

We start off by looking at when Lyapunov solutions are symmetric.

Theorem A.24. Assume that A is Sylvester. The Lyapunov solution X^Q is symmetric if and only if Q is symmetric.

Proof. We know from Theorem A.23 that X^Q is the unique solution of the Lyapunov equation

$$AX^Q + X^Q A^T + Q = 0. \quad (\text{A.126})$$

If we transpose this equation, then we get

$$A(X^Q)^T + (X^Q)^T A^T + Q^T = 0. \quad (\text{A.127})$$

Subtracting the two equations results in

$$A(X^Q - (X^Q)^T) + (X^Q - (X^Q)^T) A^T + (Q - Q^T) = 0. \quad (\text{A.128})$$

This equation is actually a new Lyapunov equation. Because the multiplication matrix A is Sylvester, there is a unique solution for $(X^Q - (X^Q)^T)$ (Theorem A.23). If Q is symmetric, then $Q - Q^T = 0$ and this unique solution must equal $X^Q - (X^Q)^T = 0$, implying that X^Q is symmetric. Similarly, if X^Q is symmetric, then $X^Q - (X^Q)^T = 0$ and we must also have $Q - Q^T = 0$, meaning that Q is symmetric. This proves both sides of the theorem. \square

We know that if A is Sylvester, then there is a unique solution for X^Q . But what happens when A is stable?

Theorem A.25. Assume that A is stable. Then the unique solution of the Lyapunov equation (A.114) equals the infinite integral

$$X^Q = \int_0^\infty e^{At} Q e^{A^T t} dt. \quad (\text{A.129})$$

Proof. Note that, because A is stable, it is also Sylvester, proving that there is a unique solution X^Q of the Lyapunov equation (Theorem A.23).

Next, we prove that X^Q equals the infinite integral. Because A is stable, we know that

$$\lim_{t \rightarrow \infty} e^{At} = 0. \quad (\text{A.130})$$

As a result, we can write Q as

$$\begin{aligned} Q &= - \left[e^{At} Q e^{A^T t} \right]_0^\infty \\ &= - \int_0^\infty \frac{d}{dt} (e^{At} Q e^{A^T t}) dt \\ &= - \int_0^\infty (A e^{At} Q e^{A^T t} + e^{At} Q e^{A^T t} A^T) dt \\ &= - A \left(\int_0^\infty e^{At} Q e^{A^T t} dt \right) - \left(\int_0^\infty e^{At} Q e^{A^T t} dt \right) A^T. \end{aligned} \quad (\text{A.131})$$

We see that the equation above is a Lyapunov equation, with the quantity within brackets as the solution. Because the solution exists, is unique and equals X^Q , the quantity within brackets must equal X^Q . So we see that (A.129) indeed holds. \square

A

The nice part about the above theorem is that, if we want to find the infinite integral (A.129), we only have to solve a Lyapunov equation and not numerically simulate an infinite integral. That will save a lot of computations.

Basically, the above theorem says that $X^Q(0, \infty) = X^Q$ when A is stable. You should be careful with using this relation, because it does not hold when A is not stable. After all, in this case the integral from (A.129) will not have a finite value, even though there will still be a unique finite solution for X^Q . For example, consider $A = 1$ and $Q = 2$. Now $X^Q = -1$, but the integral is infinite.

To prevent the integral from becoming infinitely large, we can give it finite bounds. It is now per definition A.119 equal to $X^Q(t_1, t_2)$. This definition also holds for non-Sylvester matrices A , but when A is Sylvester we can calculate $X^Q(t_1, t_2)$ using the following Theorem.

Theorem A.26. *Assume that A is Sylvester. In this case $X^Q(t_1, t_2)$, defined through (A.119), can either be found by solving the Lyapunov equation*

$$AX^Q(t_1, t_2) + X^Q(t_1, t_2)A^T + e^{At_1}Qe^{A^T t_1} - e^{At_2}Qe^{A^T t_2} = 0. \quad (\text{A.132})$$

or by first finding X^Q and then using

$$X^Q(t_1, t_2) = e^{At_1}X^Qe^{A^T t_1} - e^{At_2}X^Qe^{A^T t_2}. \quad (\text{A.133})$$

Proof. This theorem consists of two parts. To prove the first part, we consider the quantity $e^{At_1}Qe^{A^T t_1} - e^{At_2}Qe^{A^T t_2}$. It equals

$$\begin{aligned} e^{At_1}Qe^{A^T t_1} - e^{At_2}Qe^{A^T t_2} &= - \left[e^{At}Qe^{A^T t} \right]_{t_1}^{t_2} \\ &= - \int_{t_1}^{t_2} \frac{d}{dt} \left(e^{At}Qe^{A^T t} \right) dt \\ &= - A \left(\int_{t_1}^{t_2} e^{At}Qe^{A^T t} dt \right) - \left(\int_{t_1}^{t_2} e^{At}Qe^{A^T t} dt \right) A^T \\ &= - AX^Q(t_1, t_2) - X^Q(t_1, t_2)A^T. \end{aligned} \quad (\text{A.134})$$

This shows that $X^Q(t_1, t_2)$ indeed satisfies Lyapunov equation (A.132), proving the first part of the theorem.

To prove the second part too, we make use of the expression $AX^Q + X^Q A^T + Q = 0$ and of the matrix property $e^{At}A = Ae^{At}$. This allows us to find

$$\begin{aligned} e^{At_1}Qe^{A^T t_1} - e^{At_2}Qe^{A^T t_2} &= -e^{At_1}(AX^Q + X^Q A^T)e^{A^T t_1} + e^{At_2}(AX^Q + X^Q A^T)e^{A^T t_2} \\ &= -e^{At_1}AX^Qe^{A^T t_1} - e^{At_1}X^QA^Te^{A^T t_1} + e^{At_2}AX^Qe^{A^T t_2} + e^{At_2}X^QA^Te^{A^T t_2} \\ &= -Ae^{At_1}X^Qe^{A^T t_1} - e^{At_1}X^Qe^{A^T t_1}A^T + Ae^{At_2}X^Qe^{A^T t_2} + e^{At_2}X^Qe^{A^T t_2}A^T \\ &= -A(e^{At_1}X^Qe^{A^T t_1} - e^{At_2}X^Qe^{A^T t_2}) - (e^{At_1}X^Qe^{A^T t_1} - e^{At_2}X^Qe^{A^T t_2})A^T. \end{aligned} \quad (\text{A.135})$$

The above expression is actually Lyapunov equation (A.132), in which the part between brackets is replaced by $X^Q(t_1, t_2)$. Because A is Sylvester, the Lyapunov equation has a unique solution, and hence (A.133) must hold. This also proves the second part of the theorem. \square

It is interesting to note that, for stable matrices A and for $t_1 = 0$ and $t_2 \rightarrow \infty$, the above Theorem A.26 directly implies Theorem A.25.

By the way, you might have noticed something odd about our definitions of X^Q and $X^Q(t_1, t_2)$. For Sylvester matrices A , we have defined X^Q as the solution to a Lyapunov equation, which in special cases (A being stable) also happened to equal an integral over matrix exponentials. However, for any matrix A we have defined $X^Q(t_1, t_2)$ as an integral over matrix exponentials, which in special cases (A being Sylvester) also happened to equal the solution to a Lyapunov equation. By setting up the definitions in this way, we have made them as broadly applicable as possible.

A.4.4. COMBINATIONS OF LYAPUNOV SOLUTIONS

Suppose we have two Lyapunov solutions X^Q and X^V . Is their sum then also a Lyapunov solution? And what about linear products of Lyapunov solutions?

Theorem A.27. *Assume that A is Sylvester. For any Q and V we then have*

$$X^{Q+V} = X^Q + X^V. \quad (\text{A.136})$$

Proof. To prove this claim, we note that we per definition have

$$AX^Q + X^Q A^T + Q = 0, \quad (\text{A.137})$$

$$AX^V + X^V A^T + V = 0. \quad (\text{A.138})$$

If we add up these expressions, we find that

$$A(X^Q + X^V) + (X^Q + X^V)A^T + Q + V = 0. \quad (\text{A.139})$$

This is a Lyapunov expression with respect to the part between brackets. Because it has a unique solution (Theorem A.23), the part between brackets must equal the solution X^{Q+V} of the Lyapunov equation and hence we have $X^{Q+V} = X^Q + X^V$. \square

Theorem A.28. *Assume that A is Sylvester. If the matrices A and C commute (that is, $CA = AC$) then*

$$X^{CQ} = CX^Q. \quad (\text{A.140})$$

Proof. To prove this, we left-multiply the Lyapunov equation by C . This gives us

$$CAX^Q + CX^Q A^T + CQ = 0. \quad (\text{A.141})$$

If we now apply the assumption that $CA = AC$, this becomes

$$A(CX^Q) + (CX^Q)A^T + CQ = 0. \quad (\text{A.142})$$

This is a Lyapunov expression again. As a result, the part between brackets equals $X^{CQ} = CX^Q$. \square

When a Lyapunov solution is inside a trace function, it is sometimes possible to transform one Lyapunov solution into another.

A

Theorem A.29. Assume that A is Sylvester. For matrices F and G satisfying $AF = FA$ and $A^T G = GA^T$, and for any Q and V , we have

$$\text{tr}(QFX^V G) = \text{tr}(\bar{X}^Q FVG). \quad (\text{A.143})$$

Proof. We can prove the above by rewriting one into the other. That is,

$$\begin{aligned} \text{tr}(QFX^V G) &= \text{tr}((-A^T \bar{X}^Q - \bar{X}^Q A)FX^V G) \\ &= \text{tr}((-A^T \bar{X}^Q FX^V G - \bar{X}^Q AFX^V G)) \\ &= \text{tr}((-G\bar{X}^Q FX^V A^T - G\bar{X}^Q FAX^V)) \\ &= \text{tr}(G\bar{X}^Q F(-X^V A^T - AX^V)) \\ &= \text{tr}(\bar{X}^Q FVG). \end{aligned} \quad (\text{A.144})$$

This proves the statement. Do note that we have used the cyclic property $\text{tr}(ABC) = \text{tr}(CAB) = \text{tr}(BCA)$ of the trace function at various points in the above derivation. The theorem does not hold without the trace function. \square

When applying the above theorem, typical values for F are I , e^{At} and $e^{A_\alpha t}$, while often $G = F^T$.

Next to turning one Lyapunov solution into another, we can sometimes also turn a difference in Lyapunov solutions into a Lyapunov solution.

Theorem A.30. Assume that both A and A_α are Sylvester. The Lyapunov solutions X^Q , X_α^Q , $X_\alpha^{X^Q}$ and $X^{X_\alpha^Q}$ now satisfy

$$X_\alpha^{X^Q} = X^{X_\alpha^Q} = \frac{X_\alpha^Q - X^Q}{2\alpha}. \quad (\text{A.145})$$

Proof. Per definition we have

$$(A + \alpha I)X_\alpha^Q + X_\alpha^Q(A + \alpha I)^T + Q = 0, \quad (\text{A.146})$$

$$AX^Q + X^Q A^T + Q = 0. \quad (\text{A.147})$$

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

$$A(X_\alpha^Q - X^Q) + (X_\alpha^Q - X^Q)A^T + 2\alpha X_\alpha^Q = 0, \quad (\text{A.148})$$

$$A_\alpha(X_\alpha^Q - X^Q) + (X_\alpha^Q - X^Q)A_\alpha^T + 2\alpha X^Q = 0. \quad (\text{A.149})$$

Next, we divide the above equations by 2α , resulting in

$$A\left(\frac{X_\alpha^Q - X^Q}{2\alpha}\right) + \left(\frac{X_\alpha^Q - X^Q}{2\alpha}\right)A^T + X_\alpha^Q = 0, \quad (\text{A.150})$$

$$A_\alpha\left(\frac{X_\alpha^Q - X^Q}{2\alpha}\right) + \left(\frac{X_\alpha^Q - X^Q}{2\alpha}\right)A_\alpha^T + X^Q = 0. \quad (\text{A.151})$$

These are Lyapunov equations with respect to the term in brackets. Because A and A_α are Sylvester, they must have a unique solution, which equals

$$X^{X_\alpha^Q} = \frac{X_\alpha^Q - X^Q}{2\alpha} = X_\alpha^{X^Q}. \quad (\text{A.152})$$

This completes the proof. \square

A.4.5. MORE INTEGRAL EXPRESSIONS

We know from Theorem A.25 that, for stable A , we can also write X^Q as an infinite integral. Next, we will add an extra time factor t inside this integral. The following theorem tells us what this will do with the outcome.

Theorem A.31. *Assume that A is stable. Now we have*

$$X^{X^Q} = \int_0^\infty t e^{At} X^Q e^{A^T t} dt. \quad (\text{A.153})$$

Proof. According to definition (A.118) and Theorem A.25 we can write

$$X^{X^Q} = \int_0^\infty e^{At} X^Q e^{A^T t} dt = \int_0^\infty e^{At} \left(\int_0^\infty e^{A\tau} Q e^{A^T \tau} d\tau \right) e^{A^T t} dt. \quad (\text{A.154})$$

We can pull the integral signs outside, merging the integrands. This gives us

$$X^{X^Q} = \int_0^\infty \int_0^\infty e^{A(t+\tau)} Q e^{A^T(t+\tau)} d\tau dt. \quad (\text{A.155})$$

We can substitute τ for $s - t$. This turns $d\tau$ into ds , and because τ ranged from 0 to ∞ , we have s ranging from t to ∞ . This turns the above equation into

$$X^{X^Q} = \int_0^\infty \int_t^\infty e^{As} Q e^{A^T s} ds dt. \quad (\text{A.156})$$

Next, we interchange the integrals. When doing so, we should keep the integration area the same. The integration area, when plotted in the s - t -plane, is the triangle bound by $0 \leq t \leq s$. When taking into account this inequality, we can rewrite the above to

$$X^{X^Q} = \int_0^\infty \int_0^s e^{As} Q e^{A^T s} dt ds. \quad (\text{A.157})$$

The inner integral is now with respect to t , but there is no t in the integrand. As a result, we can write

$$\begin{aligned} X^{X^Q} &= \int_0^\infty \left(\int_0^s 1 dt \right) e^{As} Q e^{A^T s} ds \\ &= \int_0^\infty s e^{As} Q e^{A^T s} ds. \end{aligned} \quad (\text{A.158})$$

This equals what we wanted to prove. \square

A

Let's challenge ourselves a bit more. Let's consider what happens when we integrate over four matrix exponents.

Theorem A.32. *Assume that A is stable, and that P_1 , P_2 and Q are symmetric. We then have*

$$\operatorname{tr}\left(\int_0^\infty P_1 e^{At} Q e^{A^T t} P_2 e^{At} X^Q e^{A^T t} dt\right) = \frac{1}{2} \operatorname{tr}(P_1 X^Q P_2 X^Q). \quad (\text{A.159})$$

Proof. We will start with the right hand side of the expression. It equals

$$\begin{aligned} \frac{1}{2} \operatorname{tr}(P_1 X^Q P_2 X^Q) &= \frac{1}{2} \operatorname{tr}\left(P_1 \left(\int_0^\infty e^{At} Q e^{A^T t} dt\right) P_2 \left(\int_0^\infty e^{As} Q e^{A^T s} ds\right)\right) \\ &= \frac{1}{2} \operatorname{tr}\left(\int_0^\infty \int_0^\infty P_1 e^{At} Q e^{A^T t} P_2 e^{As} Q e^{A^T s} ds dt\right). \end{aligned} \quad (\text{A.160})$$

The above integrand is symmetric in t and s . That is, if we interchange t and s , it has exactly the same value. To see why, transpose whatever is in the trace function. This is allowed, since the trace of a matrix is equal to the trace of its transpose. After transposing, you will find exactly the same expression.

Because the integrand is symmetric in t and s , we don't have to integrate over all values of $t \geq 0$ and $s \geq 0$. If we integrate over $t \geq 0$ and $s \geq t$, then we get exactly half of what we otherwise would have gotten. Hence, we can rewrite the above to

$$\frac{1}{2} \operatorname{tr}(P_1 X^Q P_2 X^Q) = \operatorname{tr}\left(\int_0^\infty \int_t^\infty P_1 e^{At} Q e^{A^T t} P_2 e^{As} Q e^{A^T s} ds dt\right). \quad (\text{A.161})$$

If we work this out further, substituting s by $\tau + t$, we get

$$\begin{aligned} \frac{1}{2} \operatorname{tr}(P_1 X^Q P_2 X^Q) &= \operatorname{tr}\left(\int_0^\infty P_1 e^{At} Q e^{A^T t} P_2 \left(\int_t^\infty e^{As} Q e^{A^T s} ds\right) dt\right) \\ &= \operatorname{tr}\left(\int_0^\infty P_1 e^{At} Q e^{A^T t} P_2 \left(e^{At} \int_0^\infty e^{A\tau} Q e^{A^T \tau} d\tau e^{A^T t}\right) dt\right) \\ &= \operatorname{tr}\left(\int_0^\infty P_1 e^{At} Q e^{A^T t} P_2 e^{At} X^Q e^{A^T t} dt\right). \end{aligned} \quad (\text{A.162})$$

And this was what we wanted to prove. □

We have a similar theorem in case the integral does not run to infinity, but to a finite time T .

Theorem A.33. *Assume that A is Sylvester, and that P_1 , P_2 and Q are symmetric. Then we have*

$$\begin{aligned} \operatorname{tr}\left(\int_0^T P_1 e^{At} Q e^{A^T t} P_2 e^{At} X^Q e^{A^T t} dt\right) &= \frac{1}{2} \operatorname{tr}(P_1 X^Q P_2 X^Q) \\ &\quad - \frac{1}{2} \operatorname{tr}\left(P_1 e^{AT} X^Q e^{A^T T} P_2 e^{AT} X^Q e^{A^T T}\right). \end{aligned} \quad (\text{A.163})$$

Proof. We cannot use the results of Theorem A.32, because now we have assumed A to be Sylvester instead of stable. Instead, we will make repeated use of theorem A.26. First, we will use

$$X^Q = \int_0^T e^{As} Q e^{A^T s} ds + e^{AT} X^Q e^{A^T T}. \quad (\text{A.164})$$

If we apply this to the term $\text{tr}(P_1 X^Q P_2 X^Q)$ and expand the brackets, then we find that

$$\begin{aligned} & \frac{1}{2} \text{tr}(P_1 X^Q P_2 X^Q) \\ &= \frac{1}{2} \text{tr}\left(P_1 \left(\int_0^T e^{As} Q e^{A^T s} ds + e^{AT} X^Q e^{A^T T}\right) P_2 \left(\int_0^T e^{As} Q e^{A^T s} ds + e^{AT} X^Q e^{A^T T}\right)\right) \\ &= \frac{1}{2} \text{tr}\left(\int_0^T \int_0^T P_1 e^{As_1} Q e^{A^T s_1} P_2 e^{As_2} Q e^{A^T s_2} ds_1 ds_2\right) + \frac{1}{2} \text{tr}\left(\int_0^T P_1 e^{As} Q e^{A^T s} P_2 e^{AT} X^Q e^{A^T T} ds\right) \\ &\quad + \frac{1}{2} \text{tr}\left(\int_0^T P_1 e^{AT} X^Q e^{A^T T} P_2 e^{As} Q e^{A^T s} ds\right) + \frac{1}{2} \text{tr}\left(P_1 e^{AT} X^Q e^{A^T T} P_2 e^{AT} X^Q e^{A^T T}\right). \end{aligned} \quad (\text{A.165})$$

In the result, the second and third term are in fact equal. Whatever is in the trace function is simply transposed. So we can merge them into one term and get rid of the factor $\frac{1}{2}$. In addition the integrand of the first term is symmetric in s_1 and s_2 , so we can get rid of the factor $\frac{1}{2}$ by simply integrating over half of the integration area. The result will be

$$\begin{aligned} & \frac{1}{2} \text{tr}(P_1 X^Q P_2 X^Q) = \text{tr}\left(\int_0^T \int_{s_2}^T P_1 e^{As_1} Q e^{A^T s_1} P_2 e^{As_2} Q e^{A^T s_2} ds_1 ds_2\right) \\ &\quad + \text{tr}\left(\int_0^T P_1 e^{AT} X^Q e^{A^T T} P_2 e^{As} Q e^{A^T s} ds\right) + \frac{1}{2} \text{tr}\left(P_1 e^{AT} X^Q e^{A^T T} P_2 e^{AT} X^Q e^{A^T T}\right). \end{aligned} \quad (\text{A.166})$$

Theorem A.26 now also tells us that

$$\int_{s_2}^T e^{As_1} Q e^{A^T s_1} ds_1 = e^{As_2} X^Q e^{A^T s_2} - e^{AT} X^Q e^{A^T T}. \quad (\text{A.167})$$

If we apply this, we find that

$$\begin{aligned} & \frac{1}{2} \text{tr}(P_1 X^Q P_2 X^Q) = \text{tr}\left(\int_0^T P_1 \left(e^{As_2} X^Q e^{A^T s_2} - e^{AT} X^Q e^{A^T T}\right) P_2 e^{As_2} Q e^{A^T s_2} ds_2\right) \\ &\quad + \text{tr}\left(\int_0^T P_1 e^{AT} X^Q e^{A^T T} P_2 e^{As} Q e^{A^T s} ds\right) + \frac{1}{2} \text{tr}\left(P_1 e^{AT} X^Q e^{A^T T} P_2 e^{AT} X^Q e^{A^T T}\right). \end{aligned} \quad (\text{A.168})$$

Expanding the brackets will result in some terms cancelling out. We then wind up with

$$\begin{aligned} & \frac{1}{2} \text{tr}(P_1 X^Q P_2 X^Q) = \text{tr}\left(\int_0^T P_1 e^{As} Q e^{A^T s} P_2 e^{As} X^Q e^{A^T s} ds\right) \\ &\quad + \frac{1}{2} \text{tr}\left(P_1 e^{AT} X^Q e^{A^T T} P_2 e^{AT} X^Q e^{A^T T}\right), \end{aligned} \quad (\text{A.169})$$

and this is the equation which we needed to prove. \square

It is interesting to note that Theorem A.33 does directly turn into Theorem A.32 when A is stable and $T \rightarrow \infty$.

A

A.5. USING MATRIX EXPONENTIALS TO SOLVE INTEGRALS

We just saw that we can use Lyapunov solutions to solve certain matrix integrals. It is also possible to solve such integrals using matrix exponentials, effectively resulting in another way of finding Lyapunov solutions.

This interesting method was first outlined by van Loan (1978). We will repeat the theorems from this reference, though discussed in a more step by step fashion (Section A.5.1). Then we apply these theorems to solve Lyapunov equations (Section A.5.2). Finally we compare this new method to our previous method of finding Lyapunov solutions and set up a small experiment to find out which one works best (Section A.5.3).

A.5.1. INTEGRALS WITHIN MATRIX EXPONENTIALS

It turns out that within matrix exponentials there are also matrix integrals. How this works is explained by the upcoming theorem.

Theorem A.34. *If we define*

$$C = \begin{bmatrix} A_1 & B_1 \\ 0 & A_2 \end{bmatrix}, \quad (\text{A.170})$$

and write e^{Ct} as

$$e^{Ct} \equiv \begin{bmatrix} C_{11}^e(t) & C_{12}^e(t) \\ C_{21}^e(t) & C_{22}^e(t) \end{bmatrix}, \quad (\text{A.171})$$

then we have

$$C_{21}^e(t) = 0, \quad (\text{A.172})$$

$$C_{22}^e(t) = e^{A_2 t}, \quad (\text{A.173})$$

$$C_{11}^e(t) = e^{A_1 t}, \quad (\text{A.174})$$

$$C_{12}^e(t) = \int_0^t e^{A_1(t-s)} B_1 e^{A_2 s} ds. \quad (\text{A.175})$$

Proof. The key to proving this is to use the relation $\frac{d}{dt} e^{Ct} = C e^{Ct}$. That is

$$\begin{bmatrix} \dot{C}_{11}^e(t) & \dot{C}_{12}^e(t) \\ \dot{C}_{21}^e(t) & \dot{C}_{22}^e(t) \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} C_{11}^e(t) & C_{12}^e(t) \\ C_{21}^e(t) & C_{22}^e(t) \end{bmatrix}. \quad (\text{A.176})$$

We can now use the above relation to find several matrix differential equations. Starting with the bottom row and working upward, we have

$$\dot{C}_{21}^e(t) = A_2 C_{21}^e(t), \quad (\text{A.177})$$

$$\dot{C}_{22}^e(t) = A_2 C_{22}^e(t), \quad (\text{A.178})$$

$$\dot{C}_{11}^e(t) = A_1 C_{11}^e(t) + B_1 C_{21}^e(t), \quad (\text{A.179})$$

$$\dot{C}_{12}^e(t) = A_1 C_{12}^e(t) + B_1 C_{22}^e(t). \quad (\text{A.180})$$

To solve these differential equations, we do need initial conditions. To obtain them, we should note that e^{Ct} takes the value of I when $t = 0$. As a result, $C_{11}^e(0) = I$, $C_{12}^e(0) = 0$, $C_{21}^e(0) = 0$ and $C_{22}^e(0) = I$. Using this, we can solve the differential equations.

To do so, we use the method of the integrating factor. That is, for the first differential equation (for $C_{21}^e(t)$) we multiply both sides of the equation by $e^{-A_2 t}$. This turns the differential equation into

$$e^{-A_2 t} \dot{C}_{21}^e(t) - e^{-A_2 t} A_2 C_{21}^e(t) = 0. \quad (\text{A.181})$$

By doing this, we can write the term on the left as a derivative. That is,

$$\frac{d}{dt} (e^{-A_2 t} C_{21}^e(t)) = 0. \quad (\text{A.182})$$

Next, we integrate the above expression from 0 to t . However, we are already using t in the above expression, so to prevent duplicate variables we first replace t by s in the above expression and the integrate up to t . The result is

$$[e^{-A_2 s} C_{21}^e(s)]_0^t = \int_0^t 0 ds. \quad (\text{A.183})$$

The integral on the right is zero. Hence we find that

$$e^{-A_2 t} C_{21}^e(t) - e^{-A_2 0} C_{21}^e(0) = 0. \quad (\text{A.184})$$

We have $e^{-A_2 0} = I$, but $C_{21}^e(0) = 0$, so the second term drops out. If we also note that $e^{-A_2 t}$ is nonsingular (a matrix exponential is always nonsingular) we find that

$$C_{21}^e(t) = 0. \quad (\text{A.185})$$

That proves the first part of the theorem. Solving the other differential equations goes in an identical way, so we will speed up the process a bit. For $C_{22}^e(t)$ we have

$$\begin{aligned} \frac{d}{dt} (e^{-A_2 t} C_{22}^e(t)) &= 0, \\ [e^{-A_2 s} C_{22}^e(s)]_0^t &= \int_0^t 0 ds, \\ e^{-A_2 t} C_{22}^e(t) - e^{-A_2 0} C_{22}^e(0) &= 0, \\ C_{22}^e(t) &= e^{A_2 t}, \end{aligned} \quad (\text{A.186})$$

where we have used $C_{22}^e(0) = I$. For $C_{11}^e(t)$ we similarly find that

$$\begin{aligned} \frac{d}{dt} (e^{-A_1 t} C_{11}^e(t)) &= e^{-A_1 t} B_1 C_{21}^e(t), \\ e^{-A_1 t} C_{11}^e(t) - e^{-A_1 0} C_{11}^e(0) &= 0, \\ C_{11}^e(t) &= e^{A_1 t}, \end{aligned} \quad (\text{A.187})$$

where we have used our earlier result of $C_{21}^e(t) = 0$. Finally there is $C_{12}^e(t)$. Now things

A

don't conveniently turn out to be zero on the right-hand side. In this case we get

$$\begin{aligned} \frac{d}{dt} (e^{-A_1 t} C_{12}^e(t)) &= e^{-A_1 t} B_1 C_{22}^e(t), \\ [e^{-A_1 s} C_{12}^e(s)]_0^t &= \int_0^t e^{-A_1 s} B_1 C_{22}^e(s) ds, \\ e^{-A_1 t} C_{12}^e(t) - e^{-A_1 0} C_{12}^e(0) &= \int_0^t e^{-A_1 s} B_1 e^{A_2 s} ds, \\ C_{12}^e(t) &= \int_0^t e^{A_1(t-s)} B_1 e^{A_2 s} ds. \end{aligned} \quad (\text{A.188})$$

This completes the proof. \square

In practice we do not use the above theorem to find e^{Ct} . After all, calculating matrix exponentials is relatively easy compared to evaluating matrix integrals. Instead, we calculate e^{Ct} and use the result to find the difficult integral $\int_0^t e^{A_1(t-s)} B_1 e^{A_2 s} ds$.

We can also expand the matrix C , granting us more complicated integrals. This is done in the next few theorems.

Theorem A.35. *If we define*

$$C = \begin{bmatrix} A_1 & B_1 & C_1 \\ 0 & A_2 & B_2 \\ 0 & 0 & A_3 \end{bmatrix}, \quad (\text{A.189})$$

and write e^{Ct} as in theorem A.34, then $C_{11}^e(t)$, $C_{12}^e(t)$, $C_{21}^e(t)$ and $C_{22}^e(t)$ are equal to the results of theorem A.34. Furthermore, we have

$$C_{31}^e(t) = 0, \quad (\text{A.190})$$

$$C_{32}^e(t) = 0, \quad (\text{A.191})$$

$$C_{33}^e(t) = e^{A_3 t}, \quad (\text{A.192})$$

$$C_{23}^e(t) = \int_0^t e^{A_2(t-s)} B_2 e^{A_3 s} ds, \quad (\text{A.193})$$

$$C_{13}^e(t) = \int_0^t e^{A_1(t-s)} B_1 e^{A_2(s-r)} B_2 e^{A_3 r} dr ds + \int_0^t e^{A_1(t-s)} C_1 e^{A_3 s}. \quad (\text{A.194})$$

Proof. The proof for this is pretty much identical to the proof of theorem A.34. We use $\frac{d}{dt} e^{Ct} = Ce^{Ct}$, which tells us that

$$\begin{bmatrix} \dot{C}_{11}^e(t) & \dot{C}_{12}^e(t) & \dot{C}_{13}^e(t) \\ \dot{C}_{21}^e(t) & \dot{C}_{22}^e(t) & \dot{C}_{23}^e(t) \\ \dot{C}_{31}^e(t) & \dot{C}_{32}^e(t) & \dot{C}_{33}^e(t) \end{bmatrix} = \begin{bmatrix} A_1 & B_1 & C_1 \\ 0 & A_2 & B_2 \\ 0 & 0 & A_3 \end{bmatrix} \begin{bmatrix} C_{11}^e(t) & C_{12}^e(t) & C_{13}^e(t) \\ C_{21}^e(t) & C_{22}^e(t) & C_{23}^e(t) \\ C_{31}^e(t) & C_{32}^e(t) & C_{33}^e(t) \end{bmatrix}. \quad (\text{A.195})$$

We can solve for e^{Ct} by going bottom-up. For the bottom row we have

$$\dot{C}_{31}^e(t) = A_3 C_{31}^e(t), \quad (\text{A.196})$$

$$\dot{C}_{32}^e(t) = A_3 C_{32}^e(t), \quad (\text{A.197})$$

$$\dot{C}_{33}^e(t) = A_3 C_{33}^e(t), \quad (\text{A.198})$$

which can be solved in the way we have seen in the proof of theorem A.34. We find that $C_{31}^e(t) = C_{32}^e(t) = 0$ and $C_{33}^e(t) = e^{A_3 t}$.

For the second row, we have

$$\dot{C}_{21}^e(t) = A_2 C_{21}^e(t) + B_2 C_{31}^e(t), \quad (\text{A.199})$$

$$\dot{C}_{22}^e(t) = A_2 C_{22}^e(t) + B_2 C_{32}^e(t), \quad (\text{A.200})$$

$$\dot{C}_{23}^e(t) = A_2 C_{23}^e(t) + B_2 C_{33}^e(t). \quad (\text{A.201})$$

Using our results for $C_{31}^e(t)$, $C_{32}^e(t)$ and $C_{33}^e(t)$, we find that $C_{21}^e(t) = 0$ and $C_{22}^e(t) = e^{A_2 t}$. For $C_{23}^e(t)$ we get, similarly to equation (A.188),

$$C_{23}^e(t) = \int_0^t e^{A_2(t-s)} B_2 e^{A_3 s} ds. \quad (\text{A.202})$$

Then there is the top row. Now we have

$$\dot{C}_{11}^e(t) = A_1 C_{11}^e(t) + B_1 C_{21}^e(t) + C_1 C_{31}^e(t), \quad (\text{A.203})$$

$$\dot{C}_{12}^e(t) = A_1 C_{12}^e(t) + B_1 C_{22}^e(t) + C_1 C_{32}^e(t), \quad (\text{A.204})$$

$$\dot{C}_{13}^e(t) = A_1 C_{13}^e(t) + B_1 C_{23}^e(t) + C_1 C_{33}^e(t). \quad (\text{A.205})$$

Again using our previous results, we immediately find that $C_{11}^e(t) = e^{A_1 t}$. Next, identically to equation (A.188), we have

$$C_{12}^e(t) = \int_0^t e^{A_1(t-s)} B_1 e^{A_2 s} ds. \quad (\text{A.206})$$

Finally there is the toughest term to deal with, which is $C_{13}^e(t)$. When applying the method of the integrating factor, we find that

$$\frac{d}{dt} (e^{-A_1 t} C_{13}^e(t)) = e^{-A_1 t} B_1 C_{23}^e(t) + e^{-A_1 t} C_1 C_{33}^e(t). \quad (\text{A.207})$$

Substituting for $C_{23}^e(t)$ and $C_{33}^e(t)$ will give us

$$\frac{d}{dt} (e^{-A_1 t} C_{13}^e(t)) = e^{-A_1 t} B_1 \int_0^t e^{A_2(t-s)} B_2 e^{A_3 s} ds + e^{-A_1 t} C_1 e^{A_3 t}. \quad (\text{A.208})$$

Next, we will integrate from 0 to t . However, we already have a t in the above expression, so in the above we will rename t to s . Except that we already have an s too, so we will rename s to r . In this way, all parameter names are ‘moved down’ in the alphabet. Doing so will result in

$$[e^{-A_1 s} C_{13}^e(s)]_0^t = \int_0^t \left(e^{-A_1 s} B_1 \int_0^s e^{A_2(s-r)} B_2 e^{A_3 r} dr + e^{-A_1 s} C_1 e^{A_3 s} \right) ds. \quad (\text{A.209})$$

Rewriting the result will give us

$$C_{13}^e(t) = \int_0^t \int_0^s e^{A_1(t-s)} B_1 e^{A_2(s-r)} B_2 e^{A_3 r} dr ds + \int_0^t e^{A_1(t-s)} C_1 e^{A_3 s} ds. \quad (\text{A.210})$$

This completes the proof. \square

A

We have got the result for a 3×3 matrix now. We can expand this to a 4×4 matrix.

Theorem A.36. *If we define*

$$C = \begin{bmatrix} A_1 & B_1 & C_1 & D_1 \\ 0 & A_2 & B_2 & C_2 \\ 0 & 0 & A_3 & B_3 \\ 0 & 0 & 0 & A_4 \end{bmatrix}, \quad (\text{A.211})$$

and write e^{Ct} as in theorem A.34, then the results of $C_{11}^e(t)$ up to $C_{33}^e(t)$ can be found in theorems A.34 and A.35. For the rest, we have

$$C_{41}^e(t) = 0, \quad (\text{A.212})$$

$$C_{42}^e(t) = 0, \quad (\text{A.213})$$

$$C_{43}^e(t) = 0, \quad (\text{A.214})$$

$$C_{44}^e(t) = e^{A_4 t}, \quad (\text{A.215})$$

$$C_{34}^e(t) = \int_0^t e^{A_3(t-s)} B_3 e^{A_4 s} ds, \quad (\text{A.216})$$

$$C_{24}^e(t) = \int_0^t e^{A_2(t-s)} B_2 e^{A_3(s-r)} B_3 e^{A_4 r} dr ds + \int_0^t e^{A_2(t-s)} C_2 e^{A_4 s} s, \quad (\text{A.217})$$

$$\begin{aligned} C_{14}^e(t) = & \int_0^t \int_0^s \int_0^r e^{A_1(t-s)} B_1 e^{A_2(s-r)} B_2 e^{A_3(r-q)} B_3 e^{A_4 q} dq dr ds + \int_0^t e^{A_1(t-s)} D_1 e^{A_4 s} ds \\ & + \int_0^t \int_0^s e^{A_1(t-s)} B_1 e^{A_2(s-r)} C_2 e^{A_4 r} dr ds + \int_0^t \int_0^s e^{A_1(t-s)} C_1 e^{A_3(s-r)} B_3 e^{A_4 r} dr ds. \end{aligned} \quad (\text{A.218})$$

Proof. The proof for almost all terms is identical to what was done in theorems A.34 and A.35, so we will not discuss that again. We will only look at the new term $C_{14}^e(t)$.

Our starting point will be the differential equation

$$\dot{C}_{14}^e(t) = A_1 C_{14}^e(t) + B_1 C_{24}^e(t) + C_1 C_{34}^e(t) + D_1 C_{44}^e(t). \quad (\text{A.219})$$

Using the method of the integrating factor and working out the left hand side will give us

$$C_{14}^e(t) = e^{A_1 t} \int_0^t (B_1 C_{24}^e(s) + C_1 C_{34}^e(s) + D_1 C_{44}^e(s)) ds. \quad (\text{A.220})$$

If we insert earlier results for $C_{24}^e(s)$, $C_{34}^e(s)$ and $C_{44}^e(s)$, and if we again shift all parameters down in the alphabet, we find our final result

$$\begin{aligned} C_{14}^e(t) = & \int_0^t \int_0^s \int_0^r e^{A_1(t-s)} B_1 e^{A_2(s-r)} B_2 e^{A_3(r-q)} B_3 e^{A_4 q} dq dr ds + \int_0^t e^{A_1(t-s)} D_1 e^{A_4 s} ds \\ & + \int_0^t \int_0^s e^{A_1(t-s)} B_1 e^{A_2(s-r)} C_2 e^{A_4 r} dr ds + \int_0^t \int_0^s e^{A_1(t-s)} C_1 e^{A_3(s-r)} B_3 e^{A_4 r} dr ds. \end{aligned} \quad (\text{A.221})$$

□

The order of the matrices in the integrals of the above theorem might seem haphazard. However, they are most certainly not. There is an ‘intuitive’ way of looking at the above equations, which make them more easy to recognize.

To see how it works, consider the matrix C as

$$C = \begin{bmatrix} A_1 & B_1 & C_1 & D_1 \\ 0 & A_2 & B_2 & C_2 \\ 0 & 0 & A_3 & B_3 \\ 0 & 0 & 0 & A_4 \end{bmatrix}. \quad (\text{A.222})$$

To find $C_{14}^e(t)$, we will start our ‘walk’ through the matrix at A_1 . From there, we may ‘jump’ to any matrix on the right. For instance, we may jump to B_1 . From there, we must jump downward, back to the diagonal. For our case, that’s A_2 . From there the process continues. For instance, we can jump to C_2 . From there, we have to jump down to A_4 . Once we arrive at A_4 , we’re done.

Next, we will set up an expression for our walk. For that, we start with $e^{A_1 t}$. Then we look at each set of jumps which we did away from and back to the diagonal. If we jumped away from A_i to some matrix X_i (where X can be B , C , D and so on) and then downward to A_j , then we should add $e^{-A_i s} X_i e^{A_j s}$ to our expression, or use r if we’ve already used s , and so on. Furthermore, we should add an integral. Its limits should run from 0 to the previous parameter (like s , r) which we added. If we do this for our walk, then we have

$$\text{Result of walk: } \int_0^t \int_0^s e^{A_1 t} (e^{-A_1 s} B_1 e^{A_2 s}) (e^{-A_2 r} C_2 e^{A_4 r}) ds dr. \quad (\text{A.223})$$

If we set up an expression for every possible walk like this through our matrix, and add up all the results, then we arrive at the expression for $C_{14}^e(t)$. A similar trick also works for any element $C_{ij}^e(t)$, except now we need to start our walk at A_i and end it at A_j . So this may make it easier to remember the results of the above theorems.

Finally, because we may need it, we also look at the result of a matrix C of 5 by 5 submatrices.

Theorem A.37. *If we define*

$$C = \begin{bmatrix} A_1 & B_1 & C_1 & D_1 & E_1 \\ 0 & A_2 & B_2 & C_2 & D_2 \\ 0 & 0 & A_3 & B_3 & C_3 \\ 0 & 0 & 0 & A_4 & B_4 \\ 0 & 0 & 0 & 0 & A_5 \end{bmatrix}, \quad (\text{A.224})$$

A

and write e^{Ct} as in theorem A.34, then we have

$$\begin{aligned} C_{15}^e(t) = & \int_0^t \int_0^s \int_0^r \int_0^q e^{A_1(t-s)} B_1 e^{A_2(s-r)} B_2 e^{A_3(r-q)} B_3 e^{A_4(q-p)} B_4 e^{A_5 p} dp dq dr ds \\ & + \int_0^t \int_0^s \int_0^r e^{A_1(t-s)} B_1 e^{A_2(s-r)} B_2 e^{A_3(r-q)} C_3 e^{A_5 q} dq dr ds \\ & + \int_0^t \int_0^s \int_0^r e^{A_1(t-s)} B_1 e^{A_2(s-r)} C_2 e^{A_4(r-q)} B_4 e^{A_5 q} dq dr ds \\ & + \int_0^t \int_0^s \int_0^r e^{A_1(t-s)} C_1 e^{A_3(s-r)} B_3 e^{A_4(r-q)} B_4 e^{A_5 q} dq dr ds \\ & + \int_0^t \int_0^s e^{A_1(t-s)} B_1 e^{A_2(s-r)} D_2 e^{A_5 r} dr ds + \int_0^t \int_0^s e^{A_1(t-s)} D_1 e^{A_4(s-r)} B_4 e^{A_5 r} dr ds \\ & + \int_0^t \int_0^s e^{A_1(t-s)} C_1 e^{A_3(s-r)} C_3 e^{A_5 r} dr ds + \int_0^t e^{A_1(t-s)} E_1 e^{A_5 s} ds. \end{aligned} \quad (\text{A.225})$$

Proof. The proof for this is done identically as the proofs of the previous three theorems (A.34, A.35 and A.36). After applying the method of the integrating factor, we will find

$$C_{15}^e(t) = e^{A_1 t} \int_0^t (B_1 C_{25}^e(s) + C_1 C_{35}^e(s) + D_1 C_{45}^e(s) + E_1 C_{55}^e(s)) ds. \quad (\text{A.226})$$

If we work this out, keeping track of all the integrals, we will arrive at the above expression for $C_{15}^e(t)$. \square

In this thesis, we will generally set all C , D and E matrices to zero. As a result, all matrix terms (yes, even $C_{15}^e(t)$) will consist of only one term. That term may contain multiple integrals, but it is still an expression which will fit on one line.

A.5.2. USING MATRIX EXPONENTIALS TO SOLVE LYAPUNOV EQUATIONS

Usually Lyapunov equations are solved through matrix inversion (Theorem A.23). However, it is also possible to use matrix exponentials to find Lyapunov solutions. For this subsection it is important to recall the notation described in subsection A.4.1.

We start with solving a Lyapunov equation using matrix exponentials.

Theorem A.38. *For any k_1 and k_2 satisfying $k_1 + k_2 = 2k$, and for any matrix A , we have*

$$X_{k\alpha}^Q(t_1, t_2) = e^{A_{k_1\alpha} t_2} [I \ 0] \exp \left(\begin{bmatrix} -A_{k_1\alpha} & Q \\ 0 & A_{k_2\alpha}^T \end{bmatrix} (t_2 - t_1) \right) \begin{bmatrix} 0 \\ I \end{bmatrix} e^{A_{k_2\alpha}^T t_1}. \quad (\text{A.227})$$

Proof. From theorem A.34 we know that

$$\int_0^t e^{A_1(t-s)} B_1 e^{A_2 s} ds = [I \ 0] \exp \left(\begin{bmatrix} A_1 & B_1 \\ 0 & A_2 \end{bmatrix} t \right) \begin{bmatrix} 0 \\ I \end{bmatrix}. \quad (\text{A.228})$$

If we use $A_1 = -A_{k_1\alpha}$, $B_1 = Q$, $A_2 = A_{k_2\alpha}^T$ and $t = t_2 - t_1$, then we get

$$\int_0^{t_2 - t_1} e^{A_{k_1\alpha}(s-(t_2-t_1))} Q e^{A_{k_2\alpha}^T s} ds = [I \ 0] \exp \left(\begin{bmatrix} -A_{k_1\alpha} & Q \\ 0 & A_{k_2\alpha}^T \end{bmatrix} (t_2 - t_1) \right) \begin{bmatrix} 0 \\ I \end{bmatrix}. \quad (\text{A.229})$$

If we now substitute s by $s - t_1$, updating integral limits accordingly, and subsequently pull terms not depending on s out of the integral, we directly find that

$$e^{-A_{k_1\alpha}t_2} \left(\int_{t_1}^{t_2} e^{A_{k_1\alpha}s} Q e^{A_{k_2\alpha}^T s} ds \right) e^{-A_{k_2\alpha}^T t_1} = [I \ 0] \exp \left(\begin{bmatrix} -A_{k_1\alpha} & Q \\ 0 & A_{k_2\alpha}^T \end{bmatrix} (t_2 - t_1) \right) \begin{bmatrix} 0 \\ I \end{bmatrix}. \quad (\text{A.230})$$

The part within brackets now equals $X_{k\alpha}^Q(t_1, t_2)$, with $k = \frac{k_1+k_2}{2}$. As a result, we have

$$X_{k\alpha}^Q(t_1, t_2) = e^{A_{k_1\alpha}t_2} [I \ 0] \exp \left(\begin{bmatrix} -A_{k_1\alpha} & Q \\ 0 & A_{k_2\alpha}^T \end{bmatrix} (t_2 - t_1) \right) \begin{bmatrix} 0 \\ I \end{bmatrix} e^{A_{k_2\alpha}^T t_1}, \quad (\text{A.231})$$

which completes this proof. \square

It is interesting to note that the above theorem does not require A or any other matrix to be Sylvester. It works for any matrix A .

Now that we know how to find $X_{k\alpha}^Q(t_1, t_2)$ for each matrix A , we look at how to find $\tilde{X}_{k\alpha}^Q(t)$, defined by A.121.

Theorem A.39. *When $\tilde{X}_{k_1\alpha, k_2\alpha}^Q(t)$ is defined as in (A.121), then for any n we have*

$$\begin{aligned} \tilde{X}_{k\alpha}^Q(t) &= [I \ 0] \exp \left(\begin{bmatrix} A_{k\alpha} & Q \\ 0 & A \end{bmatrix} t \right) \begin{bmatrix} 0 \\ I \end{bmatrix} \\ &= e^{-n\alpha t} [I \ 0] \exp \left(\begin{bmatrix} A_{(k+n)\alpha} & Q \\ 0 & A_{n\alpha} \end{bmatrix} t \right) \begin{bmatrix} 0 \\ I \end{bmatrix}. \end{aligned} \quad (\text{A.232})$$

Proof. From theorem A.34 we know that

$$\int_0^t e^{A_1(t-s)} B_1 e^{A_2 s} ds = [I \ 0] \exp \left(\begin{bmatrix} A_1 & B_1 \\ 0 & A_2 \end{bmatrix} t \right) \begin{bmatrix} 0 \\ I \end{bmatrix}. \quad (\text{A.233})$$

If we use $A_1 = A_{(k+n)\alpha}$, $B_1 = Q$ and $A_2 = A_{n\alpha}$ and premultiply by $e^{-n\alpha t}$, then this becomes

$$e^{-n\alpha t} \int_0^t e^{A_{(k+n)\alpha}(t-s)} Q e^{A_{n\alpha} s} ds = e^{-n\alpha t} [I \ 0] \exp \left(\begin{bmatrix} A_{(k+n)\alpha} & Q \\ 0 & A_{n\alpha}^T \end{bmatrix} t \right) \begin{bmatrix} 0 \\ I \end{bmatrix}. \quad (\text{A.234})$$

Because the left hand side of the expression equals the definition of $\tilde{X}_{k\alpha}^Q(t)$, we have proven (A.232) for any n . \square

So now we have two different yet effective methods of calculating Lyapunov solutions $X_{k\alpha}^Q(t_1, t_2)$. We will need these when we start solving Lyapunov equations in Appendix C.2. The term $\tilde{X}^Q(T)$ will start to come in handy from Appendix C.4.2 onwards.

A.5.3. A COMPARISON BETWEEN THE TWO METHODS

The question remains which of the two methods we have of finding Lyapunov solutions $X^Q(t)$ is better. Is it wiser to use the Lyapunov solution method (Theorem A.23) or the matrix exponentials method (Theorem A.38)?

A

There are a few cases in which the choice is obvious. If we need to find the infinite-time solution X^Q , then using matrix exponentials is impossible. Similarly, if we want to find $X^Q(t)$ when A is not Sylvester, using matrix inversion is impossible. In practice non-Sylvester matrices are quite rare though, so what do we do then?

In that case the determining factor is the numerical accuracy of both methods. Luckily, this is something which we can investigate. We can find the parameter $X^Q(t)$ through both methods, for various random matrices A and Q and for a given t . Then we test both results for their numerical accuracy.

The tricky part is measuring the amount of numerical accuracies that take place. To get some indication of this, we can calculate the quantity

$$AX^Q(t) + X^Q(t)A^T + Q - e^{At}Qe^{A^T t} \quad (\text{A.235})$$

for both calculated values of $X^Q(t)$. According to Theorem A.26 this quantity must equal the zero matrix, so any discrepancy is caused by numerical inaccuracies. The more we have, the more numerical inaccuracies there are.

If we add up the magnitude of all the elements of the resulting matrix, we get an indication of the numerical accuracies that are present. To get rid of scale effects, we will look at the ratio of these two errors. If we calculate this ratio for many different random matrices A and Q , and do so for varying times t , then we get Figure A.1.

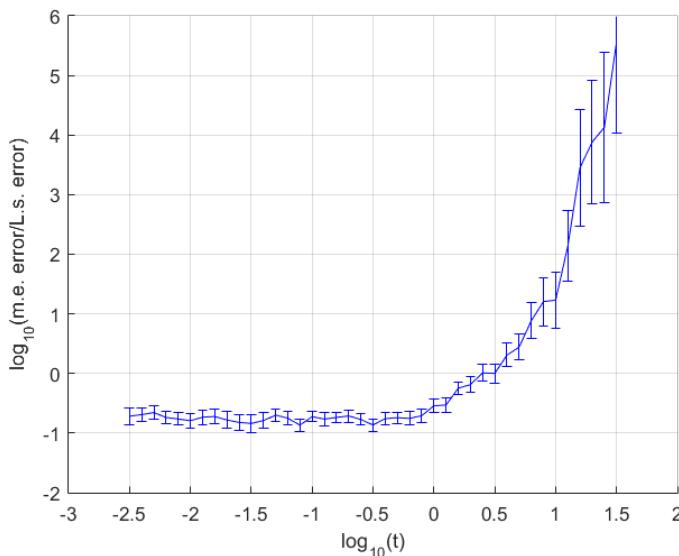


Figure A.1: The numerical error of the matrix exponential method compared to the Lyapunov solution method. Each error bar is the result of 100 experiments with different random matrices A and Q . The horizontal axis shows the time t in logarithmic scale. It ranges from $10^{-2.5}$ to $10^{1.5}$. The vertical axis shows the numerical error of the matrix exponential method divided by that of the Lyapunov solution method, also in logarithmic scale.

From Figure A.1 we can see that for small t (up to $t \approx 10^{0.5} \approx 3$ s) the magnitude of the numerical errors within the matrix exponential method is roughly $10^{-0.8} \approx \frac{1}{6}$ times that

of the Lyapunov solution method. So the matrix exponential method works better. For larger times t the performance of the matrix exponential method quickly and severely degrades though.

Very similar results were also found by [Wahlström et al. \(2014\)](#). This confirms that the matrix exponential method is actually more accurate for small t , generally as long as $t < 1$, though this of course depends on the magnitude of the matrix A . For $t \approx 1$ it does not matter much which method to use. For larger t it is not recommended to apply the matrix exponential method in practice.

A.6. MISCELLANEOUS

We close this chapter off with a theorem which does not really fit in anywhere, so I just put it here. It concerns the sum of sinusoids.

Theorem A.40. *We can write any sinusoid with amplitude A , frequency f and phase difference ϕ as the sum of a sine and a cosine with the same frequency but without a phase difference. That is,*

$$A\sin(2\pi f t + \phi) = a\sin(2\pi f t) + b\cos(2\pi f t), \quad (\text{A.236})$$

where a and b can be found through

$$a = A\cos(\phi), \quad (\text{A.237})$$

$$b = A\sin(\phi), \quad (\text{A.238})$$

or conversely where A and ϕ can be found through

$$A = \sqrt{a^2 + b^2}, \quad (\text{A.239})$$

$$\phi = \tan^{-1}(b, a). \quad (\text{A.240})$$

Proof. Our starting point is the *trigonometric addition formula*, sometimes also known as the *Prosthaphaeresis formula* or *Simpson's formula*,

$$\sin(\alpha + \beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta). \quad (\text{A.241})$$

If we multiply this by the amplitude A and substitute $\alpha = 2\pi f t$ and $\beta = \phi$, we can write

$$\begin{aligned} A\sin(2\pi f t + \phi) &= A\cos(\phi)\sin(2\pi f t) + A\sin(\phi)\cos(2\pi f t) \\ &= a\sin(2\pi f t) + b\cos(2\pi f t), \end{aligned} \quad (\text{A.242})$$

where $a = A\cos(\phi)$ and $b = A\sin(\phi)$. This proves the transformation one way. For the other way, we can find A through

$$A = \sqrt{A^2(\cos(\phi)^2 + \sin(\phi)^2)} = a^2 + b^2. \quad (\text{A.243})$$

We can find ϕ either through $\sin(\phi) = \frac{b}{A}$ or $\cos(\phi) = \frac{a}{A}$. Using $\tan(\phi) = \frac{b}{a}$ does not always work, because $\tan(\phi) = \frac{b}{a} = \frac{-b}{-a}$. So we may find the wrong phase ϕ , unless we specifically take the signs of a and b into account. Most programming languages have a function `atan2(b, a)` which does exactly this. If we use this function, then we can directly find ϕ without first having to calculate A . \square

B

PROBABILITY THEORY

Summary — A random variable is a variable which can take different values, based on certain probabilities. The way in which it does this – its distribution – is described by its probability density function. When we deal with multiple random variables, we need to consider their joint distribution. As long as these random variables are not independent, then learning more about one of these variables will mean we also learn more about the others.

We can obtain the distribution of a random variable either based on prior data, or by performing measurements. When multiple independent distributions of a random variable are known, then these distributions can be merged together.

Important properties of random variables are their mean (their expected value) and their variance. When dealing with multiple random variables, the covariance between these variables is also important.

A very common type of random variable is the Gaussian random variable. Its distribution is fully determined by the mean vector and the covariance matrix. Such random variables have a variety of interesting properties. For instance, linear combinations of Gaussian random variables will always result in yet another Gaussian random variable.

We can learn more about Gaussian random variables by performing measurements. It is also possible to incorporate these measurements if we only measure linear combinations of Gaussian random variables. In addition, when the Gaussian random variables are conditionally independent, then there may be more (computationally) efficient ways of incorporating the measurement data into the distributions of the random variables.

In this chapter we will recap some basic probability theories. As prior knowledge, you should at least have a bit of a clue about what random variables, probability density functions and Gaussian distributions are, even though we will repeat the most important properties.

B

We will start by looking at the basics of probability density functions in Section B.1, continue by examining the properties of the mean and covariance in Section B.2 and then find out how probability density functions change when the corresponding random variable is subjected to transformations (Section B.3). We then switch our focus to Gaussian random variables, first studying their basic properties (Section B.4), then considering methods of manipulating them (Section B.5) and finally figuring out what happens when Gaussian distributions are conditionally independent (Section B.6).

B.1. INTRODUCING THE PROBABILITY DENSITY FUNCTION

A fundamental part in probability theory is the probability density function of a random variable. In this section we will look at how it is defined, starting with the default definition (Section B.1.1), adding the joint definition (Section B.1.2) and finalizing with the conditional definition (Section B.1.3). In the end we also look at a few special probability density functions (Section B.1.4).

B.1.1. DEFINITION OF THE PROBABILITY DENSITY FUNCTION

Consider a scalar variable \underline{x} . The underline indicates that it is a *random variable*, meaning that it does not have a deterministic value but can have one of various values, each with its own probability. These probabilities depend on the *distribution* of the random variable, which is indicated by the *Probability Density Function* (PDF)¹ $f_{\underline{x}}(x)$. This function $f_{\underline{x}}(x)$ is defined as²

$$f_{\underline{x}}(x) \equiv \frac{p(x \leq \underline{x} < x + dx)}{dx}, \quad (\text{B.1})$$

with $p(A)$ denoting the probability that the given event A occurs. As a result of the above, any PDF $f_{\underline{x}}(x)$ must satisfy

$$\int_{-\infty}^{\infty} f_{\underline{x}}(x) dx = \int_{-\infty}^{\infty} p(x \leq \underline{x} < x + dx) = p(-\infty \leq \underline{x} < \infty) = 1. \quad (\text{B.2})$$

We can extend the above definition to *random vectors* \underline{x} , where the vector size is denoted by n . For the case where $n = 2$, we have

$$f_{\underline{x}}(\underline{x}) \equiv \frac{p(x_1 \leq \underline{x}_1 < x_1 + dx_1, x_2 \leq \underline{x}_2 < x_2 + dx_2)}{dx_1 dx_2} = \frac{p(\underline{x} \leq \underline{x} + d\underline{x})}{d\underline{x}}. \quad (\text{B.3})$$

¹In most literature, the subscript x at PDFs is not written, because it is clear which random variable the PDF belongs to. In addition, often the letter p is used instead of f . So $p(x)$ then actually denotes a probability density function.

In this appendix, we will stick with the notation $f_{\underline{x}}(x)$ for PDFs, while $p(\dots)$ is used to indicate probabilities of events. However, in the main body of this thesis, we will most often stick with the notation commonly used in the GP community, where $p(x)$ denotes a PDF.

²For discrete random variables things work differently. In this thesis we will only consider continuous random variables though.

Note that dividing by the vector differential $d\mathbf{x}$ actually means dividing by its (scalar) ‘volume’. That is, the product of all elements of the vector $d\mathbf{x}$.

For such vector random variables, the PDF now must satisfy (for $n = 2$)

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{\underline{x}}(\mathbf{x}) dx_1 dx_2 = \int_X f_{\underline{x}}(\mathbf{x}) d\mathbf{x} = 1. \quad (\text{B.4})$$

B

The set X in the above equation denotes the set of all possible values which \mathbf{x} may take, which is generally the full space \mathbb{R}^n .

All the above definitions and properties can be expanded identically for any $n > 2$.

B.1.2. JOINT DISTRIBUTIONS

Suppose that we have a random vector \underline{x} , which consists of two separate random vectors \underline{x}_a and \underline{x}_b . So $\underline{x} = \begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix}$. Also suppose that we know the *joint PDF* $f_{\underline{x}}(\mathbf{x}) = f_{\underline{x}_a, \underline{x}_b}(\mathbf{x}_a, \mathbf{x}_b)$. How do we then find the PDF $f_{\underline{x}_a}(\mathbf{x}_a)$ of \underline{x}_a ? (Or equivalently, that of \underline{x}_b ?) This is answered by the following theorem, which shows us the process of *marginalization*.

Theorem B.1. *When \underline{x}_a and \underline{x}_b have a joint PDF $f_{\underline{x}_a, \underline{x}_b}(\mathbf{x}_a, \mathbf{x}_b)$, then the PDF of \underline{x}_a is*

$$f_{\underline{x}_a}(\mathbf{x}_a) = \int_{X_b} f_{\underline{x}_a, \underline{x}_b}(\mathbf{x}_a, \mathbf{x}_b) d\mathbf{x}_b. \quad (\text{B.5})$$

Proof. We start with the definition of the joint PDF. It equals

$$f_{\underline{x}_a, \underline{x}_b}(\mathbf{x}_a, \mathbf{x}_b) = \frac{p(\mathbf{x}_a \leq \underline{x}_a < \mathbf{x}_a + d\mathbf{x}_a, \mathbf{x}_b \leq \underline{x}_b < \mathbf{x}_b + d\mathbf{x}_b)}{d\mathbf{x}_a d\mathbf{x}_b}. \quad (\text{B.6})$$

When we integrate over X_b , which is the set of all possible values of \mathbf{x}_b , we find that

$$\begin{aligned} \int_{X_b} f_{\underline{x}_a, \underline{x}_b}(\mathbf{x}_a, \mathbf{x}_b) d\mathbf{x}_b &= \int_{X_b} \frac{p(\mathbf{x}_a \leq \underline{x}_a < \mathbf{x}_a + d\mathbf{x}_a, \mathbf{x}_b \leq \underline{x}_b < \mathbf{x}_b + d\mathbf{x}_b)}{d\mathbf{x}_a} d\mathbf{x}_b \\ &= \frac{p(\mathbf{x}_a \leq \underline{x}_a < \mathbf{x}_a + d\mathbf{x}_a, -\infty \leq \underline{x}_b < \infty)}{d\mathbf{x}_a} \\ &= \frac{p(\mathbf{x}_a \leq \underline{x}_a < \mathbf{x}_a + d\mathbf{x}_a)}{d\mathbf{x}_a} \\ &= f_{\underline{x}_a}(\mathbf{x}_a). \end{aligned} \quad (\text{B.7})$$

Note that, because the event $-\infty \leq \underline{x}_b \leq \infty$ is obviously true, we have dropped it from our probability function. \square

B.1.3. CONDITIONAL DISTRIBUTIONS

When we have two random variables \underline{x}_a and \underline{x}_b , then these variables may depend on each other. That is, if we know that \underline{x}_b actually equals some measured value $\hat{\mathbf{x}}_b$, then we can say something more about the distribution of \underline{x}_a . In fact, we write the resulting random variable as $\underline{x}_a | \underline{x}_b = \hat{\mathbf{x}}_b$ or shorter as $\underline{x}_a | \hat{\mathbf{x}}_b$. The conditional probability bar $|$ can be read as ‘given that’. This parameter now has $f_{\underline{x}_a | \hat{\mathbf{x}}_b}(\mathbf{x}_a)$ as PDF. But what is this PDF?

Officially, the conditional probability of an event A given that an event B occurred, is defined as

$$p(A|B) = \frac{p(A, B)}{p(B)}. \quad (\text{B.8})$$

B

Using this definition, we can find the PDF of our conditional distribution.

Theorem B.2. Assume that the random variables \underline{x}_a and \underline{x}_b have a joint distribution $f_{\underline{x}_a, \underline{x}_b}(\underline{x}_a, \underline{x}_b)$. When it is given that \underline{x}_b equals a value $\hat{\underline{x}}_b$, then the PDF of \underline{x}_a becomes

$$f_{\underline{x}_a|\hat{\underline{x}}_b}(\underline{x}_a) = \frac{f_{\underline{x}_a, \underline{x}_b}(\underline{x}_a, \hat{\underline{x}}_b)}{f_{\underline{x}_b}(\hat{\underline{x}}_b)}. \quad (\text{B.9})$$

Proof. Before we start, we should note that, for infinitesimally small $d\hat{\underline{x}}_b$, we have $p(\hat{\underline{x}}_b \leq \underline{x}_b < \hat{\underline{x}}_b + d\hat{\underline{x}}_b) \rightarrow p(\underline{x}_b = \hat{\underline{x}}_b)$. With this knowledge, we can find that

$$\begin{aligned} f_{\underline{x}_a|\hat{\underline{x}}_b}(\underline{x}_a) &\equiv \frac{p(\underline{x}_a \leq \underline{x}_a < \underline{x}_a + d\underline{x}_a | \underline{x}_b = \hat{\underline{x}}_b)}{d\underline{x}_a} \\ &= \frac{p(\underline{x}_a \leq \underline{x}_a < \underline{x}_a + d\underline{x}_a, \underline{x}_b = \hat{\underline{x}}_b)}{d\underline{x}_a} \frac{1}{p(\underline{x}_b = \hat{\underline{x}}_b)} \\ &= \frac{p(\underline{x}_a \leq \underline{x}_a < \underline{x}_a + d\underline{x}_a, \hat{\underline{x}}_b \leq \underline{x}_b < \hat{\underline{x}}_b + d\hat{\underline{x}}_b)}{d\underline{x}_a d\hat{\underline{x}}_b} \frac{d\hat{\underline{x}}_b}{p(\hat{\underline{x}}_b \leq \underline{x}_b < \hat{\underline{x}}_b + d\hat{\underline{x}}_b)} \\ &= \frac{f_{\underline{x}_a, \underline{x}_b}(\underline{x}_a, \hat{\underline{x}}_b)}{f_{\underline{x}_b}(\hat{\underline{x}}_b)}. \end{aligned} \quad (\text{B.10})$$

This proves the theorem. \square

It may happen that \underline{x}_a and \underline{x}_b are *independent*. Theoretically, this is defined as having

$$f_{\underline{x}_a, \underline{x}_b}(\underline{x}_a, \underline{x}_b) = f_{\underline{x}_a}(\underline{x}_a) f_{\underline{x}_b}(\underline{x}_b). \quad (\text{B.11})$$

In practice, what it means is that, even if we know that \underline{x}_b equals some value $\hat{\underline{x}}_b$, we still cannot say anything extra about the distribution of \underline{x}_a . In other words, the random variable \underline{x}_a (without any additional data) has the same distribution as the random variable $\underline{x}_a|\hat{\underline{x}}_b$. This means that

$$f_{\underline{x}_a|\hat{\underline{x}}_b}(\underline{x}_a) = f_{\underline{x}_a}(\underline{x}_a). \quad (\text{B.12})$$

B.1.4. SPECIAL CASES OF THE PROBABILITY DENSITY FUNCTION

The PDF describes how much we know about a random variable \underline{x} . There are two limit cases: where we know everything about a random variable and where we know absolutely nothing. What are the PDFs in these cases?

First, suppose that we know \underline{x} deterministically. In other words, we know that \underline{x} equals a given number c . The PDF of \underline{x} is now given by

$$f_{\underline{x}}(\underline{x}) = \delta(\underline{x} - c), \quad (\text{B.13})$$

with $\delta(\dots)$ being the *delta function*. This PDF, called the *delta distribution*, is zero everywhere, except at the value c , where it has an infinitely large peak.

But what if we know absolutely nothing about \underline{x} ? When its value can literally be anywhere between $-\infty$ and ∞ , with equal probability? In this case the PDF is a constant value γ . So,

$$f_{\underline{x}}(\underline{x}) = \gamma, \quad (\text{B.14})$$

I call this distribution the *null distribution*. But what exactly is the value of γ ? Well, we know that the PDF must satisfy

$$\int_{-\infty}^{\infty} f_{\underline{x}}(\underline{x}) d\underline{x} = \int_{-\infty}^{\infty} \gamma d\underline{x} = 1. \quad (\text{B.15})$$

γ is hence defined as the number which satisfies the above equation. It is basically an infinitely small number, which is not yet zero. (This distribution is a type of singular distribution, but we will not go further into that here.)

B

B.2. THE MEAN AND THE COVARIANCE

Important properties of any random variable \underline{x} are the mean and the covariance. We can find them from the probability density function. In this section we will look at how they are defined and what properties they have. We start with their definitions and fundamental properties (Section B.2.1), then look at how the mean and covariance are affected when we linearly transform the corresponding random variables (Section B.2.2) and finally we look at a few other important properties of the mean and the covariance (Section B.2.3).

B.2.1. THE FUNDAMENTALS BEHIND THE MEAN AND THE COVARIANCE

The mean of a random variable \underline{x} is defined as

$$\mathbb{E}[\underline{x}] = \int_{-\infty}^{\infty} \underline{x} f_{\underline{x}}(\underline{x}) d\underline{x}. \quad (\text{B.16})$$

The covariance matrix is then defined as

$$\begin{aligned} \mathbb{V}[\underline{x}] &= \mathbb{E} \left[(\underline{x} - \mathbb{E}[\underline{x}]) (\underline{x} - \mathbb{E}[\underline{x}])^T \right] \\ &= \int_{-\infty}^{\infty} (\underline{x} - \mathbb{E}[\underline{x}]) (\underline{x} - \mathbb{E}[\underline{x}])^T f_{\underline{x}}(\underline{x}) d\underline{x}. \end{aligned} \quad (\text{B.17})$$

Alternatively, we can define the covariance matrix of two different random variables as

$$\begin{aligned} \mathbb{V}[\underline{x}_a, \underline{x}_b] &= \mathbb{E} \left[(\underline{x}_a - \mathbb{E}[\underline{x}_a]) (\underline{x}_b - \mathbb{E}[\underline{x}_b])^T \right] \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (\underline{x}_a - \mathbb{E}[\underline{x}_a]) (\underline{x}_b - \mathbb{E}[\underline{x}_b])^T f_{\underline{x}_a, \underline{x}_b}(\underline{x}_a, \underline{x}_b) d\underline{x}_a d\underline{x}_b. \end{aligned} \quad (\text{B.18})$$

So the notation $\mathbb{V}[\underline{x}_a]$ is just a shorthand for $\mathbb{V}[\underline{x}_a, \underline{x}_a]$.

The covariance matrix $\mathbb{V}[\underline{x}]$ of a single random vector \underline{x} always satisfies certain properties. First of all, it is symmetric. After all, $\mathbb{E}[(x_i - \mu_i)(x_j - \mu_j)]$ equals $\mathbb{E}[(x_j - \mu_j)(x_i - \mu_i)]$. Furthermore, it is always positive semidefinite and usually even positive definite. Why exactly this is the case will become clear in Section B.4.5, and in Section B.5.1 we will look at what a negative definite covariance matrix might mean.

B.2.2. LINEAR TRANSFORMATIONS OF RANDOM VARIABLES

Let's take a random variable \underline{x} , (left-)multiply it by a matrix P and add a constant deterministic vector \mathbf{c} . What effect will this then have on the mean and covariance matrix?

Theorem B.3. Consider the random variable \underline{x} of size n , the $p \times n$ matrix P and the (deterministic) vector \mathbf{c} of size n . We have

$$\mathbb{E}[P\underline{x} + \mathbf{c}] = P\mathbb{E}[\underline{x}] + \mathbf{c}, \quad (\text{B.19})$$

$$\mathbb{V}[P\underline{x} + \mathbf{c}] = P\mathbb{V}[\underline{x}]P^T. \quad (\text{B.20})$$

Proof. This can be proven by applying the definition of the mean and covariance. For the mean, we have

$$\begin{aligned} \mathbb{E}[P\underline{x} + \mathbf{c}] &= \int_{-\infty}^{\infty} (P\underline{x} + \mathbf{c}) f_{\underline{x}}(\underline{x}) d\underline{x} \\ &= P \int_{-\infty}^{\infty} \underline{x} f_{\underline{x}}(\underline{x}) d\underline{x} + \int_{-\infty}^{\infty} \mathbf{c} f_{\underline{x}}(\underline{x}) d\underline{x} \\ &= P\mathbb{E}[\underline{x}] + \mathbf{c}. \end{aligned} \quad (\text{B.21})$$

(Note that we have applied (B.2) in the last step.) This actually confirms that the expectation operator is a linear operator, meaning that it satisfies (B.19).

To prove this theorem for the variance, we should already apply our result for the mean. When we do, we find that

$$\begin{aligned} \mathbb{V}[P\underline{x} + \mathbf{c}] &= \mathbb{E}[(P\underline{x} + \mathbf{c} - \mathbb{E}[P\underline{x} + \mathbf{c}])(P\underline{x} + \mathbf{c} - \mathbb{E}[P\underline{x} + \mathbf{c}])^T] \\ &= \mathbb{E}[P(\underline{x} - \mathbb{E}[\underline{x}])(\underline{x} - \mathbb{E}[\underline{x}])^T P^T] \\ &= P\mathbb{V}[\underline{x}]P^T. \end{aligned} \quad (\text{B.22})$$

This completes the proof. \square

B.2.3. FURTHER PROPERTIES OF THE MEAN AND THE COVARIANCE

Now that we know how to deal with linear transformations of random variables, we can look at a few other properties of random variables.

Theorem B.4. Consider the random variables \underline{x}_a and \underline{x}_b . We have

$$\mathbb{E}[\underline{x}_a + \underline{x}_b] = \mathbb{E}[\underline{x}_a] + \mathbb{E}[\underline{x}_b], \quad (\text{B.23})$$

$$\mathbb{V}[\underline{x}_a + \underline{x}_b] = \mathbb{V}[\underline{x}_a] + \mathbb{V}[\underline{x}_b] + \mathbb{V}[\underline{x}_a, \underline{x}_b] + \mathbb{V}[\underline{x}_b, \underline{x}_a]. \quad (\text{B.24})$$

Proof. Consider the vector $\underline{x} = \begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix}$. We now (per definition) have

$$\mathbb{E}[\underline{x}] = \mathbb{E}\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} = \begin{bmatrix} \mathbb{E}[\underline{x}_a] \\ \mathbb{E}[\underline{x}_b] \end{bmatrix}, \quad (\text{B.25})$$

$$\mathbb{V}[\underline{x}] = \mathbb{V}\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} = \begin{bmatrix} \mathbb{V}[\underline{x}_a, \underline{x}_a] & \mathbb{V}[\underline{x}_a, \underline{x}_b] \\ \mathbb{V}[\underline{x}_b, \underline{x}_a] & \mathbb{V}[\underline{x}_b, \underline{x}_b] \end{bmatrix}. \quad (\text{B.26})$$

If we define $P = [I \quad I]$ and $\mathbf{c} = \mathbf{0}$ and apply Theorem B.3, then the desired result immediately follows. \square

Theorem B.5. *The covariance $\mathbb{V}[\underline{x}_a, \underline{x}_b]$ of two random variables \underline{x}_a and \underline{x}_b equals*

$$\mathbb{V}[\underline{x}_a, \underline{x}_b] = \mathbb{E}[\underline{x}_a \underline{x}_b^T] - \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T. \quad (\text{B.27})$$

Proof. We will start with our proof at the definition of $\mathbb{V}[\underline{x}_a, \underline{x}_b]$. We expand that expression until we arrive at our final result. So,

$$\begin{aligned} \mathbb{V}[\underline{x}_a, \underline{x}_b] &= \mathbb{E}[(\underline{x}_a - \mathbb{E}[\underline{x}_a])(\underline{x}_b - \mathbb{E}[\underline{x}_b])^T] \\ &= \mathbb{E}[\underline{x}_a \underline{x}_b^T - \underline{x}_a \mathbb{E}[\underline{x}_b]^T - \mathbb{E}[\underline{x}_a] \underline{x}_b^T + \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T] \\ &= \mathbb{E}[\underline{x}_a \underline{x}_b^T] - \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T - \mathbb{E}[\mathbb{E}[\underline{x}_a] \underline{x}_b^T] + \mathbb{E}[\mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T] \\ &= \mathbb{E}[\underline{x}_a \underline{x}_b^T] - \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T - \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b^T] + \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T \\ &= \mathbb{E}[\underline{x}_a \underline{x}_b^T] - \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T. \end{aligned} \quad (\text{B.28})$$

Note that we have applied Theorem B.3 a few times in the above derivations. \square

Theorem B.6. *When two random variables \underline{x}_a and \underline{x}_b are independent, then*

$$\mathbb{E}[\underline{x}_a \underline{x}_b^T] = \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T, \quad (\text{B.29})$$

$$\mathbb{V}[\underline{x}_a, \underline{x}_b] = 0, \quad (\text{B.30})$$

$$\mathbb{V}[\underline{x}_a + \underline{x}_b] = \mathbb{V}[\underline{x}_a] + \mathbb{V}[\underline{x}_b]. \quad (\text{B.31})$$

Proof. This theorem contains three expressions which need to be proven. To derive the first, we should apply the definition (B.11) of independent random variables. We then find that

$$\begin{aligned} \mathbb{E}[\underline{x}_a \underline{x}_b^T] &= \int_{X_a} \int_{X_b} \underline{x}_a \underline{x}_b^T f_{\underline{x}_a, \underline{x}_b}(\underline{x}_a, \underline{x}_b) d\underline{x}_a d\underline{x}_b \\ &= \int_{X_a} \int_{X_b} \underline{x}_a \underline{x}_b^T f_{\underline{x}_a}(\underline{x}_a) f_{\underline{x}_b}(\underline{x}_b) d\underline{x}_a d\underline{x}_b \\ &= \left(\int_{X_a} \underline{x}_a f_{\underline{x}_a}(\underline{x}_a) d\underline{x}_a \right) \left(\int_{X_b} \underline{x}_b f_{\underline{x}_b}(\underline{x}_b) d\underline{x}_b \right)^T \\ &= \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T. \end{aligned} \quad (\text{B.32})$$

The second expression now directly follows from Theorem B.5. This theorem tells us that

$$\mathbb{V}[\underline{x}_a, \underline{x}_b] = \mathbb{E}[\underline{x}_a \underline{x}_b^T] - \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T = \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T - \mathbb{E}[\underline{x}_a] \mathbb{E}[\underline{x}_b]^T = 0. \quad (\text{B.33})$$

Next, it also directly follows from Theorem B.4 that

$$\mathbb{V}[\underline{x}_a + \underline{x}_b] = \mathbb{V}[\underline{x}_a] + \mathbb{V}[\underline{x}_b] + \mathbb{V}[\underline{x}_a, \underline{x}_b] + \mathbb{V}[\underline{x}_b, \underline{x}_a] = \mathbb{V}[\underline{x}_a] + \mathbb{V}[\underline{x}_b]. \quad (\text{B.34})$$

This completes the proof. \square

B.3. TRANSFORMATIONS OF PROBABILITY DENSITY FUNCTIONS

We know how probability density functions are defined. But when we transform a random variable \underline{x} , how will its probability density function $f_{\underline{x}}(\underline{x})$ be affected? That is what we will look at in this section. We start with linear transformations (Section B.3.1), continue with nonlinear transformations (Section B.3.2) and finally we look at how we can merge multiple distributions together (Section B.3.3).

B

B.3.1. LINEAR TRANSFORMATIONS OF A RANDOM VARIABLE

Suppose that we have a random variable \underline{x} . If we transform this to a new random variable $\underline{y} = P\underline{x} + \underline{c}$, what will the PDF of \underline{y} be?

Theorem B.7. Consider the random variable \underline{x} of size n , the invertible $n \times n$ matrix P and the (deterministic) vector \underline{c} of size n . If we define $\underline{y} = P\underline{x} + \underline{c}$, then

$$f_{\underline{y}}(\underline{y}) = \frac{1}{|P|} f_{\underline{x}}(P^{-1}(\underline{y} - \underline{c})). \quad (\text{B.35})$$

Proof. Let's denote the probability that \underline{x} falls within the n -dimensional rectangle bounded by \underline{x} and $\underline{x} + d\underline{x}$ as $p(\underline{x} \leq \underline{x} < \underline{x} + d\underline{x})$. This inequality does not hold from the strict point of view. (If $d\underline{x}$ has negative elements, the equality itself would not hold.) However, if we stick with this new notation, then we have

$$\begin{aligned} p(\underline{x} \leq \underline{x} < \underline{x} + d\underline{x}) &= p(P\underline{x} + \underline{c} \leq P\underline{x} + \underline{c} < P\underline{x} + Pd\underline{x} + \underline{c}) \\ &= p(\underline{y} \leq \underline{y} < \underline{y} + d\underline{y}), \end{aligned} \quad (\text{B.36})$$

where we have defined $\underline{y} = P\underline{x} + \underline{c}$ and $d\underline{y} = d(P\underline{x} + \underline{c}) = Pd\underline{x}$. Here, you should again read $p(\underline{y} \leq \underline{y} < \underline{y} + d\underline{y})$ as the probability that \underline{y} falls within the n -dimensional rectangle bounded by \underline{y} and $\underline{y} + d\underline{y}$, even though some elements of $d\underline{y}$ may be negative. It now follows that

$$\begin{aligned} f_{\underline{y}}(\underline{y}) &= \frac{p(\underline{y} \leq \underline{y} < \underline{y} + d\underline{y})}{d\underline{y}} \\ &= \frac{d\underline{x}}{d\underline{y}} \frac{p(\underline{x} \leq \underline{x} < \underline{x} + d\underline{x})}{d\underline{x}} \\ &= \frac{d\underline{x}}{d\underline{y}} f_{\underline{x}}(\underline{x}). \end{aligned} \quad (\text{B.37})$$

We should remember that, in this relation, dividing by $d\underline{y}$ actually means dividing by the volume of $d\underline{y}$, and similarly for $d\underline{x}$. As a result, the ratio $\frac{d\underline{x}}{d\underline{y}}$ is the ratio between the volumes of the blocks $d\underline{x}$ and $d\underline{y}$. It is known in mathematics that, when applying a transformation $\underline{y} = P\underline{x}$ to a certain space, then all volumes are increased by a factor $|P|$. And because P is (assumed) invertible, this determinant is nonzero. Hence, $\frac{d\underline{x}}{d\underline{y}} = \frac{1}{|P|}$. This results in

$$f_{\underline{y}}(\underline{y}) = \frac{1}{|P|} f_{\underline{x}}(\underline{x}). \quad (\text{B.38})$$

Because P is invertible, we can substitute \underline{x} by $P^{-1}(\underline{y} - \underline{c})$, which completes this proof. \square

B.3.2. NONLINEAR TRANSFORMATIONS OF RANDOM VARIABLES

Contrary to linear transformations, nonlinear transformations of random variables are very tricky. Suppose we have a random variable \underline{x} with PDF $f_{\underline{x}}(x)$, and we define $\underline{y} = g(\underline{x})$ for some function $g(\cdot)$. What is then the PDF of \underline{y} ?

There is no easy general solution for this problem. But for some specific functions we can find solutions.

B

Theorem B.8. Consider the scalar random variables $\underline{x} > 0$ and \underline{y} satisfying $\underline{y} = \log(\underline{x})$ and equivalently $\underline{x} = e^{\underline{y}}$. The PDFs of \underline{x} and \underline{y} are now related through

$$f_{\underline{x}}(x) = \frac{1}{e^y} f_{\underline{y}}(y), \quad (\text{B.39})$$

$$f_{\underline{y}}(y) = x f_{\underline{x}}(x). \quad (\text{B.40})$$

Proof. Per definition, the PDF of \underline{x} is given by

$$f_{\underline{x}}(x) = \frac{p(x \leq \underline{x} < x + dx)}{dx}. \quad (\text{B.41})$$

Let's define $y = \log(x)$, meaning that $x = e^y$. It follows that $dx = e^y dy$. We now have

$$f_{\underline{x}}(x) = \frac{p(e^y \leq e^{\underline{y}} < e^y + e^y dy)}{e^y dy} = \frac{p(e^y \leq e^{\underline{y}} < e^y(1 + dy))}{e^y dy}. \quad (\text{B.42})$$

Next, we will take the logarithm of every term within the probability function. Because the logarithm is a strictly ascending function, the inequality does not change. As a result, we have

$$f_{\underline{x}}(x) = \frac{p(y \leq \underline{y} < y + \log(1 + dy))}{e^y dy}. \quad (\text{B.43})$$

We can expand $\log(1 + dy)$ according to its Taylor polynomial

$$\log(1 + dy) = dy - \frac{dy^2}{2} + \frac{dy^3}{3} - \frac{dy^4}{4} + \dots \quad (\text{B.44})$$

In the limit of $dy \rightarrow 0$, only the first term is relevant, and so then $\log(1 + dy) = dy$. As a result, we have

$$f_{\underline{x}}(x) = \frac{1}{e^y} \frac{p(y \leq \underline{y} < y + dy)}{dy} = \frac{1}{e^y} f_{\underline{y}}(y). \quad (\text{B.45})$$

From the definition $x = e^y$ now also directly follows that

$$f_{\underline{y}}(y) = x f_{\underline{x}}(x). \quad (\text{B.46})$$

This completes the proof. \square

B.3.3. MERGING DISTRIBUTIONS

In practice, how do we find probability density functions? This usually happens through external data. For instance, suppose that we want to estimate some parameter \underline{x} . We do not know what it is yet, so we write it as a random variable \underline{x} .

Usually, we have some prior idea about what \underline{x} may be. We can summarize this in the *prior distribution* $f_{\underline{x}}^{pr}(\underline{x})$ of \underline{x} . In this distribution, we make probable values of \underline{x} more likely, and improbable values of \underline{x} less likely. And in the extreme case that we really know nothing about \underline{x} , we give it the null distribution from Section B.1.4.

Only having prior data is pretty useless, but we can improve on this by performing a measurement of \underline{x} . This measurement usually does not give us the precise value of \underline{x} , for instance because of measurement noise. However, what it does give us is another PDF, being the first measurement PDF $f_{\underline{x}}^1(\underline{x})$.

So basically, we now have two PDFs for \underline{x} and both are true. How do we merge them? There are various ways to do that based on how they have been obtained, as is also explained by [Clemen and Winkler \(2007\)](#). However, in this thesis we will use the following theorem, which corresponds to the intuitive view described in Section 2.1.3.

Theorem B.9. Suppose that we have two PDFs $f_{\underline{x}}^1(\underline{x})$ and $f_{\underline{x}}^2(\underline{x})$ for the same random variable \underline{x} . When these two PDFs have been obtained independently, then the posterior PDF of \underline{x} is given by

$$f_{\underline{x}}^{po}(\underline{x}) = \frac{f_{\underline{x}}^1(\underline{x}) f_{\underline{x}}^2(\underline{x})}{\int_X f_{\underline{x}}^1(\underline{x}') f_{\underline{x}}^2(\underline{x}') d\underline{x}'}. \quad (\text{B.47})$$

Proof. Let's denote the random variable corresponding to the first measurement by \underline{x}_a and the random variable corresponding to the second measurement by \underline{x}_b . We know that \underline{x}_a and \underline{x}_b are equal, since they both equal \underline{x} . Hence, the probability that \underline{x}_a equals some vector \underline{x} , given this data, is equal to

$$\begin{aligned} p(\underline{x} = \underline{x}_a | \underline{x}_a = \underline{x}_b) &= \frac{p(\underline{x} = \underline{x}_a, \underline{x}_a = \underline{x}_b)}{p(\underline{x}_a = \underline{x}_b)} \\ &= \frac{p(\underline{x} = \underline{x}_a = \underline{x}_b)}{\int_X p(\underline{x}' = \underline{x}_a = \underline{x}_b)} \\ &= \frac{p(\underline{x} = \underline{x}_a, \underline{x} = \underline{x}_b)}{\int_X p(\underline{x}' = \underline{x}_a, \underline{x}' = \underline{x}_b)}. \end{aligned} \quad (\text{B.48})$$

Because \underline{x}_a and \underline{x}_b have been obtained independently, we may split up the probability operators to get

$$p(\underline{x} = \underline{x}_a | \underline{x}_a = \underline{x}_b) = \frac{p(\underline{x} = \underline{x}_a) p(\underline{x} = \underline{x}_b)}{\int_X p(\underline{x}' = \underline{x}_a) p(\underline{x}' = \underline{x}_b)}. \quad (\text{B.49})$$

For infinitesimally small steps $d\underline{x}$ this can be rewritten to

$$p(\underline{x} \leq \underline{x}_a < \underline{x} + d\underline{x} | \underline{x}_a = \underline{x}_b) = \frac{p(\underline{x} \leq \underline{x}_a < \underline{x} + d\underline{x}) p(\underline{x} \leq \underline{x}_b < \underline{x} + d\underline{x})}{\int_X p(\underline{x}' \leq \underline{x}_a < \underline{x}' + d\underline{x}') p(\underline{x}' \leq \underline{x}_b < \underline{x}' + d\underline{x}')}. \quad (\text{B.50})$$

If we then also divide both sides of the equation by this step $d\mathbf{x}$ (or an equally large step $d\mathbf{x}'$), we find that

$$\frac{p(\underline{\mathbf{x}} \leq \underline{x}_a < \underline{x} + d\underline{\mathbf{x}} | \underline{x}_a = \underline{x}_b)}{d\underline{\mathbf{x}}} = \frac{\frac{p(\underline{\mathbf{x}} \leq \underline{x}_a < \underline{x} + d\underline{\mathbf{x}})}{d\underline{\mathbf{x}}} \frac{p(\underline{\mathbf{x}} \leq \underline{x}_b < \underline{x} + d\underline{\mathbf{x}})}{d\underline{\mathbf{x}}}}{\int_X \frac{p(\underline{\mathbf{x}}' \leq \underline{x}_a < \underline{x}' + d\underline{\mathbf{x}}')}{d\underline{\mathbf{x}}'} \frac{p(\underline{\mathbf{x}}' \leq \underline{x}_b < \underline{x}' + d\underline{\mathbf{x}}')}{d\underline{\mathbf{x}}'} d\underline{\mathbf{x}}'}, \quad (\text{B.51})$$

$$f_{\underline{x}_a | \underline{x}_a = \underline{x}_b}(\underline{\mathbf{x}}) = \frac{f_{\underline{x}_a}(\underline{\mathbf{x}}) f_{\underline{x}_b}(\underline{\mathbf{x}})}{\int_X f_{\underline{x}_a}(\underline{\mathbf{x}}') f_{\underline{x}_b}(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'}. \quad (\text{B.52})$$

This proves that, given two independently obtained PDFs $f_{\underline{x}}^1(\underline{\mathbf{x}})$ and $f_{\underline{x}}^2(\underline{\mathbf{x}})$ of the same random variable $\underline{\mathbf{x}}$, the posterior distribution of $\underline{\mathbf{x}}$ equals (B.47). \square

When we have multiple measurements, we can also just merge them in a similar way, as is shown by the next theorem.

Theorem B.10. Suppose that we have n PDFs $f_{\underline{x}}^1(\underline{\mathbf{x}}), \dots, f_{\underline{x}}^n(\underline{\mathbf{x}})$ for the same random variable $\underline{\mathbf{x}}$. When these PDFs have been obtained independently, then the posterior PDF of $\underline{\mathbf{x}}$ is given by

$$f_{\underline{x}}^{po}(\underline{\mathbf{x}}) = \frac{f_{\underline{x}}^1(\underline{\mathbf{x}}) \dots f_{\underline{x}}^n(\underline{\mathbf{x}})}{\int_X f_{\underline{x}}^1(\underline{\mathbf{x}}') \dots f_{\underline{x}}^n(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'}. \quad (\text{B.53})$$

Proof. This can be proven by mathematical induction. The case when $n = 1$ equals

$$f_{\underline{x}}^{po}(\underline{\mathbf{x}}) = \frac{f_{\underline{x}}^1(\underline{\mathbf{x}})}{\int_X f_{\underline{x}}^1(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'} = f_{\underline{x}}^1(\underline{\mathbf{x}}). \quad (\text{B.54})$$

This is obviously true. If we only have one PDF, then the posterior PDF equals that PDF.

Now assume that this theorem holds up to $n - 1$. We then have as posterior distribution $f_{\underline{x}}^{po,n-1}(\underline{\mathbf{x}})$ after $n - 1$ measurements

$$f_{\underline{x}}^{po,n-1}(\underline{\mathbf{x}}) = \frac{f_{\underline{x}}^1(\underline{\mathbf{x}}) \dots f_{\underline{x}}^{n-1}(\underline{\mathbf{x}})}{\int_X f_{\underline{x}}^1(\underline{\mathbf{x}}') \dots f_{\underline{x}}^{n-1}(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'}. \quad (\text{B.55})$$

If we add a measurement n , giving us another PDF $f_{\underline{x}}^n(\underline{\mathbf{x}})$, we ought to add this to the posterior distribution too. To do so, we merge the previous posterior distribution with our new measurement using Theorem B.9. This gives us the new posterior distribution

$$\begin{aligned} f_{\underline{x}}^{po,n}(\underline{\mathbf{x}}) &= \frac{f_{\underline{x}}^{po,n-1}(\underline{\mathbf{x}}) f_{\underline{x}}^n(\underline{\mathbf{x}})}{\int_X f_{\underline{x}}^{po,n-1}(\underline{\mathbf{x}}') f_{\underline{x}}^n(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'} \\ &= \frac{\frac{f_{\underline{x}}^1(\underline{\mathbf{x}}) \dots f_{\underline{x}}^{n-1}(\underline{\mathbf{x}})}{\int_X f_{\underline{x}}^1(\underline{\mathbf{x}}') \dots f_{\underline{x}}^{n-1}(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'} f_{\underline{x}}^n(\underline{\mathbf{x}})}{\int_X \frac{f_{\underline{x}}^1(\underline{\mathbf{x}}') \dots f_{\underline{x}}^{n-1}(\underline{\mathbf{x}}')}{\int_X f_{\underline{x}}^1(\underline{\mathbf{x}}') \dots f_{\underline{x}}^{n-1}(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'} f_{\underline{x}}^n(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'}. \end{aligned} \quad (\text{B.56})$$

We should note here that the integral $\int_X f_{\underline{x}}^1(\underline{\mathbf{x}}') \dots f_{\underline{x}}^{n-1}(\underline{\mathbf{x}}') d\underline{\mathbf{x}}'$ is a constant. That is, it does not depend on $\underline{\mathbf{x}}$. Because it appears both in the numerator and the denominator,

it cancels out. We remain with

$$f_{\underline{x}}^{po,n}(\mathbf{x}) = \frac{f_{\underline{x}}^1(\mathbf{x}) \dots f_{\underline{x}}^{n-1}(\mathbf{x}) f_{\underline{x}}^n(\mathbf{x})}{\int_X f_{\underline{x}}^1(\mathbf{x}'') \dots f_{\underline{x}}^{n-1}(\mathbf{x}'') f_{\underline{x}}^n(\mathbf{x}'') d\mathbf{x}''}. \quad (\text{B.57})$$

B

So, assuming that the theorem holds for $n-1$, it also holds for n . Through induction, this completes the proof. \square

We will use this idea of merging distributions a lot in this thesis, which is why we introduce a special notation for it. (Also see Section 2.1.3.) We use the operator \oplus and say that the above is (per definition) equivalent to

$$f_{\underline{x}}^{po,n}(\mathbf{x}) = f_{\underline{x}}^1(\mathbf{x}) \oplus f_{\underline{x}}^2(\mathbf{x}) \oplus \dots \oplus f_{\underline{x}}^n(\mathbf{x}). \quad (\text{B.58})$$

It is worthwhile to note that, just like with regular addition and multiplication, the order in which we merge distributions has no effect whatsoever.

You may be wondering, ‘What do we do if one measurement is more accurate than another? Shouldn’t we take that into account in some way?’ The answer to this is, ‘We already do.’

If some measurement i is very uncertain, then its PDF $f_{\underline{x}}^i(\mathbf{x})$ will be very spread out, with as ultimate situation the null distribution $f_{\underline{x}}^i(\mathbf{x}) = \gamma$ from Section B.1.4. Such a distribution does not have any effect on the posterior distribution $f_{\underline{x}}^{po}(\mathbf{x})$ and might as well be ignored.

Alternatively, if some measurement j is highly certain, then its PDF $f_{\underline{x}}^j(\mathbf{x})$ will have a strong peak, with as ultimate situation the delta PDF $f_{\underline{x}}^j(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{c})$. Such a distribution has a deciding effect on the posterior distribution $f_{\underline{x}}^{po}(\mathbf{x})$, making it also equal to $\delta(\mathbf{x} - \mathbf{c})$, irrespective of other measurements.

A final interesting result is that, if $f_{\underline{x}}^1(\mathbf{x}), \dots, f_{\underline{x}}^n(\mathbf{x})$ are all the same PDFs, then the posterior distribution $f_{\underline{x}}^{po}(\mathbf{x})$ generally does *not* equal this PDF. Instead, it is more peaked. This may initially seem counterintuitive, but it actually does make sense. To see how, we can consider an example. Suppose that one measurement tells us that some random variable \underline{x} probably has value 1, but maybe has value 2. At this point we are not very convinced yet about the value of \underline{x} . But if ten independent measurements all tell us the same thing, we can be quite sure that \underline{x} indeed is very likely to have value 1 and very unlikely to have value 2. Naturally, things may be different when the measurements are not independent, but that is a complicated case which we will not consider in this thesis.

B.4. THE GAUSSIAN DISTRIBUTION

In this section we will examine the (arguably) most well-known distribution, being the Gaussian distribution. We start by looking at its probability density function (Section B.4.1) and the special case of the standard Gaussian distribution (Section B.4.2). We then look at how the distribution changes when we apply linear transformations to it (Section B.4.3) or when we obtain more data (Section B.4.4). We continue by looking at a few special cases of the Gaussian distribution, including what happens when the covariance matrix is not positive definite anymore (Section B.4.5). Finally, we study the distribution of power forms of Gaussian random variables (Section B.4.6).

B.4.1. THE GAUSSIAN PROBABILITY DENSITY FUNCTION

A random variable \underline{x} has a *Gaussian distribution*, also known as a *normal distribution*, if its PDF equals the *Gaussian probability density function*

$$f_{\underline{x}}(\underline{x}) = \mathcal{N}(\underline{x}|\boldsymbol{\mu}, \Sigma) \equiv \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\underline{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\underline{x}-\boldsymbol{\mu})\right). \quad (\text{B.59})$$

B

In this expression, $\boldsymbol{\mu}$ is the *mean vector* and Σ is the *covariance matrix*. Once these are known, the PDF of the Gaussian random variable is fully defined.

To indicate that a random variable \underline{x} has a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance Σ , we usually write

$$\underline{x} \sim \mathcal{N}(\underline{x}|\boldsymbol{\mu}, \Sigma). \quad (\text{B.60})$$

The symbol ‘~’ here can be read as ‘has as probability density function’ or ‘is distributed according to’. In addition, when it is clear or when it doesn’t matter which parameter \underline{x} is inserted in the Gaussian PDF, we may also simply omit it and write

$$\underline{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma). \quad (\text{B.61})$$

It is interesting to note that the Gaussian distribution is a probability density function with mean $\boldsymbol{\mu}$ and covariance Σ . As such, we must have

$$1 = \int_{-\infty}^{\infty} \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\underline{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\underline{x}-\boldsymbol{\mu})\right) d\underline{x}, \quad (\text{B.62})$$

$$\boldsymbol{\mu} = \int_{-\infty}^{\infty} \underline{x} \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\underline{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\underline{x}-\boldsymbol{\mu})\right) d\underline{x}, \quad (\text{B.63})$$

$$\Sigma = \int_{-\infty}^{\infty} (\underline{x}-\boldsymbol{\mu})(\underline{x}-\boldsymbol{\mu})^T \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\underline{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\underline{x}-\boldsymbol{\mu})\right) d\underline{x}. \quad (\text{B.64})$$

These relations will be useful when we examine Gaussian exponentials more closely.

B.4.2. THE STANDARD GAUSSIAN DISTRIBUTION

The *standard Gaussian distribution* is the scalar Gaussian distribution with zero mean and unit variance. That is, \underline{x} has a standard Gaussian distribution if $\underline{x} \sim \mathcal{N}(x|0, 1)$. The PDF of such a distribution, called the *standard probability density function*, is often written as

$$\phi(x) \equiv \mathcal{N}(x|0, 1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right). \quad (\text{B.65})$$

There is also a higher-dimensional equivalent, which equals

$$\phi(\underline{x}) \equiv \mathcal{N}(\underline{x}|\mathbf{0}, I) = \frac{1}{\sqrt{(2\pi)^n}} \exp\left(-\frac{1}{2}\underline{x}^T \underline{x}\right), \quad (\text{B.66})$$

with n the size of the vector \underline{x} .

It is possible to express any Gaussian PDF in that of the standard PDF. To do so, we can use the following theorem.

Theorem B.11. We can express any Gaussian exponential function $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ in the standard PDF through

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{|\Sigma|}} \phi(L^{-1}(\mathbf{x} - \boldsymbol{\mu})), \quad (\text{B.67})$$

where L is the (lower-triangular) Cholesky decomposition of Σ .

Proof. We know from our linear algebra course that we can write any positive definite matrix Σ as LL^T , where L is the lower triangular matrix known as the *Cholesky decomposition* of Σ . In this case, we can rewrite $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ as

$$\begin{aligned} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) &= \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \\ &= \frac{1}{\sqrt{|2\pi I||\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T L^{-T}L^{-1}(\mathbf{x} - \boldsymbol{\mu})\right). \end{aligned} \quad (\text{B.68})$$

If we now define $\mathbf{y} = L^{-1}(\mathbf{x} - \boldsymbol{\mu})$, then this turns into

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{|\Sigma|}} \frac{1}{\sqrt{|2\pi I|}} \exp\left(-\frac{1}{2}\mathbf{y}^T I^{-1}\mathbf{y}\right) = \frac{1}{\sqrt{|\Sigma|}} \mathcal{N}(\mathbf{y}|\mathbf{0}, I), \quad (\text{B.69})$$

which is what we needed to prove. \square

In the above theorem, we always have $\sqrt{|\Sigma|} = \sqrt{|L||L^T|} = |L|$. Additionally, in the scalar case we have $\Sigma = \sigma^2 = L^2$, with σ being the standard deviation of \underline{x} .

When we integrate the PDF of the standard Gaussian distribution, we find the *Cumulative Density Function* (CDF) of the standard Gaussian distribution. It is defined as

$$\Phi(x) \equiv \int_{-\infty}^x \phi(x') dx', \quad (\text{B.70})$$

or for higher-dimensional functions as

$$\Phi(\mathbf{x}) \equiv \int_{-\infty}^{x_1} \dots \int_{-\infty}^{x_n} \phi(\mathbf{x}') dx'_n \dots dx'_1 = \int_{-\infty}^{\mathbf{0}} \phi(\mathbf{x}') d\mathbf{x}'. \quad (\text{B.71})$$

When we integrate a non-standard Gaussian distribution, we can use the following theorem to compute the result.

Theorem B.12. The integral of the Gaussian exponential function $\underline{x} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ over the region X equals

$$\int_X \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) d\mathbf{x} = \int_Y \phi(\mathbf{y}) d\mathbf{y}, \quad (\text{B.72})$$

where $\mathbf{y} \equiv L^{-1}(\mathbf{x} - \boldsymbol{\mu})$ and Y is the region resulting from applying this transformation to all points within X . In addition, L is the (lower-triangular) Cholesky decomposition of Σ .

Proof. To prove this, we will use Theorem B.11. This theorem expands the integrand into

$$\int_X \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) d\mathbf{x} = \int_X \frac{1}{\sqrt{|\Sigma|}} \phi(L^{-1}(\mathbf{x}' - \boldsymbol{\mu})) d\mathbf{x}. \quad (\text{B.73})$$

We will now substitute \mathbf{x} as integration parameter for $\mathbf{y} = L^{-1}(\mathbf{x} - \boldsymbol{\mu})$. Because L is a triangular matrix, we have

$$d\mathbf{x} = dx_1 dx_2 \dots dx_n = L_{11} dy_1 L_{22} dy_2 \dots L_{nn} dy_n = |L| d\mathbf{y}. \quad (\text{B.74})$$

Also note that $|L| = \sqrt{|\Sigma|}$. If we then also adjust the integration area accordingly, replacing X by Y , we directly find (B.72). \square

The hard part when applying the above theorem is that, even if X is a nice rectangular reason, the region Y does not have to be rectangular, making the resulting integral hard to solve.

If we however can solve the integral and find the corresponding CDF, then this will be very useful for calculating probabilities. Suppose that $\underline{\mathbf{x}}$ has a standard Gaussian distribution. In that case, we directly have

$$p(\mathbf{a} \leq \underline{\mathbf{x}} \leq \mathbf{b}) = \int_{\mathbf{a}}^{\mathbf{b}} \phi(\mathbf{x}) \mathbf{x} = \Phi(\mathbf{b}) - \Phi(\mathbf{a}), \quad (\text{B.75})$$

where the inequality must hold for every element within the vector. A similar trick can be performed for non-standard Gaussian random variables, if the corresponding CDF can be found.

B.4.3. LINEAR TRANSFORMATIONS OF GAUSSIAN DISTRIBUTIONS

Gaussian random variables are useful for many reasons. First of all, we only need two parameters ($\boldsymbol{\mu}$ and Σ) to fully specify an entire distribution. Secondly, linear combinations of Gaussian random variables are also Gaussian random variables. The following theorem shows us why, as well as what the parameters of these distributions will be.

Theorem B.13. Consider a Gaussian random variable $\underline{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ of size n and an $n \times n$ invertible matrix P . The random variable $\mathbf{y} = P\underline{\mathbf{x}} + \mathbf{c}$ now has the distribution

$$\mathbf{y} = P\underline{\mathbf{x}} + \mathbf{c} \sim \mathcal{N}(\mathbf{y}|P\boldsymbol{\mu} + \mathbf{c}, P\Sigma P^T). \quad (\text{B.76})$$

Proof. To prove this theorem, we look at the PDF of $\underline{\mathbf{y}}$. From Theorem B.7 we know that

$$f_{\underline{\mathbf{y}}}(\mathbf{y}) = \frac{1}{|P|} f_{\underline{\mathbf{x}}}(\mathbf{y} - P\mathbf{c}). \quad (\text{B.77})$$

If we use the definition of the Gaussian PDF (B.59), as well as apply the matrix determinant rules $|P| = |P^T|$ and $|A||B| = |AB|$, we find that

$$\begin{aligned} f_{\underline{\mathbf{y}}}(\mathbf{y}) &= \frac{1}{|P|} \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(P^{-1}(\mathbf{y} - \mathbf{c}) - \boldsymbol{\mu})^T \Sigma^{-1} (P^{-1}(\mathbf{y} - \mathbf{c}) - \boldsymbol{\mu})\right) \\ &= \frac{1}{\sqrt{|P| |2\pi\Sigma| |P^T|}} \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{c} - P\boldsymbol{\mu})^T P^{-T} \Sigma^{-1} P^{-1} (\mathbf{y} - \mathbf{c} - P\boldsymbol{\mu})\right) \\ &= \frac{1}{\sqrt{|2\pi P\Sigma P^T|}} \exp\left(-\frac{1}{2}(\mathbf{y} - (P\boldsymbol{\mu} + \mathbf{c}))^T (P\Sigma P^T)^{-1} (\mathbf{y} - (P\boldsymbol{\mu} + \mathbf{c}))\right) \\ &= \mathcal{N}(\mathbf{y}|P\boldsymbol{\mu} + \mathbf{c}, P\Sigma P^T). \end{aligned} \quad (\text{B.78})$$

This not only proves that \underline{y} is Gaussian, but also that it has mean $P\mu + c$ and covariance matrix $P\Sigma P^T$, completing the proof. \square

It may be interesting to know (though we will not prove this here) that the above theorem also holds for non-invertible matrices P . In that case we will wind up with a covariance matrix $P\Sigma P^T$ which is not positive definite. Later on, in Section B.4.5, we will look at what exactly this means for the distribution.

B.4.4. MARGINALIZATION AND CONDITIONING OF GAUSSIAN DISTRIBUTIONS

In Section B.1.2 we looked at marginalization of distributions, and in Section B.1.3 we examined conditional distributions. We can apply these ideas to Gaussian distributions too.

Suppose that we have two Gaussian random variables \underline{x}_a and \underline{x}_b . A priori (before doing any measurements) we know that they have a joint distribution

$$\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}\right). \quad (\text{B.79})$$

When we talk about prior distributions, we often use \mathbf{m} to denote the mean and K the covariance, while μ and Σ are used to indicate the posterior distribution (after incorporating measurements). This is a useful distinction to keep in mind.

If we apply marginalization to this, we find the following theorem.

Theorem B.14. *When two Gaussian random variables are distributed according to (B.79), then we have $\underline{x}_a \sim \mathcal{N}(\underline{x}_a | \mathbf{m}_a, K_{aa})$ and $\underline{x}_b \sim \mathcal{N}(\underline{x}_b | \mathbf{m}_b, K_{bb})$.*

Proof. We can find the PDF of \underline{x}_a by applying marginalization (Theorem B.1). This means that

$$f_{\underline{x}_a}(\underline{x}_a) = \int_{X_b} f_{\underline{x}_a, \underline{x}_b}(\underline{x}_a, \underline{x}_b) d\underline{x}_b = \int_{X_b} \mathcal{N}\left(\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}\right) d\underline{x}_b. \quad (\text{B.80})$$

If we now apply Theorem A.16, then it directly follows that $\underline{x}_a \sim \mathcal{N}(\underline{x}_a | \mathbf{m}_a, K_{aa})$. And identically, it follows that $\underline{x}_b \sim \mathcal{N}(\underline{x}_b | \mathbf{m}_b, K_{bb})$. \square

We can also find the conditional distribution of Gaussian variables. This follows from the next theorem.

Theorem B.15. *When two Gaussian random variables are distributed according to (B.79), then the conditional distribution of \underline{x}_b , given that $\underline{x}_a = \hat{\underline{x}}_a$, equals*

$$f_{\underline{x}_b | \hat{\underline{x}}_a}(\underline{x}_b) = \mathcal{N}(\underline{x}_b | \mu_b, \Sigma_{bb}) = \mathcal{N}(\underline{x}_b | \mathbf{m}_b + K_{ba}K_{aa}^{-1}(\hat{\underline{x}}_a - \mathbf{m}_a), K_{bb} - K_{ba}K_{aa}^{-1}K_{ab}). \quad (\text{B.81})$$

Proof. We can find the conditional distribution of $\underline{x}_b | \underline{x}_a = \hat{\underline{x}}_a$ through Theorem B.2. It tells us that

$$f_{\underline{x}_b | \hat{\underline{x}}_a}(\underline{x}_b) = \frac{f_{\underline{x}_a, \underline{x}_b}(\hat{\underline{x}}_a, \underline{x}_b)}{f_{\underline{x}_a}(\hat{\underline{x}}_a)} = \frac{\mathcal{N}\left(\begin{bmatrix} \hat{\underline{x}}_a \\ \underline{x}_b \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}\right)}{\mathcal{N}(\hat{\underline{x}}_a | \mathbf{m}_a, K_{bb})}. \quad (\text{B.82})$$

Note that we have applied Theorem B.14 here, which told us that $f_{\hat{\mathbf{x}}_a}(\hat{\mathbf{x}}_a) = \mathcal{N}(\hat{\mathbf{x}}_a | \mathbf{m}_a, K_{aa})$. Next, we only have to apply Theorem A.15. If we do, and work out the results, we immediately find (B.81). \square

The previous theorem is useful when we have measured the value of $\underline{\mathbf{x}}_a$ in an exact way. That is, we are fully confident it equals $\hat{\mathbf{x}}_a$. And because $\hat{\mathbf{x}}_a$ is known precisely, we do not even have to write it as a random variable.

But often there is some measurement noise $\underline{\mathbf{v}}$ added to our measurement. Here, we assume that $\underline{\mathbf{v}}$ has a zero-mean Gaussian distribution $\underline{\mathbf{v}} \sim \mathcal{N}(\underline{\mathbf{v}} | \mathbf{0}, \hat{\Sigma})$. So we have

$$\hat{\mathbf{x}}_a = \underline{\mathbf{x}}_a + \underline{\mathbf{v}}. \quad (\text{B.83})$$

Note that in this case $\hat{\mathbf{x}}_a$ actually is a random variable with a certain distribution. When we take a measurement, we take a sample from this distribution. This measurement is an actual number, and so we denote the measurement by $\hat{\mathbf{x}}_a$.

The question now is, what is the posterior distribution of $\underline{\mathbf{x}}_b$, after we have performed our measurement?

Theorem B.16. *When two Gaussian random variables are distributed according to (B.79), then the conditional distribution of $\underline{\mathbf{x}}_b$, given a noisy measurement $\hat{\mathbf{x}}_a$ sampled from (B.83), equals*

$$f_{\underline{\mathbf{x}}_b | \hat{\mathbf{x}}_a}(\underline{\mathbf{x}}_b) = \mathcal{N}\left(\underline{\mathbf{x}}_b | \mathbf{m}_b + K_{ba}(K_{aa} + \hat{\Sigma})^{-1}(\hat{\mathbf{x}}_a - \mathbf{m}_a), K_{bb} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ab}\right). \quad (\text{B.84})$$

Proof. There are many ways in which we can prove this. Later on in Theorem B.22 we will see a more powerful version of this theorem, with a prove which I personally find more intuitive. (It involves merging distributions.) But here we will stick with the proof given in most textbooks.

Consider (B.83). We will use Theorem B.4 to find the prior mean and covariance of $\hat{\mathbf{x}}_a$. Here, we should realize that $\underline{\mathbf{x}}_a$ and $\underline{\mathbf{v}}$ are independent (and hence uncorrelated). Prior to doing any measurement, we hence have

$$\hat{\mathbf{x}}_a = \underline{\mathbf{x}}_a + \underline{\mathbf{v}} \sim \mathcal{N}(\hat{\mathbf{x}}_a | \mathbf{m}_a + \mathbf{0}, K_{aa} + \hat{\Sigma}). \quad (\text{B.85})$$

We can expand this vector by adding $\underline{\mathbf{x}}_b$. If we once more apply Theorem B.4 to calculate the mean and covariance, noting that $\underline{\mathbf{v}}$ is also independent from $\underline{\mathbf{x}}_b$, then we find that

$$\begin{bmatrix} \hat{\mathbf{x}}_a \\ \underline{\mathbf{x}}_b \end{bmatrix} = \begin{bmatrix} \underline{\mathbf{x}}_a \\ \underline{\mathbf{x}}_b \end{bmatrix} + \begin{bmatrix} \underline{\mathbf{v}} \\ \mathbf{0} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \hat{\mathbf{x}}_a \\ \underline{\mathbf{x}}_b \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{aa} + \hat{\Sigma} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}\right). \quad (\text{B.86})$$

The key now is that, although we did not measure $\underline{\mathbf{x}}_a$ exactly, we do have an exact measurement (sample) $\hat{\mathbf{x}}_a$ from $\hat{\mathbf{x}}_a$. Hence, we can apply Theorem B.15 to find that

$$\underline{\mathbf{x}}_b | \hat{\mathbf{x}}_a = \hat{\mathbf{x}}_a \sim \mathcal{N}\left(\underline{\mathbf{x}}_b | \mathbf{m}_b + K_{ba}(K_{aa} + \hat{\Sigma})^{-1}(\hat{\mathbf{x}}_a - \mathbf{m}_a), K_{bb} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ab}\right). \quad (\text{B.87})$$

This proves (B.84). And it is interesting to note that, for $\hat{\Sigma} = 0$, this theorem reduces back to Theorem B.15. \square

B.4.5. SPECIAL CASES OF THE GAUSSIAN DISTRIBUTION

In most practical applications, the covariance matrix Σ is finite and positive definite. That is, all its eigenvalues are finite and larger than zero. But we can also express the special PDFs of Section B.1.4 using Gaussian distributions, and then we get a few interesting cases.

For instance, the null distribution $f_{\underline{x}}(\underline{x}) = \gamma$, for a fully unknown parameter \underline{x} , can also be described through a Gaussian random variable with infinite variance. The covariance matrix hence equals $\Sigma = \infty I$, which we often shorten to ∞ . That is, all its eigenvalues are infinite. If this is the case, then the value of the mean μ does not matter anymore. We write this as $\underline{x} \sim \mathcal{N}(\underline{x}|*, \infty)$, where the star $*$ denotes an immaterial value. If we now insert the infinite covariance matrix into the Gaussian PDF (B.59), we find that the exponent becomes 0 and hence $\exp(\dots) = 1$. However, with $|\Sigma| = \infty$, we still have a PDF with a near-zero value γ .

Similarly, the delta distribution $f_{\underline{x}}(\underline{x}) = \delta(\underline{x} - \underline{c})$, for a fully known parameter $\underline{x} = \underline{c}$ (officially called a degenerate distribution) can be described by a Gaussian random variable with zero variance. The covariance matrix hence equals $\Sigma = 0I$, which we often shorten to 0. That is, all its eigenvalues are zero. We write this as $\underline{x} \sim \mathcal{N}(\underline{x}|\underline{c}, 0)$. If we insert this into (B.59), the exponent becomes minus infinity, and we hence have $\exp(-\infty) = 0$, with the only exception when $\underline{x} = \underline{c}$. In this case, the exponential does have a finite value. And because $|\Sigma| = 0$, we wind up with a PDF equal to $\delta(\underline{x} - \underline{c})$.

The interesting thing is that combinations of these distributions are also possible. For instance, if $\Sigma = \text{diag}(\infty, 1, 0)$, then \underline{x}_1 is fully unknown, \underline{x}_2 has a variance of 1 and is hence a ‘regular’ random variable, and \underline{x}_3 is fully certain and therefore deterministic.

But what do we do when Σ is singular, but not diagonal? In that case, the following theorem will provide some interesting insights.

Theorem B.17. *When a random variable \underline{x} has a Gaussian distribution $\underline{x} \sim \mathcal{N}(\mu, \Sigma)$, in which the covariance matrix Σ is symmetric and positive semidefinite, then the random vector $\underline{x} - \mu$ cannot have a value contained in the null space of Σ .*

Proof. Because Σ is real and symmetric, it follows that it is diagonalizable by an orthogonal matrix P . To be precise, P is a matrix whose columns equal the eigenvectors (of unit length) of Σ , it satisfies $P^{-1} = P^T$, and we have $\Sigma = PDP^T$, where D is a diagonal matrix consisting of the corresponding eigenvalues $\lambda_1, \dots, \lambda_n$ of Σ along its diagonal. Without loss of generality, we can assume that these eigenvalues are ordered in decreasing order. So, $\lambda_1 \geq \dots \geq \lambda_n$. Our assumption that Σ is positive semidefinite implies that $\lambda_n \geq 0$.

Now define the parameter $\underline{y} = P^T(\underline{x} - \mu)$. It follows from Theorem B.13 that \underline{y} is distributed according to

$$\underline{y} \sim \mathcal{N}(\underline{y}|\mathbf{0}, P^T \Sigma P) = \mathcal{N}(\underline{y}|\mathbf{0}, D). \quad (\text{B.88})$$

Because D is a diagonal matrix, it is relatively easy to imagine what the distribution of each individual parameter \underline{y}_i looks like. For example, if λ_1 (and possibly λ_2, λ_3 , and so on) equals infinity, then \underline{y}_1 is fully uncertain and can be any value with equal probability. Similarly, if λ_n (and possibly $\lambda_{n-1}, \lambda_{n-2}$, and so on) equals zero, then \underline{y}_n is deterministic and has value 0.

So we see that, when Σ has an eigenvalue λ equal to zero, with multiplicity k , then the bottom k elements of \underline{y} are equal to zero. As a result, $(\underline{x} - \boldsymbol{\mu}) = P\underline{y}$ is constrained to the span of all eigenvectors associated with eigenvalues that are not zero. The null space of Σ (which is the span of all eigenvectors associated with eigenvalues that *are* zero) can therefore not be reached by $\underline{x} - \boldsymbol{\mu}$. \square

B

Conventionally Σ is symmetric and positive definite. The above proof has shown us that Σ can also be positive semidefinite. In that case, it has one or more eigenvalues equal to zero, causing the random variable \underline{x} to have a deterministic part. That is, \underline{x} is constrained to a certain subspace.

Finally you may wonder, is it also possible for Σ to have an eigenvalue lower than zero? The answer is ‘Not in the conventional way.’ After all, if a scalar random variable \underline{x} has a distribution $\underline{x} \sim \mathcal{N}(x|0, -1)$, it would mean that $\mathbb{E}[\underline{x}^2] = -1$. This is only possible when \underline{x} takes complex values, although then we would enter a whole new realm of mathematics. (When using complex random variables $\underline{x} = \underline{a} + \underline{b}i$, it is usually easier to define a joint distribution for \underline{a} and \underline{b} , although the theory behind that is outside the scope of this appendix.)

Later on however, right after Theorem B.23, we will see that the idea of a negative (semi-)definite covariance matrix *can* be useful in some cases. In fact, while positive (semi-)definite covariance matrices add information about \underline{x} , negative (semi-)definite matrices subtract information (or add disinformation) about \underline{x} .

B.4.6. POWER FORMS OF GAUSSIAN RANDOM VARIABLES

Suppose that $\underline{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is a Gaussian random variable. A *power form* of \underline{x} is a polynomial in the elements of \underline{x} . An example is $\underline{x}^T P \underline{x}$, for some matrix P , which is a *quadratic power form*. What can we say about the properties of such power forms?

The first thing that we can notice is that these power forms are *not* Gaussian. As a counterexample, let’s define $\underline{x} \sim \mathcal{N}(0, 1)$ and consider \underline{x}^2 . We now obviously have $\underline{x}^2 \geq 0$, meaning that $p(\underline{x}^2 < 0) = 0$. However, any Gaussian distribution with finite variance can be negative. Hence, \underline{x}^2 cannot be Gaussian.

So what distribution is it then? That depends. A squared Gaussian parameter with zero mean and unit variance (or a sum of such parameters) results in a *χ -squared distribution*. In case we use a nonzero mean, we get a *noncentral χ -squared distribution*, and if we also use other covariance matrices, we wind up with a *generalized noncentral χ -squared distribution*. (For further details, see Muirhead (2005).) This is a distribution which does not have an analytic expression for its PDF.

While there is no analytic expression for the PDF of quadratic power forms of Gaussian variables, the distribution at least has a name. The same cannot be said for higher powers of Gaussian variables though. In this case, the best we can do is find the mean of the resulting variables. We will do that here, starting with the mean of the quadratic power forms.

While doing so, we make use of the *expected squared value*

$$\Psi \equiv \mathbb{E} [\underline{x}\underline{x}^T]. \quad (\text{B.89})$$

For a Gaussian random variable $\underline{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ it also equals

$$\Psi = \Sigma + \boldsymbol{\mu}\boldsymbol{\mu}^T. \quad (\text{B.90})$$

When examining power forms, this parameter Ψ is often more useful than the covariance matrix Σ , as we can already see when looking at the next theorem.

Theorem B.18. *For a Gaussian random variable $\underline{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and any matrix P it holds that*

$$\mathbb{E}[\underline{x}^T P \underline{x}] = \text{tr}(\Sigma P) + \boldsymbol{\mu}^T P \boldsymbol{\mu} = \text{tr}(\Psi P). \quad (\text{B.91})$$

Proof. To prove this, we need the trace operator. Both the trace operator and the expectation operator are linear operators, and so we may interchange the order in which they are applied. If we also use the cyclic property of the trace operator (see Theorem A.1) we find that

$$\mathbb{E}[\underline{x}^T P \underline{x}] = \mathbb{E}[\text{tr}(\underline{x}^T P \underline{x})] = \text{tr}(\mathbb{E}[\underline{x} \underline{x}^T P]) = \text{tr}(\Psi P) = \text{tr}(\Sigma P) + \boldsymbol{\mu}^T P \boldsymbol{\mu}. \quad (\text{B.92})$$

This concludes our proof. \square

Next, we will look at the mean of quartic power forms of \underline{x} .

Theorem B.19. *For a Gaussian random variable $\underline{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and any matrices P and Q it holds that*

$$\mathbb{E}[\underline{x}^T P \underline{x} \underline{x}^T Q \underline{x}] = \text{tr}(\Psi P) \text{tr}(\Psi Q) + 2\text{tr}(\Psi P \Psi Q) - 2\boldsymbol{\mu}^T P \boldsymbol{\mu} \boldsymbol{\mu}^T Q \boldsymbol{\mu}. \quad (\text{B.93})$$

Proof. We are going to start halfway into our proof, by introducing a theorem given by Kendrick (1981), Appendix F. Here, the theorem is stated and proven that, for a zero mean process $\underline{y} = \underline{x} - \boldsymbol{\mu}$, with covariance matrix Σ , we have

$$\mathbb{E}[\underline{y}^T P \underline{y} \underline{y}^T Q \underline{y}] = \text{tr}(\Sigma P) \text{tr}(\Sigma Q) + 2\text{tr}(\Sigma P \Sigma Q). \quad (\text{B.94})$$

However, we want to know what the above expression is for \underline{x} . So we write

$$\mathbb{E}[\underline{x}^T P \underline{x} \underline{x}^T Q \underline{x}] = \mathbb{E}[(\underline{y} + \boldsymbol{\mu})^T P (\underline{y} + \boldsymbol{\mu})(\underline{y} + \boldsymbol{\mu})^T Q (\underline{y} + \boldsymbol{\mu})]. \quad (\text{B.95})$$

If we work out all the brackets, we will get sixteen terms. However, \underline{y} is a zero-mean Gaussian, so any term which has either one or three times ' \underline{y} ' in it will have an expectation of zero. This causes eight terms to drop out.

For the remaining eight terms, we can find out that some of them are equal. For instance, $(\underline{y}^T P \boldsymbol{\mu})(\underline{y}^T Q \boldsymbol{\mu})$ equals $(\underline{y}^T P \boldsymbol{\mu})(\boldsymbol{\mu}^T Q \underline{y})$. Here, we have transposed the right half, which is allowed because it is a scalar. We could have also transposed the left half, finding something else which is equal.

As a result, we can write

$$\begin{aligned} \mathbb{E}[\underline{x}^T P \underline{x} \underline{x}^T Q \underline{x}] &= \mathbb{E}[\underline{y}^T P \underline{y} \underline{y}^T Q \underline{y} + \underline{y}^T P \underline{y} \boldsymbol{\mu}^T Q \boldsymbol{\mu} + \boldsymbol{\mu}^T P \boldsymbol{\mu} \underline{y}^T Q \underline{y} + \boldsymbol{\mu}^T P \boldsymbol{\mu} \boldsymbol{\mu}^T Q \boldsymbol{\mu} \\ &\quad + 2\boldsymbol{\mu}^T P \underline{y} \underline{y}^T Q \boldsymbol{\mu} + 2\underline{y}^T P \boldsymbol{\mu} \boldsymbol{\mu}^T Q \underline{y}]. \end{aligned} \quad (\text{B.96})$$

Now we will introduce the trace operator. We can take the trace of an entire term (which is a scalar), like ‘ $\text{tr}(\underline{\mathbf{y}}^T P \underline{\mathbf{y}} \mu^T Q \mu)$ ’, or only of half of a term (which is also a scalar) like ‘ $\text{tr}(\underline{\mathbf{y}}^T P \underline{\mathbf{y}}) \text{tr}(\mu^T Q \mu)$ ’. One of the results we could get is

$$\begin{aligned}\mathbb{E}[\underline{\mathbf{x}}^T P \underline{\mathbf{x}} \underline{\mathbf{x}}^T Q \underline{\mathbf{x}}] &= \mathbb{E}[\underline{\mathbf{y}}^T P \underline{\mathbf{y}} \underline{\mathbf{y}}^T Q \underline{\mathbf{y}} + \text{tr}(\underline{\mathbf{y}}^T P \underline{\mathbf{y}}) \text{tr}(\mu^T Q \mu) + \text{tr}(\mu^T P \mu) \text{tr}(\underline{\mathbf{y}}^T Q \underline{\mathbf{y}}) \\ &\quad + \text{tr}(\mu^T P \mu) \text{tr}(\mu^T Q \mu) + 2\text{tr}(\mu^T P \underline{\mathbf{y}} \underline{\mathbf{y}}^T Q \mu) + 2\text{tr}(\underline{\mathbf{y}}^T P \mu \mu^T Q \underline{\mathbf{y}})].\end{aligned}\quad (\text{B.97})$$

Next, we are going to apply (B.94). Simultaneously, we are also going to work out the expectation operator, using $\mathbb{E}[\underline{\mathbf{y}} \underline{\mathbf{y}}^T] = \Sigma$. We then get

$$\begin{aligned}\mathbb{E}[\underline{\mathbf{x}}^T P \underline{\mathbf{x}} \underline{\mathbf{x}}^T Q \underline{\mathbf{x}}] &= \text{tr}(\Sigma P) \text{tr}(\Sigma Q) + \text{tr}(\Sigma P) \text{tr}(\mu \mu^T Q) + \text{tr}(\mu \mu^T P) \text{tr}(\Sigma Q) + \text{tr}(\mu \mu^T P) \text{tr}(\mu \mu^T Q) \\ &\quad + 2\text{tr}(\Sigma P \Sigma Q) + 2\text{tr}(\mu \mu^T P \Sigma Q) + 2\text{tr}(\Sigma P \mu \mu^T Q).\end{aligned}\quad (\text{B.98})$$

If you look closely, you can already see some structure appearing in the above equation. The next step is to bring terms between brackets. (Remember that the trace operator is a linear operator.) Doing so will give us

$$\begin{aligned}\mathbb{E}[\underline{\mathbf{x}}^T P \underline{\mathbf{x}} \underline{\mathbf{x}}^T Q \underline{\mathbf{x}}] &= \text{tr}((\Sigma + \mu \mu^T) P) \text{tr}((\Sigma + \mu \mu^T) Q) \\ &\quad + 2\text{tr}((\Sigma + \mu \mu^T) P (\Sigma + \mu \mu^T) Q) - 2\text{tr}(\mu \mu^T P \mu \mu^T Q).\end{aligned}\quad (\text{B.99})$$

We know that $\Psi = \Sigma + \mu \mu^T$. This allows us to simplify the above equation, directly giving us our final result (B.93). \square

We can generalize the above theorem further to the case where there are multiple Gaussian parameters.

Theorem B.20. Consider two Gaussian random variables \mathbf{x} and \mathbf{y} with joint distribution

$$\begin{bmatrix} \underline{\mathbf{x}} \\ \underline{\mathbf{y}} \end{bmatrix} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{\mathbf{x}} \\ \boldsymbol{\mu}_{\mathbf{y}} \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}\right)\quad (\text{B.100})$$

and squared expectation Ψ_{ab} as $\Sigma_{ab} + \boldsymbol{\mu}_a \boldsymbol{\mu}_b^T$, where the subscripts a and b can be substituted by x and/or y . For any matrices P and Q it holds that

$$\mathbb{E}[\underline{\mathbf{x}}^T P \underline{\mathbf{x}} \underline{\mathbf{y}}^T Q \underline{\mathbf{y}}] = \text{tr}(\Psi_{xx} P) \text{tr}(\Psi_{yy} Q) + 2\text{tr}(\Psi_{yx} P \Psi_{xy} Q) - 2\boldsymbol{\mu}_x^T P \boldsymbol{\mu}_x \boldsymbol{\mu}_y^T Q \boldsymbol{\mu}_y. \quad (\text{B.101})$$

Proof. This theorem follows directly from Theorem B.19 when we apply it to

$$\underline{\mathbf{x}}' = \begin{bmatrix} \underline{\mathbf{x}} \\ \underline{\mathbf{y}} \end{bmatrix}, \quad P' = \begin{bmatrix} P & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad Q' = \begin{bmatrix} 0 & 0 \\ 0 & Q \end{bmatrix}. \quad (\text{B.102})$$

For these parameters, we have

$$\mathbb{E}[\underline{\mathbf{x}}'^T P' \underline{\mathbf{x}}' \underline{\mathbf{x}}'^T Q \underline{\mathbf{x}}'] = \mathbb{E}[\underline{\mathbf{x}}^T P \underline{\mathbf{x}} \underline{\mathbf{y}}^T Q \underline{\mathbf{y}}], \quad (\text{B.103})$$

but because of Theorem B.19, the above also equals

$$\mathbb{E}[\underline{\mathbf{x}}'^T P' \underline{\mathbf{x}}' \underline{\mathbf{x}}'^T Q \underline{\mathbf{x}}'] = \text{tr}(\Psi P') \text{tr}(\Psi Q') + 2\text{tr}(\Psi P' \Psi Q') - 2\boldsymbol{\mu}^T P' \boldsymbol{\mu} \boldsymbol{\mu}^T Q' \boldsymbol{\mu}, \quad (\text{B.104})$$

where $\Psi = \Sigma + \boldsymbol{\mu} \boldsymbol{\mu}^T$. If we expand the above matrix equation, we directly find (B.101). \square

B.5. MANIPULATING GAUSSIAN DISTRIBUTIONS

In this section we will look at ways of manipulating Gaussian distributions. We start by expanding the idea of merging distributions to Gaussian distributions (Section B.5.1). Then we look at what happens when we measure linear relations of Gaussian random variables (Section B.5.2). Finally we examine linear functions whose weights have a Gaussian prior distribution, and how function measurements affect the distribution of those weights (Section B.5.3).

B.5.1. MERGING GAUSSIAN DISTRIBUTIONS

In Section B.3.3 we already looked at merging distributions. We can apply these theories to Gaussian PDFs. Suppose that we have n independent measures of \underline{x} , and each of them tells us that \underline{x} has some Gaussian distribution. The following theorem tells us how we can combine those measurements into a single posterior distribution.

Theorem B.21. *Consider the case where, through n independent measurements, we have found that the random variable \underline{x} satisfies $\underline{x} \sim \mathcal{N}(\underline{x}|\mathbf{m}_1, K_1), \dots, \underline{x} \sim \mathcal{N}(\underline{x}|\mathbf{m}_n, K_n)$. The posterior distribution for \underline{x} is now given by*

$$f_{\underline{x}}^{1:n}(\underline{x}) = \mathcal{N}(\underline{x}|\boldsymbol{\mu}_{1:n}, \Sigma_{1:n}), \quad (\text{B.105})$$

where the mean $\boldsymbol{\mu}_{1:n}$ and the variance $\Sigma_{1:n}$ equal

$$\Sigma_{1:n} = (K_1^{-1} + \dots + K_n^{-1})^{-1}, \quad (\text{B.106})$$

$$\boldsymbol{\mu}_{1:n} = \Sigma_{1:n}(K_1^{-1}\mathbf{m}_1 + \dots + K_n^{-1}\mathbf{m}_n). \quad (\text{B.107})$$

Proof. This theorem can be proven by mathematical induction. For $n = 1$ measurements it is trivial that it holds: if we only know that $\underline{x} \sim \mathcal{N}(\underline{x}|\mathbf{m}_1, K_1)$, then this directly is the posterior distribution of \underline{x} . So now assume that this theorem holds for $n - 1$ measurements. We hence have as posterior distribution for the first $n - 1$ measurements

$$f_{\underline{x}}^{1:(n-1)}(\underline{x}) = \mathcal{N}(\underline{x}|\boldsymbol{\mu}_{1:(n-1)}, \Sigma_{1:(n-1)}), \quad (\text{B.108})$$

where we have

$$\Sigma_{1:(n-1)} = (K_1^{-1} + \dots + K_{n-1}^{-1})^{-1}, \quad (\text{B.109})$$

$$\boldsymbol{\mu}_{1:(n-1)} = \Sigma_{1:(n-1)}(K_1^{-1}\mathbf{m}_1 + \dots + K_{n-1}^{-1}\mathbf{m}_{n-1}). \quad (\text{B.110})$$

If we add measurement n , then Theorem B.9 tells us that the posterior distribution satisfies

$$\begin{aligned} f_{\underline{x}}^{1:n}(\underline{x}) &= \frac{f_{\underline{x}}^{1:(n-1)}(\underline{x})f_{\underline{x}}^n(\underline{x})}{\int_X f_{\underline{x}}^{1:(n-1)}(\underline{x}')f_{\underline{x}}^n(\underline{x}')d\underline{x}'} \\ &= \frac{\mathcal{N}(\underline{x}|\boldsymbol{\mu}_{1:(n-1)}, \Sigma_{1:(n-1)})\mathcal{N}(\underline{x}|\mathbf{m}_n, K_n)}{\int_X \mathcal{N}(\underline{x}'|\boldsymbol{\mu}_{1:(n-1)}, \Sigma_{1:(n-1)})\mathcal{N}(\underline{x}'|\mathbf{m}_n, K_n)d\underline{x'}}. \end{aligned} \quad (\text{B.111})$$

Theorem A.13 now also tells us that

$$\mathcal{N}(\underline{x}|\boldsymbol{\mu}_{1:(n-1)}, \Sigma_{1:(n-1)})\mathcal{N}(\underline{x}|\mathbf{m}_n, K_n) = C\mathcal{N}(\underline{x}|\boldsymbol{\mu}_{1:n}, \Sigma_{1:n}), \quad (\text{B.112})$$

where C is a constant not depending on \underline{x} , while $\boldsymbol{\mu}_{1:n}$ and $\Sigma_{1:n}$ are given by

$$\begin{aligned}\Sigma_{1:n} &= (\Sigma_{1:(n-1)}^{-1} + K_n^{-1})^{-1} \\ &= (K_1^{-1} + \dots + K_n^{-1})^{-1},\end{aligned}\tag{B.113}$$

$$\begin{aligned}\boldsymbol{\mu}_{1:n} &= \Sigma_{1:n} (\Sigma_{1:(n-1)}^{-1} \boldsymbol{\mu}_{1:(n-1)} + K_n^{-1} \mathbf{m}_n) \\ &= \Sigma_{1:n} (\Sigma_{1:(n-1)}^{-1} (\Sigma_{1:(n-1)} (K_1^{-1} \mathbf{m}_1 + \dots + K_{n-1}^{-1} \mathbf{m}_{n-1})) + K_n^{-1} \mathbf{m}_n) \\ &= \Sigma_{1:n} (K_1^{-1} \mathbf{m}_1 + \dots + K_n^{-1} \mathbf{m}_n).\end{aligned}\tag{B.114}$$

It follows that

$$f_{\underline{x}}^{1:n}(\underline{x}) = \frac{C}{\int_X C \mathcal{N}(\underline{x}' | \boldsymbol{\mu}_{1:n}, \Sigma_{1:n}) d\underline{x}'} \mathcal{N}(\underline{x} | \boldsymbol{\mu}_{1:n}, \Sigma_{1:n}).\tag{B.115}$$

We only still need to solve the integral. If we pull C out of the integral, then this integral is an integral over a Gaussian PDF. According to (B.62) this equals 1 and hence drops out. Therefore, the posterior distribution after combining n measurements is

$$f_{\underline{x}}^{1:n}(\underline{x}) = \mathcal{N}(\underline{x} | \boldsymbol{\mu}_{1:n}, \Sigma_{1:n}),\tag{B.116}$$

which proves this theorem by induction. \square

Note that, using the notation introduced in Section B.3.3, we can write

$$\mathcal{N}(\boldsymbol{\mu}_{1:n}, \Sigma_{1:n}) = \mathcal{N}(\mathbf{m}_1, K_1) \oplus \mathcal{N}(\mathbf{m}_2, K_2) \oplus \dots \oplus \mathcal{N}(\mathbf{m}_n, K_n).\tag{B.117}$$

While the above theorem tells us what to do when we have multiple distributions for the full vector \underline{x} , we sometimes may measure only a part of the vector \underline{x} . What happens then? The following theorem explains it.

Theorem B.22. Consider two Gaussian random variables \underline{x}_a and \underline{x}_b with joint (prior) distribution

$$\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} = \underline{x}_{a,b} \sim \mathcal{N}(\underline{x}_{a,b} | \boldsymbol{\mu}_{a,b}, \Sigma_{a,b}) = \mathcal{N}\left(\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} \middle| \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}\right).\tag{B.118}$$

If we, through some independently performed set of measurements, also find that \underline{x}_a has as distribution $\underline{x}_a \sim \mathcal{N}(\underline{x}_a | \hat{\boldsymbol{\mu}}, \hat{\Sigma})$, we have as posterior distribution for $\underline{x}_{a,b}$

$$\begin{aligned}\underline{x}_{a,b} &\sim \mathcal{N}(\underline{x}_{a,b} | \boldsymbol{\mu}_{a,b}, \Sigma_{a,b}), \\ \Sigma_{a,b} &= \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix} = \begin{bmatrix} K_{aa} - K_{aa}(K_{aa} + \hat{\Sigma})^{-1} K_{aa} & K_{ab} - K_{aa}(K_{aa} + \hat{\Sigma})^{-1} K_{ab} \\ K_{ba} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1} K_{aa} & \Sigma_{bb} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1} K_{ab} \end{bmatrix} \\ &= \begin{bmatrix} K_{aa}(K_{aa} + \hat{\Sigma})^{-1} \hat{\Sigma} & \hat{\Sigma}(K_{aa} + \hat{\Sigma})^{-1} K_{ab} \\ K_{ba}(\Sigma_{aa} + \hat{\Sigma})^{-1} \hat{\Sigma} & K_{bb} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1} K_{ab} \end{bmatrix}, \\ \boldsymbol{\mu}_{a,b} &= \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} = \begin{bmatrix} \mathbf{m}_a + K_{aa}(K_{aa} + \hat{\Sigma})^{-1} (\hat{\boldsymbol{\mu}} - \mathbf{m}_a) \\ \mathbf{m}_b + K_{ba}(K_{aa} + \hat{\Sigma})^{-1} (\hat{\boldsymbol{\mu}} - \mathbf{m}_a) \end{bmatrix} \\ &= \begin{bmatrix} \Sigma_{aa} (K_{aa}^{-1} \mathbf{m}_a + \hat{\Sigma}^{-1} \hat{\boldsymbol{\mu}}) \\ \mathbf{m}_b + K_{ba}(K_{aa} + \hat{\Sigma})^{-1} (\hat{\boldsymbol{\mu}} - \mathbf{m}_a) \end{bmatrix}.\end{aligned}\tag{B.119}$$

B

Proof. We will prove this by merging distributions. We have two distributions for $\underline{x}_{a,b}$. The first one is the prior distribution (B.118). The second one is the measured distribution $\underline{x}_a \sim \mathcal{N}(\underline{x}_a | \hat{\mu}, \hat{\Sigma})$. This second distribution tells us nothing about \underline{x}_b . As a result, we can also write this second distribution as

$$\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} = \underline{x}_{a,b} \sim \mathcal{N}(\underline{x}_{a,b} | \hat{\mu}_{a,b}, \hat{\Sigma}_{a,b}) = \mathcal{N}\left(\begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} \mid \begin{bmatrix} \hat{\mu} \\ * \end{bmatrix}, \begin{bmatrix} \hat{\Sigma} & * \\ * & \infty \end{bmatrix}\right). \quad (\text{B.120})$$

The $*$ in the above expression indicates a value which is immaterial, since it will drop out of our equations anyway.

If we now merge the prior distribution with this measurement distribution using Theorem B.21, we should in theory immediately find our desired result

$$\underline{x}_{a,b} \sim \mathcal{N}(\underline{x} | \mu_{a,b}, \Sigma_{a,b}), \quad (\text{B.121})$$

The idea behind the method is nice and intuitive. Now let's look at the math.

We start with the posterior covariance matrix $\Sigma_{a,b}$. We can calculate it through

$$\Sigma_{a,b} = \left(K_{a,b}^{-1} + \hat{\Sigma}_{a,b}^{-1} \right)^{-1} = \left(\begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}^{-1} + \begin{bmatrix} \hat{\Sigma} & * \\ * & \infty \end{bmatrix}^{-1} \right)^{-1}. \quad (\text{B.122})$$

Using either result of Theorem A.11, this directly turns into either of the two expressions of $\Sigma_{a,b}$ from (B.119).

Finding the posterior mean $\mu_{a,b}$ is a somewhat more involved process. We have

$$\begin{aligned} \mu_{a,b} &= \left(K_{a,b}^{-1} + \hat{\Sigma}_{a,b}^{-1} \right)^{-1} \left(K_{a,b}^{-1} \mathbf{m}_{a,b} + \hat{\Sigma}_{a,b}^{-1} \hat{\mu}_{a,b} \right) \\ &= \left(K_{a,b}^{-1} + \hat{\Sigma}_{a,b}^{-1} \right)^{-1} \left(\left(K_{a,b}^{-1} + \hat{\Sigma}_{a,b}^{-1} \right) \mathbf{m}_{a,b} + \hat{\Sigma}_{a,b}^{-1} \hat{\mu}_{a,b} - \hat{\Sigma}_{a,b}^{-1} \mathbf{m}_{a,b} \right) \\ &= \mathbf{m}_{a,b} + \left(K_{a,b}^{-1} + \hat{\Sigma}_{a,b}^{-1} \right)^{-1} \hat{\Sigma}_{a,b}^{-1} (\hat{\mu}_{a,b} - \mathbf{m}_{a,b}) \\ &= \mathbf{m}_{a,b} + K_{a,b} (K_{a,b} + \hat{\Sigma}_{a,b})^{-1} (\hat{\mu}_{a,b} - \mathbf{m}_{a,b}), \end{aligned} \quad (\text{B.123})$$

where in the last part we have used the matrix relation $(P^{-1} + Q^{-1})^{-1} = P(P+Q)^{-1}Q$. The above relation now directly proves the first expression of $\mu_{a,b}$. To prove the second, we rewrite the top term of the vector $\mu_{a,b}$ to

$$\begin{aligned} \mathbf{m}_a + K_{aa} (K_{aa} + \hat{\Sigma})^{-1} (\hat{\mu} - \mathbf{m}_a) &= (K_{aa} + \hat{\Sigma}) (K_{aa} + \hat{\Sigma})^{-1} \mathbf{m}_a + K_{aa} (K_{aa} + \hat{\Sigma})^{-1} (\hat{\mu} - \mathbf{m}_a) \\ &= \hat{\Sigma} (K_{aa} + \hat{\Sigma})^{-1} \mathbf{m}_a + K_{aa} (K_{aa} + \hat{\Sigma})^{-1} \hat{\mu} \\ &= (K_{aa}^{-1} + \hat{\Sigma}^{-1})^{-1} (K_{aa}^{-1} \mathbf{m}_a + \hat{\Sigma}^{-1} \hat{\mu}), \end{aligned} \quad (\text{B.124})$$

which in turn equals the second expression of $\mu_{a,b}$. \square

The result we have found here is actually not very surprising. First of all, we see that the posterior distribution of \underline{x}_a is just the merged distribution of $\mathcal{N}(\mathbf{m}_a, K_{aa})$ and $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$. It is equal to what it would have been if \underline{x}_b did not exist. This makes sense,

because we did not get any additional information about \underline{x}_b . We have only done measurements on \underline{x}_a . Secondly, the posterior distribution for \underline{x}_b equals what we already found at Theorem B.16. The result we have found here is a bit more powerful though, because it also gives us the joint posterior distribution of \underline{x}_a and \underline{x}_b .

Next, in addition to merging distributions, we can also *unmerge* distributions. Suppose that we know the merged distribution of n distributions, but suddenly we find that the last distribution n was actually incorrectly obtained. How do we take it out of the resulting distribution? That's what the following theorem states.

Theorem B.23. *Consider the case where, after n independent measurements of \underline{x} , we have a posterior distribution of*

$$\underline{x} \sim \mathcal{N}(\underline{x} | \boldsymbol{\mu}_{1:n}, \Sigma_{1:n}). \quad (\text{B.125})$$

If we want to take out the effects of measurement n , which claimed that \underline{x} was distributed according to $\underline{x} \sim \mathcal{N}(\underline{x} | \mathbf{m}_n, K_n)$, then this can be done through

$$\Sigma_{1:(n-1)} = (\Sigma_{1:n}^{-1} - K_n^{-1})^{-1}, \quad (\text{B.126})$$

$$\boldsymbol{\mu}_{1:(n-1)} = \Sigma_{1:(n-1)} (\Sigma_{1:n}^{-1} \boldsymbol{\mu}_{1:n} - K_n^{-1} \mathbf{m}_n). \quad (\text{B.127})$$

Proof. This can be proven directly by applying Theorem B.21. From this theorem, we first of all know that the variance satisfies

$$\Sigma_{1:n} = (\Sigma_{1:(n-1)}^{-1} + K_n^{-1})^{-1}, \quad (\text{B.128})$$

which directly implies that

$$\Sigma_{1:(n-1)} = (\Sigma_{1:n}^{-1} - K_n^{-1})^{-1}. \quad (\text{B.129})$$

Theorem B.21 also tells us that the mean equals

$$\boldsymbol{\mu}_{1:n} = \Sigma_{1:n} (K_1^{-1} \mathbf{m}_1 + \dots + K_n^{-1} \mathbf{m}_n) \quad (\text{B.130})$$

For $n - 1$ distributions this can be written as

$$\begin{aligned} \boldsymbol{\mu}_{1:(n-1)} &= \Sigma_{1:(n-1)} (K_1^{-1} \mathbf{m}_1 + \dots + K_{n-1}^{-1} \mathbf{m}_{n-1}) \\ &= \Sigma_{1:(n-1)} (\Sigma_{1:n}^{-1} \Sigma_{1:n} (K_1^{-1} \mathbf{m}_1 + \dots + K_{n-1}^{-1} \mathbf{m}_{n-1} + K_n^{-1} \mathbf{m}_n) - K_n^{-1} \mathbf{m}_n) \\ &= \Sigma_{1:(n-1)} (\Sigma_{1:n}^{-1} \boldsymbol{\mu}_{1:n} - K_n^{-1} \mathbf{m}_n). \end{aligned} \quad (\text{B.131})$$

This completes the proof of this theorem. \square

For unmerging we use the symbol \ominus . Hence we have

$$\mathcal{N}(\boldsymbol{\mu}_{1:(n-1)}, \Sigma_{1:(n-1)}) = \mathcal{N}(\boldsymbol{\mu}_{1:n}, \Sigma_{1:n}) \ominus \mathcal{N}(\mathbf{m}_n, K_n). \quad (\text{B.132})$$

The above theorem might not be very surprising, but it does tell us an interesting fact if we compare it with Theorem B.21. The difference between these two theorems is that, to add/merge a distribution, we need to use the covariance matrix K_n , while to subtract/unmerge a distribution, we need to use the matrix $-K_n$. The minus sign is the only difference! Hence, we have

$$\mathcal{N}(\mathbf{m}_a, K_a) \ominus \mathcal{N}(\mathbf{m}_b, K_b) = \mathcal{N}(\mathbf{m}_a, K_a) \oplus \mathcal{N}(\mathbf{m}_b, -K_b). \quad (\text{B.133})$$

From this, we can conclude that, while a positive (semi-)definite covariance matrix gives additional information about the distribution, a negative (semi-)definite covariance matrix gives disinformation/subtracts information. It basically gives us a distribution on what \underline{x} is *not*, although this is a rather abstract concept.

B

A final thing which we can wonder is ‘What does a covariance matrix with both positive and negative eigenvalues mean?’ In this case, the positive eigenvalues add information about the distribution in some part of the input space, while the negative eigenvalues remove information in their respective parts of the input space. Even though I do not directly see any sensible application of a covariance matrix like this, it is at least fascinating to ponder about.

B.5.2. MEASURING LINEAR RELATIONS OF GAUSSIAN VARIABLES

Suppose that we have a random variable \underline{x} with a certain prior Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$. If we would measure \underline{x} directly, we already know how to process these measurements.

But now suppose that we only measure a linear relation of the elements of \underline{x} . That is, our measurement tells us that $M\underline{x} = \underline{c}$ for some matrix M . Here, we assume that M has relatively few rows, but all rows are linearly independent. So M is of full row rank.

Next, we make the problem a bit more complicated. Let’s suppose that we do not know \underline{c} exactly because of measurement noise $\underline{v} \sim \mathcal{N}(\mathbf{0}, \hat{\Sigma}_c)$. So the measurement $\hat{\underline{c}}$ which we get is not fully accurate. What does this measurement now tell us about the posterior distribution of \underline{x} ?

The next theorem teaches us a trick with which we can solve this problem, and afterwards in Theorem B.25 will we actually solve it.

Theorem B.24. Consider the equation $M\underline{x} = \underline{c}$, where the $m \times n$ matrix M is of full row rank. It is now possible to set up a matrix $T = [T_a \quad T_b]$ such that

$$\underline{x}' = \begin{bmatrix} \underline{x}'_a \\ \underline{x}'_b \end{bmatrix} = \begin{bmatrix} M \\ T_b^T \end{bmatrix} \underline{x} = T^{-1} \underline{x}, \quad (\text{B.134})$$

with \underline{x}'_a fully set as \underline{c} and \underline{x}'_b subject to no constraints whatsoever.

Proof. The key here is to transform \underline{x} into $\underline{x}' = T^{-1}\underline{x}$, choosing T such that $\underline{x}'_a = \underline{c}$ and \underline{x}'_b is free. So how do we do that?

We know that $\underline{x} = T\underline{x}'$, so the equation $M\underline{x} = \underline{c}$ tells us that $MT\underline{x}' = \underline{c}$. To get the appropriate split of \underline{x}' , we want to have $MT = [I \quad 0]$. In other words, if we write $T = [T_a \quad T_b]$, then we should have $MT_a = I$ and $MT_b = 0$.

Let’s take a closer look at these requirements. $MT_b = 0$ requires the $n - m$ columns of T_b to be in the null space of M . (This is the space of all vectors \underline{z} satisfying $M\underline{z} = \mathbf{0}$.) But within this requirement, there is still some freedom for us. Using this freedom, we want to choose a set of $n - m$ linearly independent orthogonal vectors of length 1, which together span the null space of M , and set these as the columns of T_b . When we do, we not only have $MT_b = 0$, but also $T_b^T T_b = I$.

We also have some freedom when choosing T_a . Because we have assumed that M is of full row rank, some matrix T_a satisfying $MT_a = I$ must exist. If we now also make sure

that the columns of T_a are orthogonal to the null space of M , then T_a satisfies $T_b^T T_a = 0$ as well.

Next, we want to find T^{-1} . This is (per definition) the matrix which satisfies $T^{-1} T = I$. We can directly see that

$$T^{-1} = \begin{bmatrix} M \\ T_b^T \end{bmatrix} \quad (\text{B.135})$$

satisfies this condition. So now we have confirmed every part of (B.134). \square

Theorem B.25. Consider a random variable \underline{x} with prior distribution $\underline{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. If additional measurements have told us that $M\underline{x} = \underline{c}$, where $\underline{c} \sim \mathcal{N}(\boldsymbol{\mu}_c, \Sigma_c)$ is independent of \underline{x} , then the posterior distribution of \underline{x} is given by

$$\begin{aligned} \underline{x} &\sim \mathcal{N}(\boldsymbol{\mu}, \Sigma), \\ \Sigma &= K - KM^T (MKM^T + \Sigma_c)^{-1} MK, \\ \boldsymbol{\mu} &= \mathbf{m} + KM^T (MKM^T + \Sigma_c)^{-1} (\boldsymbol{\mu}_c - M\mathbf{m}). \end{aligned} \quad (\text{B.136})$$

Proof. The key to proving this is to apply the transformation from Theorem B.24. That is, we have $\underline{x}' = T^{-1} \underline{x}$ and will apply the merging of distributions for \underline{x}' .

Using Theorem B.13, we can find that the prior of \underline{x}' equals

$$\underline{x}' \sim \mathcal{N}(T^{-1} \mathbf{m}, T^{-1} K T^{-T}) = \mathcal{N}\left(\begin{bmatrix} M\mathbf{m} \\ T_b^T \mathbf{m} \end{bmatrix}, \begin{bmatrix} MKM^T & MKT_b \\ T_b^T KM^T & T_b^T KT_b \end{bmatrix}\right). \quad (\text{B.137})$$

Our measurement has told us that $\underline{x}'_a = M\underline{x}' = \underline{c}$, while $\underline{x}'_b = T_b^T \underline{x}'$ is still fully unknown. In other words, the measurement has given us the distribution

$$\underline{x}' \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_c \\ * \end{bmatrix}, \begin{bmatrix} \Sigma_c & * \\ * & \infty \end{bmatrix}\right). \quad (\text{B.138})$$

We now want to merge these two distributions. We could do so using Theorem B.21, although using Theorem B.22 gives us the outcome directly. Using the first expressions for both the mean and the covariance, we find that

$$\begin{aligned} \underline{x}' &\sim \mathcal{N}(\boldsymbol{\mu}', \Sigma') \quad (\text{B.139}) \\ \Sigma' &= \begin{bmatrix} MKM^T - MKM^T (MKM^T + \Sigma_c)^{-1} MKM^T & MKT_b - MKM^T (MKM^T + \Sigma_c)^{-1} MKT_b \\ T_b^T KM^T - T_b^T KM^T (MKM^T + \Sigma_c)^{-1} MKM^T & T_b^T KT_b - T_b^T KM^T (MKM^T + \Sigma_c)^{-1} MKT_b \end{bmatrix} \\ &= T^{-1} \left(K - KM^T (MKM^T + \Sigma_c)^{-1} MK \right) T^{-T}, \\ \boldsymbol{\mu}' &= \begin{bmatrix} M\mathbf{m} + MKM^T (MKM^T + \Sigma_c)^{-1} (\boldsymbol{\mu}_c - M\mathbf{m}) \\ T_b^T \mathbf{m} + T_b^T KM^T (MKM^T + \Sigma_c)^{-1} (\boldsymbol{\mu}_c - M\mathbf{m}) \end{bmatrix} \\ &= T^{-1} \left(\mathbf{m} + KM^T (MKM^T + \Sigma_c)^{-1} (\boldsymbol{\mu}_c - M\mathbf{m}) \right). \end{aligned}$$

This is the posterior distribution of \underline{x}' . Note that, because $\underline{c} = \underline{x}'_a$, we now have actually found the posterior distribution of \underline{c} as well.

However, we do not want the posterior distribution of \underline{x}' . We want the posterior distribution of \underline{x} . To find this, we apply the inverse transformation according to

$$\begin{aligned}\underline{x} &= T\underline{x}' \sim \mathcal{N}(T\mu', T\Sigma' T^T) = \mathcal{N}(\mu, \Sigma), \\ \Sigma &= K - KM^T(MKM^T + \Sigma_c)^{-1}MK, \\ \mu &= \mathbf{m} + KM^T(MKM^T + \Sigma_c)^{-1}(\mu_c - M\mathbf{m}).\end{aligned}\tag{B.140}$$

Note that both T_a and T_b have dropped out of the equations entirely, meaning we do not even have to find them. Knowing M is sufficient. \square

Next, suppose that we have two random vectors \underline{x}_a and \underline{x}_b with a joint distribution. If we now know that $M\underline{x}_a = \underline{c}$, can we then also say something about \underline{x}_b ? That is explained by the next theorem.

Theorem B.26. Consider the two random variables \underline{x}_a and \underline{x}_b with joint Gaussian distribution

$$\underline{x} = \begin{bmatrix} \underline{x}_a \\ \underline{x}_b \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}\right).\tag{B.141}$$

If we know that $M\underline{x}_a = \underline{c} \sim \mathcal{N}(\mu_c, \Sigma_c)$, then the posterior distribution of \underline{x} is given by

$$\begin{aligned}\underline{x} &\sim \mathcal{N}(\mu, \Sigma), \\ \Sigma &= \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} - \begin{bmatrix} K_{aa} \\ K_{ba} \end{bmatrix} M^T (MK_{aa}M^T + \Sigma_c)^{-1} M \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}, \\ \mu &= \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix} + \begin{bmatrix} K_{aa} \\ K_{ba} \end{bmatrix} M^T (MK_{aa}M^T + \Sigma_c)^{-1} (\mu_c - M\mathbf{m}_a).\end{aligned}\tag{B.142}$$

Proof. The key to proving this is to define $N = [M \ 0]$. Now we have $N\underline{x} = \underline{c}$ and we can directly apply Theorem B.25. That is,

$$\begin{aligned}\underline{x} &\sim \mathcal{N}(\mu, \Sigma), \\ \Sigma &= \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} - \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} \begin{bmatrix} M^T \\ 0 \end{bmatrix} \left(\begin{bmatrix} M & 0 \end{bmatrix} \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} \begin{bmatrix} M^T \\ 0 \end{bmatrix} + \Sigma_c \right)^{-1} \begin{bmatrix} M & 0 \end{bmatrix} \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}, \\ \mu &= \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix} + \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} \begin{bmatrix} M^T \\ 0 \end{bmatrix} \left(\begin{bmatrix} M & 0 \end{bmatrix} \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} \begin{bmatrix} M^T \\ 0 \end{bmatrix} + \Sigma_c \right)^{-1} (\mu_c - \begin{bmatrix} M & 0 \end{bmatrix} \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}).\end{aligned}\tag{B.143}$$

By expanding the N matrices in the above expression, we directly get (B.142). \square

It is interesting to note here that (B.142) is a generalization of (B.119). Setting M to I will directly reduce (B.142) back to (B.119).

B.5.3. LINEAR FUNCTIONS WITH GAUSSIAN WEIGHTS

Consider the linear function $f(\mathbf{x}) = \underline{w}^T \mathbf{x}$, where the weights are unknown. If we do measurements of this function, we can learn more about the distribution of the weights \underline{w} . In fact, when the weights have a prior distribution which is Gaussian, their posterior distribution will be Gaussian too. The following theorem tells us how we can find it.

Theorem B.27. Consider the linear function $f(\mathbf{x}) = \underline{\mathbf{w}}^T \mathbf{x} = \mathbf{x}^T \underline{\mathbf{w}}$, where the weights $\underline{\mathbf{w}}$ have as prior distribution $\underline{\mathbf{w}} \sim \mathcal{N}(\mathbf{m}_w, K_w)$. Suppose that, at the input points $X_m = [\mathbf{x}_{m_1}, \dots, \mathbf{x}_{m_{n_m}}]$, the function values have been measured. After corruption by noise $\underline{\mathbf{v}} \sim \mathcal{N}(\mathbf{0}, \hat{\Sigma}_f)$, the measured function values were $\hat{\mathbf{f}}_m = [\hat{f}_{m_1} \ \dots \ \hat{f}_{m_{n_m}}]^T$. The posterior distribution of $\underline{\mathbf{w}}$ now equals

$$\begin{aligned}\underline{\mathbf{w}} &\sim \mathcal{N}(\boldsymbol{\mu}_w, \Sigma_w), \\ \Sigma_w &= \left(X_m \hat{\Sigma}_f^{-1} X_m^T + K_w^{-1} \right)^{-1}, \\ \boldsymbol{\mu}_w &= \Sigma_w \left(X_m \hat{\Sigma}_f^{-1} \hat{\mathbf{f}}_m + K_w^{-1} \mathbf{m}_w \right).\end{aligned}\tag{B.144}$$

Proof. According to Bayes' theorem, we have

$$p(\mathbf{w} | \hat{\mathbf{f}}_m, X_m) = \frac{p(\hat{\mathbf{f}}_m | \mathbf{w}, X_m) p(\mathbf{w} | X_m)}{p(\hat{\mathbf{f}}_m | X_m)}.\tag{B.145}$$

The first probability $p(\hat{\mathbf{f}}_m | \mathbf{w}, X_m)$ in the fraction represents the probability that, given that the weights $\underline{\mathbf{w}}$ actually equal \mathbf{w} , we happened to wind up with the measurements $\hat{\mathbf{f}}_m$. To calculate it, we should keep in mind that, when the weights $\underline{\mathbf{w}}$ are known deterministically, then the output of the function $f(\mathbf{x})$ is fully determined as $\underline{\mathbf{f}}_m = X_m^T \underline{\mathbf{w}}$. So the output $\hat{\mathbf{f}}_m$ that we measure only varies due to noise. We hence have

$$p(\hat{\mathbf{f}}_m | \mathbf{w}, X_m) = \mathcal{N}(\hat{\mathbf{f}}_m | X_m^T \mathbf{w}, \hat{\Sigma}_f).\tag{B.146}$$

The second probability $p(\mathbf{w} | X_m)$ is the probability that we got weights \mathbf{w} , given the measurement points X_m . However, the position of the measurement points alone does not tell us anything about the weights $\underline{\mathbf{w}}$. As such, this probability equals $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_w, K_w)$. It is the prior distribution of the weights.

Finally, the third probability $p(\hat{\mathbf{f}}_m | X_m)$ does not depend on \mathbf{w} at all. It is hence a constant.

Now it is time to dive into the mathematics. We know that $p(\mathbf{w} | \hat{\mathbf{f}}_m, X_m)$ is the product of two Gaussian PDFs. It is hence also a PDF. This means that we can ignore multiplying constants. After all, the constant is only present to ensure that the integral over the PDF equals one. As a result, we can write

$$\begin{aligned}p(\mathbf{w} | \hat{\mathbf{f}}_m, X_m) &\propto \exp\left(-\frac{1}{2} \left(\hat{\mathbf{f}}_m - X_m^T \mathbf{w} \right)^T \hat{\Sigma}_f^{-1} \left(\hat{\mathbf{f}}_m - X_m^T \mathbf{w} \right)\right) \exp\left(-\frac{1}{2} (\mathbf{w} - \mathbf{m}_w)^T K_w^{-1} (\mathbf{w} - \mathbf{m}_w)\right) \\ &= \exp\left(-\frac{1}{2} (\mathbf{w}^T X_m \hat{\Sigma}_f^{-1} X_m^T \mathbf{w} - \mathbf{w}^T X_m \hat{\Sigma}_f^{-1} \hat{\mathbf{f}}_m - \hat{\mathbf{f}}_m^T \hat{\Sigma}_f^{-1} X_m^T \mathbf{w} + \hat{\mathbf{f}}_m^T \hat{\Sigma}_f^{-1} \hat{\mathbf{f}}_m\right. \\ &\quad \left. + \mathbf{w}^T K_w^{-1} \mathbf{w} - \mathbf{w}^T K_w^{-1} \mathbf{m}_w - \mathbf{m}_w^T K_w^{-1} \mathbf{w} + \mathbf{m}_w^T K_w^{-1} \mathbf{m}_w)\right).\end{aligned}\tag{B.147}$$

We want to write this as a single Gaussian PDF. So we want to have

$$\begin{aligned}p(\mathbf{w} | \hat{\mathbf{f}}_m, X_m) &\propto \exp\left(-\frac{1}{2} (\mathbf{w} - \boldsymbol{\mu}_w)^T \Sigma_w^{-1} (\mathbf{w} - \boldsymbol{\mu}_w)\right) \\ &= \exp\left(-\frac{1}{2} (\mathbf{w}^T \Sigma_w^{-1} \mathbf{w} - \mathbf{w}^T \Sigma_w^{-1} \boldsymbol{\mu}_w - \boldsymbol{\mu}_w^T \Sigma_w^{-1} \mathbf{w} + \boldsymbol{\mu}_w^T \Sigma_w^{-1} \boldsymbol{\mu}_w)\right)\end{aligned}\tag{B.148}$$

for some posterior mean μ_w and covariance Σ_w . We can now see right away that

$$\Sigma_w = \left(X_m \hat{\Sigma}_f^{-1} X_m^T + K_w^{-1} \right)^{-1}. \quad (\text{B.149})$$

And because we have $w^T \Sigma_w^{-1} \mu_w = w^T X_m \hat{\Sigma}_f^{-1} \hat{f}_m + w^T K_w^{-1} m_w$, we also have

$$\mu_w = \Sigma_w \left(X_m \hat{\Sigma}_f^{-1} \hat{f}_m + K_w^{-1} m_w \right). \quad (\text{B.150})$$

All the remaining terms from the exponential do not depend on w and are hence constants which we can ignore. So we have completed the proof. \square

We can extend the above theorem by bringing in the ideas of Section B.5.2. That is, we assume that we only measure a linear relation of the elements of \underline{x} . In that case we get the following theorem.

Theorem B.28. Consider the linear function $f(\underline{x}) = \underline{w}^T \underline{x}$, where the weights \underline{w} have as prior distribution $\underline{w} \sim \mathcal{N}(\mathbf{0}, K_w)$. Denote the set of input points by $X_m = [\underline{x}_{m_1}, \dots, \underline{x}_{m_{n_m}}]$ and the corresponding function values by $\underline{f}_m = f(X_m)$. Suppose that we have measured $\underline{c} = M \underline{f}_m$ for some known matrix M . After corruption by noise $\underline{v} \sim \mathcal{N}(\mathbf{0}, \hat{\Sigma}_c)$, our measurement gives us the vector $\hat{\underline{c}}$. The posterior distribution of \underline{w} now equals

$$\underline{w} \sim \mathcal{N}(\mu_w, \Sigma_w), \quad (\text{B.151})$$

$$\Sigma_w = \left(X_m M^T \hat{\Sigma}_c^{-1} M X_m^T + K_w^{-1} \right)^{-1},$$

$$\mu_w = \Sigma_w \left(X_m M^T \hat{\Sigma}_c^{-1} \hat{\underline{c}} + K_w^{-1} m_w \right).$$

Proof. The only difference with respect to Theorem B.27 is that we now do not measure $\underline{f}_m = X_m^T \underline{w}$ but $\underline{c} = M \underline{f}_m = M X_m^T \underline{w}$. Hence, if we replace X_m^T by $M X_m^T$, $\hat{\underline{f}}_m$ by $\hat{\underline{c}}$ and $\hat{\Sigma}_f$ by $\hat{\Sigma}_c$, then we get exactly the same problem. Making this substitution turns (B.144) into (B.151). \square

B.6. CONDITIONALLY INDEPENDENT GAUSSIAN VARIABLES

In this section we will examine the concept of conditional independence. How is it defined and what does it imply?

We will start with the basic ideas and implications of two conditionally independent vectors (Section B.6.1). Then we examine what changes when elements within a vector are also conditionally independent with respect to each other (Section B.6.2). We generalize this idea to when parts of a vector (that is, small groups of elements) are conditionally independent with each other (Section B.6.3). Finally, we look into methods through which we can update the posterior distribution of random vectors when we add a single conditionally independent element or group of elements (Section B.6.4).

B.6.1. CONDITIONAL INDEPENDENCE OF RANDOM VECTORS

We know that two random variables \underline{x}_a and \underline{x}_b are independent if and only if they satisfy (B.11). We also know from Theorem B.6 that the covariance $\mathbb{V}[\underline{x}_a, \underline{x}_b]$ of two independent variables equals zero. Since Gaussian distributions are completely specified by

their mean and covariance, it follows that for Gaussian distributions the converse also holds: if $\mathbb{V}[\underline{x}_a, \underline{x}_b] = 0$, then \underline{x}_a and \underline{x}_b are independent.

Now we are going to introduce a concept which is similar. We say that two variables \underline{x}_a and \underline{x}_c are *conditionally independent given \underline{x}_b* if and only if

$$f_{\underline{x}_a, \underline{x}_c | \underline{x}_b = \underline{x}_b}(\underline{x}_a, \underline{x}_c) = f_{\underline{x}_a | \underline{x}_b = \underline{x}_b}(\underline{x}_a) f_{\underline{x}_c | \underline{x}_b = \underline{x}_b}(\underline{x}_c). \quad (\text{B.152})$$

We will only apply this idea to Gaussian random variables, so let's take a look at what this conditional independence implies for the covariance $\mathbb{V}[\underline{x}_a, \underline{x}_c]$ of Gaussian variables.

Theorem B.29. Consider the Gaussian random variables \underline{x}_a , \underline{x}_b and \underline{x}_c with joint distribution

$$\underline{x}_{a,b,c} = \begin{bmatrix} \underline{x}_a \\ \underline{x}_b \\ \underline{x}_c \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \\ \mathbf{m}_c \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} & K_{ac} \\ K_{ba} & K_{bb} & K_{bc} \\ K_{ca} & K_{cb} & K_{cc} \end{bmatrix} \right). \quad (\text{B.153})$$

The variables \underline{x}_a and \underline{x}_c are conditionally independent given \underline{x}_b if and only if $K_{ac} = K_{ab}K_{bb}^{-1}K_{bc}$.

Proof. Our starting point is Theorem B.15. It tells us that

$$\underline{x}_a | \underline{x}_b = \underline{x}_b \sim \mathcal{N}(\underline{x}_a | \mathbf{m}_a + K_{ab}K_{bb}^{-1}(\underline{x}_b - \mathbf{m}_b), K_{aa} - K_{ab}K_{bb}^{-1}K_{ba}), \quad (\text{B.154})$$

$$\underline{x}_c | \underline{x}_b = \underline{x}_b \sim \mathcal{N}(\underline{x}_c | \mathbf{m}_c + K_{cb}K_{bb}^{-1}(\underline{x}_b - \mathbf{m}_b), K_{cc} - K_{cb}K_{bb}^{-1}K_{bc}). \quad (\text{B.155})$$

Similarly, if we consider \underline{x}_a and \underline{x}_c together, we get

$$\begin{aligned} \left[\begin{array}{c} \underline{x}_a \\ \underline{x}_c \end{array} \right] | (\underline{x}_b = \underline{x}_b) &\sim \mathcal{N} \left(\left[\begin{array}{c} \underline{x}_a \\ \underline{x}_c \end{array} \right] \middle| \left[\begin{array}{c} \mathbf{m}_a \\ \mathbf{m}_c \end{array} \right] + \begin{bmatrix} K_{ab} \\ K_{cb} \end{bmatrix} K_{bb}^{-1}(\underline{x}_b - \mathbf{m}_b), \right. \\ &\quad \left. \begin{bmatrix} K_{aa} & K_{ac} \\ K_{ca} & K_{cc} \end{bmatrix} - \begin{bmatrix} K_{ab} \\ K_{cb} \end{bmatrix} K_{bb}^{-1} \begin{bmatrix} K_{ba} & K_{bc} \end{bmatrix} \right) \\ &= \mathcal{N} \left(\left[\begin{array}{c} \underline{x}_a \\ \underline{x}_c \end{array} \right] \middle| \left[\begin{array}{c} \mathbf{m}_a - K_{ab}K_{bb}^{-1}(\underline{x}_b - \mathbf{m}_b) \\ \mathbf{m}_c - K_{cb}K_{bb}^{-1}(\underline{x}_b - \mathbf{m}_b) \end{array} \right], \begin{bmatrix} K_{aa} - K_{ab}K_{bb}^{-1}K_{ba} & K_{ac} - K_{ab}K_{bb}^{-1}K_{bc} \\ K_{ca} - K_{cb}K_{bb}^{-1}K_{ba} & K_{cc} - K_{cb}K_{bb}^{-1}K_{bc} \end{bmatrix} \right). \end{aligned} \quad (\text{B.156})$$

From this we can see that (B.152) holds for all \underline{x}_a and \underline{x}_c if and only if $K_{ac} = K_{ab}K_{bb}^{-1}K_{bc}$, completing the proof. \square

In other words, if we assume that \underline{x}_a and \underline{x}_c are conditionally independent given \underline{x}_b , we are effectively assuming that $K_{ac} = K_{ab}K_{bb}^{-1}K_{bc}$.

What does conditional independence mean from an intuitive point of view though? We know that, if we have found \underline{x}_b deterministically, then \underline{x}_a and \underline{x}_c are independent. Extra information about \underline{x}_a will not tell us anything about the distribution of \underline{x}_c and vice versa.

If \underline{x}_b is not known yet, then extra information about \underline{x}_a *can* tell us something about \underline{x}_c . These two random variables are not independent after all; only conditionally independent. So let's do a measurement of \underline{x}_a and see how this affects the posterior distribution of all the variables. Or in other words, how is Theorem B.22 affected by the assumption of conditional independence?

Theorem B.30. Consider the Gaussian random variables \underline{x}_a , \underline{x}_b and \underline{x}_c with joint prior distribution

$$\underline{x}_{a,b,c} = \begin{bmatrix} \underline{x}_a \\ \underline{x}_b \\ \underline{x}_c \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \\ \mathbf{m}_c \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} & K_{ac} \\ K_{ba} & K_{bb} & K_{bc} \\ K_{ca} & K_{cb} & K_{cc} \end{bmatrix}\right). \quad (\text{B.157})$$

B

We assume that \underline{x}_a and \underline{x}_c are conditionally independent given \underline{x}_b . If we, through some independently performed set of measurements, also find that \underline{x}_a has as distribution $\underline{x}_a \sim \mathcal{N}(\underline{x}_a | \hat{\mu}, \hat{\Sigma})$, we have as posterior distribution for $\underline{x}_{a,b}$

$$\underline{x}_{a,b} \sim \mathcal{N}(\underline{x}_{a,b} | \boldsymbol{\mu}_{a,b}, \Sigma_{a,b}), \quad (\text{B.158})$$

$$\begin{aligned} \Sigma_{a,b} &= \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix} = \begin{bmatrix} K_{aa}(K_{aa} + \hat{\Sigma})^{-1}\hat{\Sigma} & \hat{\Sigma}(K_{aa} + \hat{\Sigma})^{-1}K_{ab} \\ K_{ba}(K_{aa} + \hat{\Sigma})^{-1}\hat{\Sigma} & K_{bb} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ab} \end{bmatrix}, \\ \boldsymbol{\mu}_{a,b} &= \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} = \begin{bmatrix} \Sigma_{aa}(K_{aa}^{-1}\mathbf{m}_a + \hat{\Sigma}^{-1}\hat{\mu}) \\ \mathbf{m}_b + K_{ba}(K_{aa} + \hat{\Sigma})^{-1}(\hat{\mu} - \mathbf{m}_a) \end{bmatrix}. \end{aligned}$$

After this, the posterior distribution of $\underline{x}_{b,c}$ can be derived using only the posterior distribution of \underline{x}_b . It equals

$$\underline{x}_{b,c} \sim \mathcal{N}(\underline{x}_{b,c} | \boldsymbol{\mu}_{b,c}, \Sigma_{b,c}), \quad (\text{B.159})$$

$$\begin{aligned} \Sigma_{b,c} &= \begin{bmatrix} \Sigma_{bb} & \Sigma_{bc} \\ \Sigma_{cb} & \Sigma_{cc} \end{bmatrix} = \begin{bmatrix} \Sigma_{bb} & \Sigma_{bb}K_{bb}^{-1}K_{bc} \\ K_{cb}K_{bb}^{-1}\Sigma_{bb} & K_{cc} - K_{cb}K_{bb}^{-1}(K_{bb} - \Sigma_{bb})K_{bb}^{-1}K_{bc} \end{bmatrix}, \\ \boldsymbol{\mu}_{b,c} &= \begin{bmatrix} \boldsymbol{\mu}_b \\ \boldsymbol{\mu}_c \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_b \\ \mathbf{m}_c + K_{cb}K_{bb}^{-1}(\boldsymbol{\mu}_b - \mathbf{m}_b) \end{bmatrix}. \end{aligned}$$

Proof. We should note that the first expression (B.158) directly follows from Theorem B.22. In fact, it equals (B.119). The second expression (B.159) is a different story though.

We will start our proof of (B.159) with the covariance matrix. If we use Theorem B.22, taking only the bottom right element of the covariance matrix from (B.119) but applying it to $\underline{x}_{b,c}$, we find that

$$\begin{aligned} \begin{bmatrix} \Sigma_{bb} & \Sigma_{bc} \\ \Sigma_{cb} & \Sigma_{cc} \end{bmatrix} &= \begin{bmatrix} K_{bb} & K_{bc} \\ K_{cb} & K_{cc} \end{bmatrix} - \begin{bmatrix} K_{ba} \\ K_{ca} \end{bmatrix} (K_{aa} + \hat{\Sigma})^{-1} \begin{bmatrix} K_{ab} & K_{ac} \end{bmatrix} \\ &= \begin{bmatrix} K_{bb} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ab} & K_{bc} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ac} \\ K_{cb} - K_{ca}(K_{aa} + \hat{\Sigma})^{-1}K_{ab} & K_{cc} - K_{ca}(K_{aa} + \hat{\Sigma})^{-1}K_{ac} \end{bmatrix}. \end{aligned} \quad (\text{B.160})$$

We should keep in mind that (B.159) does not contain any term related to \underline{x}_a . To obtain such a relation, we have to use the assumption of conditional independence and apply $K_{ac} = K_{ab}K_{bb}^{-1}K_{bc}$ from Theorem B.29.

We will do so for each term in the matrix individually. First of all, we note that the top left term equals Σ_{bb} , which we already knew. The top right term becomes

$$\begin{aligned} \Sigma_{bc} &= K_{bc} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ab}K_{bb}^{-1}K_{bc} \\ &= \left(K_{bb} - K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ab} \right) K_{bb}^{-1}K_{bc} \\ &= \Sigma_{bb}K_{bb}^{-1}K_{bc}. \end{aligned} \quad (\text{B.161})$$

The bottom left term Σ_{cb} is simply the transpose of Σ_{bc} . (Note that all involved matrices are symmetric.) Finally, the bottom right term equals

$$\begin{aligned}\Sigma_{cc} &= K_{cc} - K_{cb}K_{bb}^{-1}K_{ba}(K_{aa} + \hat{\Sigma})^{-1}K_{ab}K_{bb}^{-1}K_{bc} \\ &= K_{cc} - K_{cb}K_{bb}^{-1}(K_{bb} - \Sigma_{bb})K_{bb}^{-1}K_{bc}.\end{aligned}\quad (\text{B.162})$$

That leaves us with the mean $\mu_{b,c}$. Theorem B.22 tells us that it equals

$$\begin{bmatrix} \mu_b \\ \mu_c \end{bmatrix} = \begin{bmatrix} \mathbf{m}_b \\ \mathbf{m}_c \end{bmatrix} + \begin{bmatrix} K_{ba} \\ K_{ca} \end{bmatrix}(K_{aa} + \hat{\Sigma})^{-1}(\hat{\mu} - \mathbf{m}_a). \quad (\text{B.163})$$

The top term now directly equals μ_b , which again confirms what we already knew. The bottom term can be rewritten to

$$\begin{aligned}\mu_c &= \mathbf{m}_c + K_{ca}(K_{aa} + \hat{\Sigma})^{-1}(\hat{\mu} - \mathbf{m}_a) \\ &= \mathbf{m}_c + K_{cb}K_{bb}^{-1}K_{ba}(K_{aa} + \hat{\Sigma})^{-1}(\hat{\mu} - \mathbf{m}_a) \\ &= \mathbf{m}_c + K_{cb}K_{bb}^{-1}(\mu_b - \mathbf{m}_b).\end{aligned}\quad (\text{B.164})$$

And with this term we have completed the proof. \square

The above theorem is useful, because it allows us to calculate \underline{x}_c in steps. First we use our measurement of \underline{x}_a to calculate the posterior distribution of \underline{x}_b . Then we can discard all data concerning \underline{x}_a , because knowing the posterior distribution of \underline{x}_b is sufficient to calculate the posterior distribution of \underline{x}_c . In fact, that is the whole idea behind conditional independence. What it effectively comes down to, is that \underline{x}_a and \underline{x}_c can only ‘exchange information’ through \underline{x}_b .

There is also an alternative view on the above theorem. We can also derive it through merging distributions. The following theorem explains how this works.

Theorem B.31. *The following merging of the prior and measured distributions*

$$\mathcal{N}\left(\begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}\right) \oplus \mathcal{N}\left(\begin{bmatrix} \hat{\mu} \\ * \end{bmatrix}, \begin{bmatrix} \hat{\Sigma} & * \\ * & \infty \end{bmatrix}\right) \quad (\text{B.165})$$

results in (B.158), while the following merging of the prior distribution of $\underline{x}_{b,c}$ and posterior distribution of \underline{x}_b and unmerging of the prior distribution of \underline{x}_b

$$\mathcal{N}\left(\begin{bmatrix} \mathbf{m}_b \\ \mathbf{m}_c \end{bmatrix}, \begin{bmatrix} K_{bb} & K_{bc} \\ K_{cb} & K_{cc} \end{bmatrix}\right) \oplus \mathcal{N}\left(\begin{bmatrix} \mu_b \\ * \end{bmatrix}, \begin{bmatrix} \Sigma_{bb} & * \\ * & \infty \end{bmatrix}\right) \ominus \mathcal{N}\left(\begin{bmatrix} \mathbf{m}_b \\ * \end{bmatrix}, \begin{bmatrix} K_{bb} & * \\ * & \infty \end{bmatrix}\right) \quad (\text{B.166})$$

results in (B.159).

Proof. The first of these two claims is trivial. The reason is that (B.158) followed from Theorem B.22 which was proven by merging these exact distributions and working out the results. So the proof is identical to the proof of Theorem B.22.

For the second claim, we need some more mathematics. It helps if we first resolve the unmerge \ominus sign. Let’s define its resolution as

$$\begin{aligned}\mathcal{N}(\bar{\mu}_b, \bar{\Sigma}_{bb}) &\equiv \mathcal{N}(\mu_b, \Sigma_{bb}) \ominus \mathcal{N}(\mathbf{m}_b, K_{bb}) \\ &= \mathcal{N}\left(\left(\Sigma_{bb}^{-1} - K_{bb}^{-1}\right)^{-1}(\Sigma_{bb}^{-1}\mu_b - K_{bb}^{-1}\mathbf{m}_b), \left(\Sigma_{bb}^{-1} - K_{bb}^{-1}\right)^{-1}\right).\end{aligned}\quad (\text{B.167})$$

Next, we merge this together with the prior distribution of $\underline{x}_{b,c}$, while applying Theorem B.22. As covariance matrix, we now get

$$\begin{bmatrix} \Sigma_{bb} & \Sigma_{bc} \\ \Sigma_{cb} & \Sigma_{cc} \end{bmatrix} = \begin{bmatrix} K_{bb}(K_{bb} + \bar{\Sigma}_{bb})^{-1} \bar{\Sigma}_{bb} & \bar{\Sigma}_{bb}(K_{bb} + \bar{\Sigma}_{bb})^{-1} K_{bc} \\ K_{cb}(K_{bb} + \bar{\Sigma}_{bb})^{-1} \bar{\Sigma}_{bb} & K_{cc} - K_{cb}(K_{bb} + \bar{\Sigma}_{bb})^{-1} K_{bc} \end{bmatrix}. \quad (\text{B.168})$$

B

All of the terms in the matrix contain $(K_{bb} + \bar{\Sigma}_{bb})^{-1}$. So to rewrite this, it helps if we use definition (B.167) of $\bar{\Sigma}_{bb}$ to find that

$$\begin{aligned} (K_{bb} + \bar{\Sigma}_{bb})^{-1} &= K_{bb}^{-1} (K_{bb}^{-1} + \bar{\Sigma}_{bb}^{-1})^{-1} \bar{\Sigma}_{bb}^{-1} \\ &= K_{bb}^{-1} (K_{bb}^{-1} + \Sigma_{bb}^{-1} - K_{bb}^{-1})^{-1} \bar{\Sigma}_{bb}^{-1} \\ &= K_{bb}^{-1} \Sigma_{bb} \bar{\Sigma}_{bb}^{-1} \\ &= K_{bb}^{-1} \Sigma_{bb} (\Sigma_{bb}^{-1} - K_{bb}^{-1}) \\ &= K_{bb}^{-1} (K_{bb} - \Sigma_{bb}) K_{bb}^{-1}. \end{aligned} \quad (\text{B.169})$$

By cleverly inserting the right lines from (B.169) (or their transposes) into (B.168), we can turn it into

$$\begin{bmatrix} \Sigma_{bb} & \Sigma_{bc} \\ \Sigma_{cb} & \Sigma_{cc} \end{bmatrix} = \begin{bmatrix} \Sigma_{bb} & \Sigma_{bb} K_{bb}^{-1} K_{bc} \\ K_{cb} K_{bb}^{-1} \Sigma_{bb} & K_{cc} - K_{cb} K_{bb}^{-1} (K_{bb} - \Sigma_{bb}) K_{bb}^{-1} K_{bc} \end{bmatrix}, \quad (\text{B.170})$$

which equals the covariance from (B.159). Evaluating the mean is done similarly. Theorem B.22 tells us that

$$\begin{bmatrix} \boldsymbol{\mu}_b \\ \boldsymbol{\mu}_c \end{bmatrix} = \begin{bmatrix} \mathbf{m}_b + K_{bb}(K_{bb} + \bar{\Sigma}_{bb})^{-1}(\bar{\boldsymbol{\mu}}_b - \mathbf{m}_b) \\ \mathbf{m}_c + K_{cb}(K_{bb} + \bar{\Sigma}_{bb})^{-1}(\bar{\boldsymbol{\mu}}_b - \mathbf{m}_b) \end{bmatrix}. \quad (\text{B.171})$$

The common term in both of the above vector elements is $(K_{bb} + \bar{\Sigma}_{bb})^{-1}(\bar{\boldsymbol{\mu}}_b - \mathbf{m}_b)$. By using definition (B.167) of $\bar{\boldsymbol{\mu}}_b$ and by using the third line of (B.169), we can write this as

$$\begin{aligned} (K_{bb} + \bar{\Sigma}_{bb})^{-1}(\bar{\boldsymbol{\mu}}_b - \mathbf{m}_b) &= K_{bb}^{-1} \Sigma_{bb} \bar{\Sigma}_{bb}^{-1} (\bar{\Sigma}_{bb} (\Sigma_{bb}^{-1} \boldsymbol{\mu}_b - K_{bb}^{-1} \mathbf{m}_b) - \mathbf{m}_b) \\ &= K_{bb}^{-1} (\boldsymbol{\mu}_b - \Sigma_{bb} (K_{bb}^{-1} \mathbf{m}_b + \bar{\Sigma}_{bb}^{-1} \mathbf{m}_b)) \\ &= K_{bb}^{-1} (\boldsymbol{\mu}_b - \mathbf{m}_b). \end{aligned} \quad (\text{B.172})$$

Using this result will turn (B.171) into

$$\begin{bmatrix} \boldsymbol{\mu}_b \\ \boldsymbol{\mu}_c \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_b \\ \mathbf{m}_c + K_{cb} K_{bb}^{-1} (\boldsymbol{\mu}_b - \mathbf{m}_b) \end{bmatrix}, \quad (\text{B.173})$$

which equals the mean from (B.159). This shows that this merging of distributions gives exactly the same result and is hence equivalent. \square

The above theorem tells us that, by merging the distributions according to the described method, we silently also assume that \underline{x}_a and \underline{x}_c are conditionally independent given \underline{x}_b . This happens when we only use the posterior distribution of \underline{x}_b (ignoring our data about \underline{x}_a) to predict the posterior distribution of \underline{x}_c .

B.6.2. CONDITIONAL INDEPENDENCE BETWEEN VECTOR ELEMENTS

Previously we have assumed that \underline{x}_a and \underline{x}_c is conditionally independent given \underline{x}_b . We can take that one step further by assuming that every element of \underline{x}_a is also conditionally independent with respect to each other (and still with respect to \underline{x}_c) given \underline{x}_b . In other words, we assume that

$$f_{\underline{x}_a, \underline{x}_c | \underline{x}_b = \underline{x}_b}(\underline{x}_a, \underline{x}_c) = f_{\underline{x}_{a_1} | \underline{x}_b = \underline{x}_b}(x_{a_1}) \dots f_{\underline{x}_{a_{n_a}} | \underline{x}_b = \underline{x}_b}(x_{a_{n_a}}) f_{\underline{x}_c | \underline{x}_b = \underline{x}_b}(\underline{x}_c), \quad (\text{B.174})$$

where $x_{a_1}, \dots, x_{a_{n_a}}$ are the elements of \underline{x}_a and n_a denotes the size of \underline{x}_a . According to Theorem B.29, for Gaussian variables this assumption is equivalent to assuming that

$$K_{a_i a_j} = K_{a_i b} K_{bb}^{-1} K_{b a_j} \quad (\text{B.175})$$

for $i \neq j$. So the non-diagonal elements of K_{aa} are assumed to satisfy the above relation, while the non-diagonal elements of K_{aa} remain what they were. In other words, we assume that K_{aa} equals

$$K_{aa} = K_{ab} K_{bb}^{-1} K_{ba} + \text{diag}(K_{aa} - K_{ab} K_{bb}^{-1} K_{ba}) = K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa}, \quad (\text{B.176})$$

where we have defined the diagonal matrix Λ_{aa} as

$$\Lambda_{aa} \equiv \text{diag}(K_{aa} - K_{ab} K_{bb}^{-1} K_{ba}). \quad (\text{B.177})$$

Note that the above is the formal definition, but our assumptions also imply that

$$\Lambda_{aa} = K_{aa} - K_{ab} K_{bb}^{-1} K_{ba}. \quad (\text{B.178})$$

As such, we use the notation that $\Lambda_{pq} = K_{pq} - K_{pb} K_{bb}^{-1} K_{bq}$, for any sensible subscripts p and q .

The matrix Λ_{aa} also has an intuitive meaning. We know that, if \underline{x}_b is known deterministically, then the remaining variance of \underline{x}_a will equal $\Lambda_{aa} = K_{aa} - K_{ab} K_{bb}^{-1} K_{ba}$. In this expression, K_{aa} can be seen as ‘The structure which a priori is present within \underline{x}_a ’ while the intuitive meaning of $K_{ab} K_{bb}^{-1} K_{ba}$ is ‘The structure within \underline{x}_a which can be explained using knowledge from \underline{x}_b .’ As such, Λ_{aa} can be seen as ‘The part within the distribution of \underline{x}_a which can never be explained using knowledge from \underline{x}_b .’ Normally the matrix $K_{aa} - K_{ab} K_{bb}^{-1} K_{ba}$ is not necessarily diagonal, but if we assume that all elements of \underline{x}_a are conditionally independent given \underline{x}_b , then Λ_{aa} does become diagonal.

Subject to these extra assumptions, we can rederive the previous couple of theorems from Section B.6.1. We start with an equivalent version to Theorem B.30.

Theorem B.32. *Consider the assumptions of Theorem B.30. When we additionally assume that all elements of \underline{x}_a are conditionally independent given \underline{x}_b , the posterior distribution of $\underline{x}_{a,b}$ becomes*

$$\underline{x}_{a,b} \sim \mathcal{N}(\underline{x}_{a,b} | \boldsymbol{\mu}_{a,b}, \Sigma_{a,b}), \quad (\text{B.179})$$

$$\begin{aligned} \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix} &= \begin{bmatrix} \left((K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa})^{-1} + \hat{\Sigma}^{-1} \right)^{-1} & \hat{\Sigma} (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \\ K_{ba} (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} \hat{\Sigma} & K_{bb} - K_{ba} (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} &= \begin{bmatrix} \boldsymbol{m}_a + (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa}) (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} (\hat{\boldsymbol{\mu}} - \boldsymbol{m}_a) \\ \boldsymbol{m}_b + K_{ba} (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} (\hat{\boldsymbol{\mu}} - \boldsymbol{m}_a) \end{bmatrix}, \end{aligned}$$

while the posterior distribution of $\underline{x}_{b,c}$ remains (B.159).

Proof. The only difference between this theorem and Theorem B.30 is that we have assumed that (B.176). This immediately turns (B.158) into (B.179), while it does not affect (B.159) in any way. \square

B

The above theorem is not directly useful, because it does exactly the same as Theorem B.30. However, it does offer extra possibilities. In many applications of this theorem (see Section 4.1.4) the vector \underline{x}_a is much larger than the vector \underline{x}_b . As such, we would be better off inverting a matrix of size $n_b \times n_b$ than one of size $n_a \times n_a$. The following theorem tells us how we can arrange that.

Theorem B.33. Expression (B.179) of Theorem B.32 can be rewritten to

$$\begin{aligned} \underline{x}_{a,b} &\sim \mathcal{N}(\underline{x}_{a,b} | \boldsymbol{\mu}_{a,b}, \Sigma_{a,b}), \\ \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix} &= \begin{bmatrix} (\Lambda_{aa}^{-1} + \hat{\Sigma}^{-1})^{-1} + \hat{\Sigma} \hat{\Lambda}_{aa}^{-1} K_{ab} \Delta_{bb}^{-1} K_{ba} \hat{\Lambda}_{aa}^{-1} \hat{\Sigma} & \hat{\Sigma} \hat{\Lambda}_{aa}^{-1} K_{ab} \Delta_{bb}^{-1} K_{bb} \\ K_{bb} \Delta_{bb}^{-1} K_{ba} \hat{\Lambda}_{aa}^{-1} \hat{\Sigma} & K_{bb} \Delta_{bb}^{-1} K_{bb} \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} &= \begin{bmatrix} (\Lambda_{aa}^{-1} + \hat{\Sigma}^{-1})^{-1} (\Lambda_{aa}^{-1} \mathbf{m}_a + \hat{\Sigma}^{-1} \hat{\boldsymbol{\mu}}) + \hat{\Sigma} \hat{\Lambda}_{aa}^{-1} K_{ab} \Delta_{bb}^{-1} K_{ba} \hat{\Lambda}_{aa}^{-1} (\hat{\boldsymbol{\mu}} - \mathbf{m}_a) \\ \mathbf{m}_b + K_{bb} \Delta_{bb}^{-1} K_{ba} \hat{\Lambda}_{aa}^{-1} (\hat{\boldsymbol{\mu}} - \mathbf{m}_a) \end{bmatrix}, \end{aligned} \quad (\text{B.180})$$

where we have defined $\hat{\Lambda}_{aa}$ and Δ_{bb} as

$$\hat{\Lambda}_{aa} \equiv \Lambda_{aa} + \hat{\Sigma}, \quad (\text{B.181})$$

$$\Delta_{bb} \equiv K_{bb} + K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab}. \quad (\text{B.182})$$

Proof. To do this, we will rewrite each term of (B.179) separately. Starting with Σ_{bb} , we find through the matrix inversion lemma (Theorem A.7) that it equals

$$\begin{aligned} \Sigma_{bb} &= K_{bb} \left(K_{bb}^{-1} - K_{bb}^{-1} K_{ba} (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} K_{bb}^{-1} \right) K_{bb} \\ &= K_{bb} \left(K_{bb} + K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \right)^{-1} K_{bb}, \end{aligned} \quad (\text{B.183})$$

which equals $K_{bb} \Delta_{bb}^{-1} K_{bb}$, as we wanted to prove. We continue with Σ_{ab} which (through Theorem A.8) can be rewritten to

$$\begin{aligned} \Sigma_{ab} &= \hat{\Sigma} (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} K_{bb}^{-1} K_{bb} \\ &= \hat{\Sigma} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \Delta_{bb}^{-1} K_{bb}. \end{aligned} \quad (\text{B.184})$$

Identically, Σ_{ba} equals Σ_{ab}^T . That leaves Σ_{aa} . We can write this as $K_{aa} - K_{aa} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{aa}$, but identically we can also write it as

$$\begin{aligned} \Sigma_{aa} &= \hat{\Sigma} - \hat{\Sigma} (K_{ab} K_{bb}^{-1} K_{ba} + \Lambda_{aa} + \hat{\Sigma})^{-1} \hat{\Sigma} \\ &= (\Lambda_{aa} + \hat{\Sigma}) (\Lambda_{aa} + \hat{\Sigma})^{-1} \hat{\Sigma} - \hat{\Sigma} ((\Lambda_{aa} + \hat{\Sigma})^{-1} - (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \Delta_{bb}^{-1} K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1}) \hat{\Sigma} \\ &= \Lambda_{aa} (\Lambda_{aa} + \hat{\Sigma})^{-1} \hat{\Sigma} + \hat{\Sigma} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \Delta_{bb}^{-1} K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1} \hat{\Sigma}, \end{aligned} \quad (\text{B.185})$$

which equals the result from (B.180). That completes our proof for the covariance matrix and leaves us with the mean vector. The good news is that we can reuse some of our earlier results. For instance, for μ_b we have

$$\begin{aligned}\mu_b &= m_b + \Sigma_{ba} \hat{\Sigma}^{-1} (\hat{\mu} - m_a) \\ &= m_b + K_{bb} \Delta_{bb}^{-1} K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1} (\hat{\mu} - m_a).\end{aligned}\tag{B.186}$$

Similarly, for μ_a we have

$$\begin{aligned}\mu_a &= m_a + \Sigma_{aa} \hat{\Sigma}^{-1} (\hat{\mu} - m_a) \\ &= (\Lambda_{aa} + \hat{\Sigma}) (\Lambda_{aa} + \hat{\Sigma})^{-1} m_a \\ &\quad + (\Lambda_{aa} (\Lambda_{aa} + \hat{\Sigma})^{-1} + \hat{\Sigma} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \Delta_{bb}^{-1} K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1}) (\hat{\mu} - m_a) \\ &= \hat{\Sigma} (\Lambda_{aa} + \hat{\Sigma})^{-1} m_a + \Lambda_{aa} (\Lambda_{aa} + \hat{\Sigma})^{-1} \hat{\mu} + \hat{\Sigma} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \Delta_{bb}^{-1} K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1} (\hat{\mu} - m_a),\end{aligned}\tag{B.187}$$

which equals the result from (B.180), completing the proof. \square

In the above theorem, the expressions for the distribution of \underline{x}_b are surprisingly compact. But that's not their only upside. Their main advantage is that, to calculate the posterior distribution of \underline{x}_b , we only have to invert the diagonal matrices Λ_{aa} and $\hat{\Sigma}$ (and sums of them) and the relatively small matrix Δ_{bb} . This requires much less computational time than what we needed to do before.

The expressions describing the distribution of \underline{x}_a are more complicated though, but as such also more interesting. The expression for the mean μ_a consists of two parts. The first part $(\Lambda_{aa}^{-1} + \hat{\Sigma}^{-1})^{-1} (\Lambda_{aa}^{-1} m_a + \hat{\Sigma}^{-1} \hat{\mu})$ can be seen as the merger of the measurement $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$ and the part $\mathcal{N}(m_a, \Lambda_{aa})$ of the prior distribution of each element \underline{x}_{a_i} of \underline{x}_a for which \underline{x}_b can never provide information. The other part then is the information corresponding to \underline{x}_a which is passed through \underline{x}_b .

There is also a more intuitive view of the above theorem. What it basically comes down to, is that we first use the first measurement $\mathcal{N}(\hat{\mu}_1, \hat{\Sigma}_{1,1})$ to predict the distribution of \underline{x}_b . Then we use the second measurement $\mathcal{N}(\hat{\mu}_2, \hat{\Sigma}_{2,2})$ to predict \underline{x}_b all over again. We keep doing this, until we have n_a separate distributions of \underline{x}_b . We then merge all these distributions together and unmerge $(n_a - 1)$ times the prior distribution (which we have used n_a times; once in each of the n_a predictions) to get our final result. This process has been visualized in Figure 4.2. The result which we get is exactly the same as the result from Theorems B.32 and B.33. The following theorem proves this.

Theorem B.34. *The following merging of distributions*

B

$$\begin{aligned}
 & \mathcal{N} \left(\begin{bmatrix} m_{a_1} \\ * \\ \vdots \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} K_{a_1 a_1} & * & \cdots & K_{a_1 b} \\ * & \infty & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ K_{ba_1} & * & \cdots & K_{bb} \end{bmatrix} \right) \oplus \mathcal{N} \left(\begin{bmatrix} \hat{\mu}_1 \\ * \\ \vdots \\ * \end{bmatrix}, \begin{bmatrix} \hat{\Sigma}_{1,1} & * & \cdots & * \\ * & \infty & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & \infty \end{bmatrix} \right) \oplus \dots \\
 & \oplus \mathcal{N} \left(\begin{bmatrix} * \\ \vdots \\ m_{a_{n_a}} \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} \infty & \cdots & * & * \\ \vdots & \ddots & \vdots & \vdots \\ * & \cdots & K_{a_{n_a} a_{n_a}} & K_{a_{n_a} b} \\ * & \cdots & K_{ba_{n_a}} & K_{bb} \end{bmatrix} \right) \oplus \mathcal{N} \left(\begin{bmatrix} * \\ \vdots \\ \hat{\mu}_{n_a} \\ * \end{bmatrix}, \begin{bmatrix} \infty & \cdots & * & * \\ \vdots & \ddots & \vdots & \vdots \\ * & \cdots & \hat{\Sigma}_{n_a, n_a} & * \\ * & \cdots & * & \infty \end{bmatrix} \right) \\
 & \underbrace{\oplus \mathcal{N} \left(\begin{bmatrix} * \\ \vdots \\ * \\ \mathbf{m}_b \end{bmatrix}, \begin{bmatrix} \infty & \cdots & * & * \\ \vdots & \ddots & \vdots & \vdots \\ * & \cdots & \infty & * \\ * & \cdots & * & K_{bb} \end{bmatrix} \right)}_{(n_a - 1) \text{ times}}, \tag{B.188}
 \end{aligned}$$

which uses individual measurements of $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$, gives the same result as Theorems B.32 and B.33.

Proof. The key to proving this is to separately merge all prior distributions together and to merge all measured distributions together. If we merge the measured distributions (the rightmost distributions of (B.188)) we get, according to Theorem B.21, the distribution

$$\mathcal{N} \left(\begin{bmatrix} \hat{\mu} \\ * \end{bmatrix}, \begin{bmatrix} \hat{\Sigma} & * \\ * & \infty \end{bmatrix} \right). \tag{B.189}$$

Now let's look at what happens when we merge all the other (leftmost) distributions together. We start with the covariance matrix of this merger. When finding it, we will use Theorem A.6 to invert a blockwise matrix. We also note that per definition $\Lambda_{a_i a_i} = K_{a_i a_i} - K_{a_i b} K_{bb}^{-1} K_{ba_i}$. Theorem B.21 now implies that the covariance matrix of the merger

equals

$$\begin{aligned} & \left[\begin{array}{cccccc} \Lambda_{a_1 a_1}^{-1} & 0 & \cdots & -\Lambda_{a_1 a_1}^{-1} K_{a_1 b} K_{b b}^{-1} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -K_{b b}^{-1} K_{b a_1} \Lambda_{a_1 a_1}^{-1} & 0 & \cdots & K_{b b}^{-1} + K_{b b}^{-1} K_{b a_1} \Lambda_{a_1 a_1}^{-1} K_{a_1 b} K_{b b}^{-1} \end{array} \right] + \dots \quad (\text{B.190}) \\ & + \left[\begin{array}{cccccc} 0 & \cdots & 0 & & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \Lambda_{a_n a_n}^{-1} & -\Lambda_{a_n a_n}^{-1} K_{a_n b} K_{b b}^{-1} \\ 0 & \cdots & -K_{b b}^{-1} K_{b a_n} \Lambda_{a_n a_n}^{-1} & K_{b b}^{-1} + K_{b b}^{-1} K_{b a_n} \Lambda_{a_n a_n}^{-1} K_{a_n b} K_{b b}^{-1} \end{array} \right] \\ & - (n_a - 1) \left[\begin{array}{cccc} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & K_{b b}^{-1} \end{array} \right]^{-1} = \left[\begin{array}{cc} \Lambda_{a a}^{-1} & -\Lambda_{a a}^{-1} K_{a b} K_{b b}^{-1} \\ -K_{b b}^{-1} K_{b a} \Lambda_{a a}^{-1} & K_{b b}^{-1} + K_{b b}^{-1} K_{b a} \Lambda_{a a}^{-1} K_{a b} K_{b b}^{-1} \end{array} \right]. \end{aligned}$$

If we replace $\Lambda_{a a}$ in the above expression by

$$\Lambda_{a a} = \Lambda_{a a} + K_{a b} K_{b b}^{-1} K_{b a} - K_{a b} K_{b b}^{-1} K_{b a} = K_{a a} - K_{a b} K_{b b}^{-1} K_{b a}, \quad (\text{B.191})$$

then Theorem A.6 directly turns the above into

$$\begin{bmatrix} K_{a a} & K_{a b} \\ K_{b a} & K_{b b} \end{bmatrix}, \quad (\text{B.192})$$

subject to the assumption that the elements of \underline{x}_a are conditionally independent given \underline{x}_b . If we subsequently also apply Theorem B.21 to find the mean vector of the merger, we identically find that it will equal $\begin{bmatrix} \underline{m}_a \\ \underline{m}_b \end{bmatrix}$. So the merging of the leftmost matrices from (B.188) gives us the prior distribution of $\underline{x}_{a,b}$, while the merging of the rightmost matrices gives us the measured distribution. In other words, (B.188) is effectively the merging of the prior distribution of $\underline{x}_{a,b}$ together with the measured distribution. And it is this exact merger which is the starting point of Theorem B.22 through which Theorem B.32 is proven. This means that the outcome must also be the same, proving that (B.188) will result in (B.179) and equivalently also in (B.180). \square

B.6.3. CONDITIONAL INDEPENDENCE OF PARTS OF A VECTOR

So far we have studied conditional independence for \underline{x}_a as full vector (with respect to \underline{x}_c given \underline{x}_b) as well as conditional independence for the individual elements of \underline{x}_a . We can step in-between these ideas and split \underline{x}_a up into vector parts $\underline{x}_{a_1}, \dots, \underline{x}_{a_{n_p}}$, where n_p denotes the number of parts. These parts can be of equal size or of different size, as long as all parts together form \underline{x}_a without overlap.

We now assume that each of these parts $\underline{x}_{a_1}, \dots, \underline{x}_{a_{n_p}}$ as well as \underline{x}_c are conditionally independent given \underline{x}_b . How does this new assumption affect our theorems? The next theorem argues that it does not really affect anything at all, except for one minor definition change.

B

Theorem B.35. If we redefine Λ_{aa} to

$$\Lambda_{aa} = blkdiag(K_{aa} - K_{ab}K_{bb}^{-1}K_{ba}), \quad (\text{B.193})$$

then Theorems B.32 through B.34 still hold in exactly the same form.

B

Proof. The proofs of the given theorems are almost not affected by our new assumption. The only difference is that, instead of using individual elements $\underline{x}_{a_1}, \underline{x}_{a_2}, \dots$, we now use individual vector parts $\underline{x}_{a_1}, \underline{x}_{a_2}, \dots$. As a result, Λ_{aa} will not be diagonal but block-diagonal. Taking into account this difference, the proofs remain the same, implying that the theorems also hold in the same form. \square

B.6.4. ONLINE UPDATING OF DISTRIBUTIONS

Let's take a small step back. Suppose that all the elements of \underline{x}_a are conditionally independent, given \underline{x}_b , just like in Section B.6.2. Also suppose that we know the posterior distribution $\mathcal{N}(\boldsymbol{\mu}_b, \Sigma_{bb})$ of \underline{x}_b , after having done the measurement $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma})$.

What we will do now is add an extra point to \underline{x}_a . We write this extra point as \underline{x}_+ . It has $\underline{x}_+ \sim \mathcal{N}(m_+, K_{++})$ as prior distribution, and its prior covariance with \underline{x}_b is known to be K_{+b} . It is also assumed to be conditionally independent with respect to all the other elements of \underline{x}_a , and just like all other elements, we have also measured it, giving us a measured distribution of $\underline{x}_+ \sim \mathcal{N}(\hat{\mu}_+, \hat{\Sigma}_{++})$. The question now is 'What is the new posterior distribution of \underline{x}_b , given this new measurement?'

We could of course solve this by replacing

$$\begin{aligned} \underline{x}_a &\sim \mathcal{N}(\boldsymbol{m}_a, K_{aa}) \rightarrow \begin{bmatrix} \underline{x}_a \\ \underline{x}_+ \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m}_a \\ m_+ \end{bmatrix}, \begin{bmatrix} K_{aa} & K_{ab}K_{bb}^{-1}K_{ba} \\ K_{+b}K_{bb}^{-1}K_{ba} & K_{++} \end{bmatrix}\right), \\ \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma}) &\rightarrow \mathcal{N}\left(\begin{bmatrix} \hat{\boldsymbol{\mu}} \\ \hat{\mu}_+ \end{bmatrix}, \begin{bmatrix} \hat{\Sigma} & 0 \\ 0 & \hat{\Sigma}_{++} \end{bmatrix}\right), \end{aligned} \quad (\text{B.194})$$

and then redo all our calculations using Theorem B.32. However, given that all elements of \underline{x}_a are conditionally independent, given \underline{x}_b , we can do something more efficient; something that does not even require knowledge about the other elements of \underline{x}_a . We can update the distribution of \underline{x}_b directly! The following theorem tells us how.

Theorem B.36. Assume that all elements of \underline{x}_a and \underline{x}_+ are conditionally independent given \underline{x}_b . Also assume that the measurement $\underline{x}_a \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma})$ has given us a posterior distribution $\underline{x}_b \sim \mathcal{N}(\boldsymbol{\mu}_b, \Sigma_{bb})$, according to Theorem B.32. If we now also incorporate the measurement $\underline{x}_+ \sim \mathcal{N}(\hat{\mu}_+, \hat{\Sigma}_{++})$, then the posterior distribution of \underline{x}_b will equal

$$\underline{x}_b \sim \mathcal{N}(\boldsymbol{\mu}_b^+, \Sigma_{bb}^+), \quad (\text{B.195})$$

$$\Sigma_{bb}^+ = \Sigma_{bb} - \Sigma_{bb}K_{bb}^{-1}K_{b+}\Sigma_{++}^{-1}K_{+b}K_{bb}^{-1}\Sigma_{bb},$$

$$\boldsymbol{\mu}_b^+ = \boldsymbol{\mu}_b + \Sigma_{bb}K_{bb}^{-1}K_{b+}\Sigma_{++}^{-1}(\hat{\mu}_+ - (m_+ + K_{+b}K_{bb}^{-1}(\boldsymbol{\mu}_b - \boldsymbol{m}_b))),$$

where we have defined

$$\Sigma_{++} \equiv \Lambda_{++} + \hat{\Sigma}_{++} + K_{+b}K_{bb}^{-1}\Sigma_{bb}K_{bb}^{-1}K_{b+} \quad (\text{B.196})$$

$$= K_{++} + \hat{\Sigma}_{++} - K_{+b}K_{bb}^{-1}(K_{bb} - \Sigma_{bb})K_{bb}^{-1}K_{b+}. \quad (\text{B.197})$$

Proof. We will prove this using the relations for μ_b and Σ_{bb} from Theorem B.33. Let's start with the covariance matrix. Its old value was

$$\Sigma_{bb} = K_{bb} \left(K_{bb} + K_{ba} (\Lambda_{aa} + \hat{\Sigma})^{-1} K_{ab} \right)^{-1} K_{bb}. \quad (\text{B.198})$$

Personally, I'm kind of tired of writing $\hat{\Sigma}$ everywhere, so let's set up a new notation to prevent having to do that all the time. Let's define

$$\hat{K}_{aa} \equiv K_{aa} + \hat{\Sigma}, \quad (\text{B.199})$$

$$\hat{\Lambda}_{aa} \equiv \Lambda_{aa} + \hat{\Sigma} = K_{aa} + \hat{\Sigma} - K_{ab} K_{bb}^{-1} K_{ba}, \quad (\text{B.200})$$

$$\hat{K}_{++} \equiv K_{++} + \hat{\Sigma}_{++}, \quad (\text{B.201})$$

$$\hat{\Lambda}_{++} \equiv \Lambda_{++} + \hat{\Sigma}_{++} = K_{++} + \hat{\Sigma}_{++} - K_{+b} K_{bb}^{-1} K_{b+}. \quad (\text{B.202})$$

So a hat-symbol in this case means that the term $\hat{\Sigma}$ or $\hat{\Sigma}_{++}$ has been pulled in. Using this new notation, we can see that the new value of Σ_{bb} (written as Σ_{bb}^+) equals

$$\begin{aligned} \Sigma_{bb}^+ &= K_{bb} \left(K_{bb} + [K_{ba} \ K_{b+}] \begin{bmatrix} \hat{\Lambda}_{aa} & 0 \\ 0 & \hat{\Lambda}_{++} \end{bmatrix} \begin{bmatrix} K_{ab} \\ K_{b+} \end{bmatrix} \right)^{-1} K_{bb} \\ &= (K_{bb}^{-1} + K_{bb}^{-1} K_{ba} \hat{\Lambda}_{aa}^{-1} K_{ab} K_{bb}^{-1} + K_{bb}^{-1} K_{b+} \hat{\Lambda}_{++}^{-1} K_{+b} K_{bb}^{-1})^{-1} \\ &= (\Sigma_{bb}^{-1} + K_{bb}^{-1} K_{b+} \hat{\Lambda}_{++}^{-1} K_{+b} K_{bb}^{-1}). \end{aligned} \quad (\text{B.203})$$

We can rewrite it through the matrix inversion lemma (Theorem A.7). When we do, we directly find that

$$\begin{aligned} \Sigma_{bb}^+ &= \Sigma_{bb} - \Sigma_{bb} K_{bb}^{-1} K_{b+} (\hat{\Lambda}_{++} + K_{+b} K_{bb}^{-1} \Sigma_{bb} K_{bb}^{-1} K_{b+})^{-1} K_{+b} K_{bb}^{-1} \Sigma_{bb} \\ &= \Sigma_{bb} - \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} K_{+b} K_{bb}^{-1} \Sigma_{bb}, \end{aligned} \quad (\text{B.204})$$

where we have used (B.196). The above now equals the result from (B.195).

Next, we will prove the relation for the new mean μ_b^+ . We know from Theorem B.33 that the old value of μ_b was

$$\begin{aligned} \mu_b &= \mathbf{m}_b + K_{bb} (K_{bb} + K_{ba} \hat{\Lambda}_{aa}^{-1} K_{ab})^{-1} K_{ba} \hat{\Lambda}_{aa}^{-1} (\hat{\mu} - \mathbf{m}_a) \\ &= \mathbf{m}_b + \Sigma_{bb} K_{bb}^{-1} K_{ba} \hat{\Lambda}_{aa}^{-1} (\hat{\mu} - \mathbf{m}_a). \end{aligned} \quad (\text{B.205})$$

The new mean μ_b^+ now equals

$$\begin{aligned} \mu_b^+ &= \mathbf{m}_b + K_{bb} (K_{bb} + K_{ba} \hat{\Lambda}_{aa}^{-1} K_{ab} + K_{b+} \hat{\Lambda}_{++}^{-1} K_{b+})^{-1} (K_{ba} \hat{\Lambda}_{aa}^{-1} (\hat{\mu} - \mathbf{m}_a) + K_{b+} \hat{\Lambda}_{++}^{-1} (\hat{\mu}_+ - m_+)) \\ &= \mathbf{m}_b + \Sigma_{bb}^+ K_{bb}^{-1} (K_{bb} \Sigma_{bb}^{-1} (\mu_b - \mathbf{m}_b) + K_{b+} \hat{\Lambda}_{++}^{-1} (\hat{\mu}_+ - m_+)). \end{aligned} \quad (\text{B.206})$$

Note that we have used the new covariance Σ_{bb}^+ . If we insert the result that we just found, the above reduces to

$$\begin{aligned} \mu_b^+ &= \mathbf{m}_b + (\Sigma_{bb} - \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} K_{+b} K_{bb}^{-1} \Sigma_{bb}) \Sigma_{bb}^{-1} (\mu_b - \mathbf{m}_b) \\ &\quad + (\Sigma_{bb} - \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} K_{+b} K_{bb}^{-1} \Sigma_{bb}) K_{bb}^{-1} K_{b+} \hat{\Lambda}_{++}^{-1} (\hat{\mu}_+ - m_+) \\ &= \mathbf{m}_b - \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} K_{+b} K_{bb}^{-1} (\mu_b - \mathbf{m}_b) \\ &\quad + \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} (\Sigma_{++} - K_{+b} K_{bb}^{-1} \Sigma_{bb} K_{bb}^{-1} K_{b+}) \hat{\Lambda}_{++}^{-1} (\hat{\mu}_+ - m_+). \end{aligned} \quad (\text{B.207})$$

We can simplify this by applying (B.196). The above then reduces to

$$\boldsymbol{\mu}_b^+ = \boldsymbol{\mu}_b + \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} (\hat{\mu}_+ - m_+ - K_{+b} K_{bb}^{-1} (\boldsymbol{\mu}_b - \boldsymbol{m}_b)), \quad (\text{B.208})$$

which equals the result we wanted to obtain. \square

B

We have derived the above theorem in the setting of Section B.6.2: we have assumed conditional independence between all elements of \underline{x}_a as well as the new element \underline{x}_+ . We can also derive the theorem in the setting of Section B.6.3. That is, we assume conditional independence between the vector parts $\underline{x}_{a_1}, \dots, \underline{x}_{a_{n_p}}$ as well as the new vector \underline{x}_+ which we will add. If this new vector has a prior distribution of $\underline{x}_+ \sim \mathcal{N}(\boldsymbol{m}_+, K_{++})$ and a measured distribution of $\underline{x}_+ \sim \mathcal{N}(\hat{\mu}_+, \hat{\Sigma}_{++})$, then we can incorporate this in an identical way as we would incorporate a single point. We get the update law

$$\begin{aligned} \underline{x}_b &\sim \mathcal{N}(\boldsymbol{\mu}_b^+, \Sigma_{bb}^+), \\ \Sigma_{bb}^+ &= \Sigma_{bb} - \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} K_{+b} K_{bb}^{-1} \Sigma_{bb}, \\ \boldsymbol{\mu}_b^+ &= \boldsymbol{\mu}_b + \Sigma_{bb} K_{bb}^{-1} K_{b+} \Sigma_{++}^{-1} (\hat{\mu}_+ - (m_+ + K_{+b} K_{bb}^{-1} (\boldsymbol{\mu}_b - \boldsymbol{m}_b))). \end{aligned} \quad (\text{B.209})$$

This can be proven similarly to the proof of Theorem B.36. We just need to make a few adjustments because some previously scalar terms are not scalar anymore. These adjustments don't convey much any insight though, so I will omit them here.

Instead, I want to look at something else. We know it is possible to add a whole new vector part \underline{x}_+ to \underline{x}_a . But what do we do if we have a new point $\underline{x}_+ \sim \mathcal{N}(m_+, K_{++})$ which we want to add to an already existing part \underline{x}_{a_i} of \underline{x}_a ? That is, we do *not* assume that \underline{x}_a is conditionally independent given \underline{x}_b with respect to the other elements within \underline{x}_{a_i} . (We do assume that it is conditionally independent given \underline{x}_b with respect to the other parts of \underline{x}_a .) In this case, we do need to take into account the other elements of \underline{x}_{a_i} . How that works is explained by the following theorem.

Theorem B.37. *Assume that the parts $\underline{x}_{a_1}, \dots, \underline{x}_{a_{n_p}}$ of \underline{x}_a are conditionally independent given \underline{x}_b . Also assume that the measurement $\underline{x}_a \sim \mathcal{N}(\hat{\mu}, \hat{\Sigma})$ has given us a posterior distribution $\underline{x}_b \sim \mathcal{N}(\boldsymbol{\mu}_b, \Sigma_{bb})$, according to Theorem B.35. If we now add an element \underline{x}_+ to the vector part \underline{x}_{a_1} , such that \underline{x}_+ is conditionally independent given \underline{x}_b with respect to other parts of \underline{x}_a but not with respect to \underline{x}_{a_1} , and if we incorporate the measurement $\underline{x}_+ \sim \mathcal{N}(\hat{\mu}_+, \hat{\Sigma}_{++})$, then the posterior distribution of \underline{x}_b will equal*

$$\begin{aligned} \underline{x}_b &\sim \mathcal{N}(\boldsymbol{\mu}_b^+, \Sigma_{bb}^+), \\ \Sigma_{bb}^+ &= \Sigma_{bb} - \Sigma_{bb} K_{bb}^{-1} \tilde{K}_{b+} \tilde{\Sigma}_{++}^{-1} \tilde{K}_{+b} K_{bb}^{-1} \Sigma_{bb}, \\ \boldsymbol{\mu}_b^+ &= \boldsymbol{\mu}_b + \Sigma_{bb} K_{bb}^{-1} \tilde{K}_{b+} \tilde{\Sigma}_{++}^{-1} (\hat{\mu}_+ - (\tilde{m}_+ + \tilde{K}_{+b} K_{bb}^{-1} (\boldsymbol{\mu}_b - \boldsymbol{m}_b))), \end{aligned} \quad (\text{B.210})$$

where we have adjusted the definitions of Λ_{++} , K_{b+} , K_{+b} , Σ_{++} and m_+ according to

$$\tilde{\Lambda}_{++} \equiv \Lambda_{++} + \hat{\Sigma}_{++} - \Lambda_{+a_1} (\Lambda_{a_1 a_1} + \hat{\Sigma}_{a_1 a_1})^{-1} \Lambda_{a_1+}, \quad (\text{B.211})$$

$$\tilde{K}_{b+} \equiv K_{b+} - K_{ba_1} (\Lambda_{a_1 a_1} + \hat{\Sigma}_{a_1 a_1})^{-1} \Lambda_{a_1+} \equiv \tilde{K}_{+b}^T, \quad (\text{B.212})$$

$$\tilde{\Sigma}_{++} \equiv \tilde{\Lambda}_{++} + \tilde{K}_{+b} K_{bb}^{-1} \Sigma_{bb} K_{bb}^{-1} \tilde{K}_{b+}, \quad (\text{B.213})$$

$$\tilde{m}_+ \equiv m_+ + \Lambda_{+a_1} (\Lambda_{a_1 a_1} + \hat{\Sigma}_{a_1 a_1})^{-1} (\hat{\mu}_{a_1} - \boldsymbol{m}_{a_1}), \quad (\text{B.214})$$

and where the subscript a_1 refers to the part $\underline{x}_{\mathbf{a}_1}$ and not the element x_{a_1} .

Proof. The proof of this is very similar to the proof of Theorem B.36. In fact, we will again use the hat-notation from (B.199) to (B.202). Nevertheless, there are a few notable differences between the proofs.

These differences mainly concern the term $K_{bb} + K_{ba}\hat{\Lambda}_{aa}^{-1}K_{ab}$ within the expression for Σ_{bb} . Earlier, when the elements of $\underline{x}_{\mathbf{a}}$ were all conditionally independent with respect to each other, we could write this as

$$K_{bb} + K_{ba}\hat{\Lambda}_{aa}^{-1}K_{ab} = K_{bb} + \sum_{i=1}^{n_m} K_{ba_i}\hat{\Lambda}_{a_i a_i}K_{a_i b}, \quad (\text{B.215})$$

where we have

$$\hat{\Lambda}_{a_i a_i} = \hat{K}_{a_i a_i} - K_{a_i b}K_{bb}^{-1}K_{ba_i}. \quad (\text{B.216})$$

The main difference lies in that we are now not adding over individual elements, but over parts of $\underline{x}_{\mathbf{a}}$. That is, we now have

$$K_{bb} + K_{ba}\hat{\Lambda}_{aa}^{-1}K_{ab} = K_{bb} + \sum_{i=1}^{n_p} K_{ba_i}\hat{\Lambda}_{a_i a_i}K_{a_i b}, \quad (\text{B.217})$$

where the subscript a_i now does not relate to the element x_{a_i} but to the part $\underline{x}_{\mathbf{a}_i}$. This means that $\hat{\Lambda}_{a_i a_i}$ still satisfies (B.216), but is now a matrix instead of a scalar. In this case, if we add a single point \underline{x}_+ to $\underline{x}_{\mathbf{a}_1}$, the above becomes

$$\begin{aligned} K_{bb} + K_{ba}\hat{\Lambda}_{aa}^{-1}K_{ab} &\rightarrow K_{bb} + [K_{ba_1} \quad K_{b+}] \begin{bmatrix} \hat{\Lambda}_{a_1 a_1} & \Lambda_{a_1+} \\ \Lambda_{+a_1} & \hat{\Lambda}_{++} \end{bmatrix}^{-1} \begin{bmatrix} K_{a_1 b} \\ K_{+b} \end{bmatrix} + \left(\sum_{i=2}^{n_p} K_{ba_i} \hat{\Lambda}_{a_i a_i} K_{a_i b} \right), \\ &= K_{bb} + \left(\sum_{i=1}^{n_p} K_{ba_i} \hat{\Lambda}_{a_i a_i} K_{a_i b} \right) + [K_{ba_1} \quad K_{b+}] \begin{bmatrix} \hat{\Lambda}_{a_1 a_1} & \Lambda_{a_1+} \\ \Lambda_{+a_1} & \hat{\Lambda}_{++} \end{bmatrix}^{-1} \begin{bmatrix} K_{a_1 b} \\ K_{+b} \end{bmatrix} - K_{ba_1} \hat{\Lambda}_{a_1 a_1}^{-1} K_{a_1 b}. \end{aligned}$$

In the expression for Σ_{bb} , we used to have only the first two terms. To be precise, together they equaled $K_{bb}\Sigma_{bb}^{-1}K_{bb}$. The last two terms are new though, so let's focus on those. We will rewrite them, first using Theorem A.6 to expand the matrix inverse and then using definition B.211 of $\tilde{\Lambda}_{++}$ for easy notation. This will turn these last two terms into

$$\begin{aligned} &[K_{ba_1} \quad K_{b+}] \begin{bmatrix} \hat{\Lambda}_{a_1 a_1}^{-1} + \hat{\Lambda}_{a_1 a_1}^{-1} \Lambda_{a_1+} \tilde{\Lambda}_{++}^{-1} \Lambda_{+a_1} \hat{\Lambda}_{a_1 a_1}^{-1} & -\hat{\Lambda}_{a_1 a_1}^{-1} \Lambda_{a_1+} \tilde{\Lambda}_{++}^{-1} \\ -\tilde{\Lambda}_{++}^{-1} \Lambda_{+a_1} \hat{\Lambda}_{a_1 a_1}^{-1} & \tilde{\Lambda}_{++}^{-1} \end{bmatrix} \begin{bmatrix} K_{a_1 b} \\ K_{+b} \end{bmatrix} - K_{ba_1} \hat{\Lambda}_{a_1 a_1}^{-1} K_{a_1 b} \\ &= [K_{ba_1} \quad K_{b+}] \begin{bmatrix} -\hat{\Lambda}_{a_1 a_1}^{-1} \Lambda_{a_1+} \\ 1 \end{bmatrix} \tilde{\Lambda}_{++}^{-1} \begin{bmatrix} -\Lambda_{+a_1} \hat{\Lambda}_{a_1 a_1}^{-1} & 1 \end{bmatrix} \begin{bmatrix} K_{a_1 b} \\ K_{+b} \end{bmatrix} \\ &= (K_{b+} - K_{ba_1} \hat{\Lambda}_{a_1 a_1}^{-1} \Lambda_{a_1+}) \tilde{\Lambda}_{++}^{-1} (K_{+b} - \Lambda_{+a_1} \hat{\Lambda}_{a_1 a_1}^{-1} K_{a_1 b}) \\ &= \tilde{K}_{b+} \tilde{\Lambda}_{++}^{-1} \tilde{K}_{+b}. \end{aligned} \quad (\text{B.218})$$

It follows that the new covariance matrix Σ_{bb}^+ is given by

$$\Sigma_{bb}^+ = K_{bb} (K_{bb} + K_{ba}\hat{\Lambda}_{aa}^{-1}K_{ab} + \tilde{K}_{b+} \tilde{\Lambda}_{++}^{-1} \tilde{K}_{+b})^{-1} K_{bb}. \quad (\text{B.219})$$

We have already seen this expression before, in the proof of Theorem B.36, albeit without the tilde. As such, the result will also be the same, equaling the result from (B.210).

We can do something similar for the mean. The new mean $\boldsymbol{\mu}_b^+$ can be written as

B

$$\begin{aligned}
 \boldsymbol{\mu}_b^+ &= \boldsymbol{m}_b + \Sigma_{bb}^+ K_{bb}^{-1} \left(K_{ba} \hat{\Lambda}_{aa}^{-1} (\hat{\boldsymbol{\mu}} - \boldsymbol{m}_a) \right. \\
 &\quad \left. + [K_{ba_1} \quad K_{b+}] \begin{bmatrix} \hat{\Lambda}_{a_1 a_1} & \Lambda_{a_1+} \\ \Lambda_{+a_1} & \hat{\Lambda}_{++} \end{bmatrix}^{-1} \left(\begin{bmatrix} \hat{\boldsymbol{\mu}}_{\boldsymbol{a}_1} \\ \hat{\mu}_+ \end{bmatrix} - \begin{bmatrix} \boldsymbol{m}_{\boldsymbol{a}_1} \\ m_+ \end{bmatrix} \right) \right) - K_{ba_1} \hat{\Lambda}_{a_1 a_1}^{-1} (\hat{\boldsymbol{\mu}}_{\boldsymbol{a}_1} - \boldsymbol{m}_{\boldsymbol{a}_1}) \Big) \\
 &= \boldsymbol{m}_b + \Sigma_{bb}^+ K_{bb}^{-1} \left(K_{bb} \Sigma_{bb}^{-1} (\boldsymbol{\mu}_b - \boldsymbol{m}_b) \right. \\
 &\quad \left. + [K_{ba_1} \quad K_{b+}] \begin{bmatrix} -\hat{\Lambda}_{a_1 a_1}^{-1} \Lambda_{a_1+} \\ 1 \end{bmatrix} \tilde{\Lambda}_{++}^{-1} [-\Lambda_{+a_1} \hat{\Lambda}_{a_1 a_1}^{-1} \quad 1] \begin{bmatrix} \hat{\boldsymbol{\mu}}_{\boldsymbol{a}_1} - \boldsymbol{m}_{\boldsymbol{a}_1} \\ \hat{\mu}_+ - m_+ \end{bmatrix} \right) \\
 &= \boldsymbol{m}_b + \Sigma_{bb}^+ K_{bb}^{-1} (K_{bb} \Sigma_{bb}^{-1} (\boldsymbol{\mu}_b - \boldsymbol{m}_b) + \tilde{K}_{b+} \tilde{\Lambda}_{++}^{-1} (\hat{\mu}_+ - \tilde{m}_+)) .
 \end{aligned} \tag{B.220}$$

we have seen this relation before at (B.206). Because of this, the result must also be the same, equaling the result from (B.210). \square

The adjustments we have made to Λ_{++} , K_{b+} , K_{+b} , Σ_{++} and m_+ can be seen as compensations to take into account the fact that $\underline{x}_{\boldsymbol{a}_1}$ and \underline{x}_+ are not conditionally independent given \underline{x}_b . If they are, then we will have $\Lambda_{+a_1} = \Lambda_{a_1+} = 0$ and the above Theorem reduces to Theorem B.36.

We should note here that it is not only possible to add new elements \underline{x}_+ to the part $\underline{x}_{\boldsymbol{a}_1}$, but also to any other parts $\underline{x}_{\boldsymbol{a}_i}$. I just picked $\underline{x}_{\boldsymbol{a}_1}$ to make our notation a bit easier. We should also keep in mind though, that the bigger $\underline{x}_{\boldsymbol{a}_i}$ is, the longer the update will take. It does help if we keep track of the matrix $(\Lambda_{a_i a_i} + \hat{\Sigma}_{a_i a_i})^{-1}$ as it grows, but still we should make sure that the parts $\underline{x}_{\boldsymbol{a}_i}$ do not grow too large or our algorithm will become slow.

C

LINEAR SYSTEMS THEORY

Summary — A linear system is a system in which the state derivative linearly depends on the state itself and possibly on the input and on process noise. When no noise is present, the state evolution can be calculated analytically. When Gaussian white noise is present, the state will be a Gaussian random variable, whose mean and covariance can be calculated analytically at every point in time.

For such systems, we can define a quadratic cost function, completing the Linear Quadratic Gaussian (LQG) set-up. This cost function can optionally also be discounted over time. The resulting cost will not have a Gaussian distribution but a generalized noncentral χ^2 -distribution. It is possible to calculate the expected value as well as the cost variance analytically.

When an input is present, it is also possible to find the optimal control law minimizing the expected cost. This optimal control law will be linear and can be calculated analytically as well. In addition, when a measurement equation including measurement noise is present, an observer can be set up for the system to approximate the system state. When this observer is optimal – minimizing the steady-state error covariance – then the separation principle can be applied. We can feed this state estimate to the optimal control law and optimally control the system in this way.

'Optimal' here still means minimizing the expected cost. When the goal is not to minimize the expected cost, but to minimize the chance that the cost exceeds a given threshold, a different control strategy may need to be applied. It may then be better to for instance reduce the cost variance.

In this appendix we look at linear systems. Specifically, we look at linear systems subject to Gaussian process/measurement noise and with a quadratic cost function that needs to be minimized. The resulting subject is known as *Linear Quadratic Gaussian* (LQG) control.

We start by looking at how the state evolves over time for such systems (Section C.1). We then look at how we can calculate the expected value of the cost (Section C.2). Subsequently we add input, which allows us to minimize this cost (Section C.3). Eventually we further study the cost function, mostly examining expressions for its variance (Section C.4). At the end we set up a few applications of the derived methods (Section C.5) and look at the state of the literature (Section C.6).

C

C.1. LINEAR SYSTEMS AND THEIR EVOLUTION

We will first examine the definition of a linear system, as we will use it here (Section C.1.1). Then we look at how the distribution of the system state evolves over time (Section C.1.2). These ideas are far from novel and can for instance be found in [Anderson and Moore \(1990\)](#), [Skogestad and Postlethwaite \(2005\)](#), [Bosgra et al. \(2008\)](#).

C.1.1. SYSTEM DEFINITION

In this appendix we study continuous-time linear systems subject to process noise. There are multiple ways to write such a system and we start by examining two different such methods.

We first look at the most common notation in control theory applications. It is used by for instance [Skogestad and Postlethwaite \(2005\)](#). Here we write our system as

$$\dot{\underline{x}}(t) = A\underline{x}(t) + \underline{v}(t), \quad (\text{C.1})$$

where $\underline{x}(t)$ is the *state* and A is the *system matrix*. Additionally, $\underline{v}(t)$ is the *process noise*, where the underline means it is a random variable. (Appendix B can tell you more about random variables.)

We assume that $\underline{v}(t)$ is zero-mean Gaussian white noise with intensity V . The ‘Gaussian’ part implies that $\underline{v}(t)$ is a time-dependent Gaussian process, the ‘zero-mean’ part implies that the mean function $\underline{m}(t) = 0$ equals zero and the ‘white noise’ part implies that $\underline{v}(t)$ and $\underline{v}(t')$ are uncorrelated. In other words, the covariance function of $\underline{v}(t)$ is $k(t, t') = V\delta(t - t')$, with $\delta(\cdot)$ the (*Dirac Delta function*).

The problem which many mathematicians have with this formulation is that the signal $\underline{v}(t)$ cannot exist. It is not measurable with nonzero probability, because effectively the variance $k(t, t)$ is infinite. In addition, it is not continuous because $\underline{v}(t)$ and $\underline{v}(t')$ are not correlated, even when t and t' are nearly equal.

The issues behind this are well outlined in [Øksendal \(1985\)](#). This book also suggests a more formal way of writing linear systems. While many control engineers prefer the above notation, most formal mathematicians would rather write a linear system as

$$d\underline{x}(t) = A\underline{x}(t) dt + d\underline{w}(t), \quad (\text{C.2})$$

where $\underline{w}(t)$ is a vector of *Brownian motions*. The above can subsequently be evaluated using Itô integrals. You can learn more about those in the book by [Øksendal \(1985\)](#), or if you are enthusiastic in the original work by [Itô \(1951\)](#).

But what is a Brownian motion? I do not want to go into depth on the details here – for a full treatise, see Mörters and Peres (2010) – but I will give a short summary. You can intuitively see a Brownian motion $\underline{\mathbf{w}}(t)$ as a ‘random walk’. At $t = 0$ we start at $\underline{\mathbf{w}}(t) = \mathbf{0}$. Then, at every point in time, we take a small step. The size of this step is related to the motion parameter W , but the direction is random. We may happen to walk in the same direction as previously, or go back. As a result, the mean $\mathbb{E}[\underline{\mathbf{w}}(t)]$ of our position remains zero. However, as time passes, the distance we are likely to be from our starting position increases, causing the variance $\mathbb{E}[\underline{\mathbf{w}}(t)\underline{\mathbf{w}}^T(t)]$ to increase linearly as Wt .

If we compare (C.1) with (C.2), we see that $\underline{\mathbf{v}}(t)$ actually equals $\frac{d\underline{\mathbf{w}}(t)}{dt}$. And if we integrate the Gaussian process $\underline{\mathbf{v}}(t)$ using the techniques from Section 2.5.4 to get $\underline{\mathbf{w}}(t)$, we find that $\underline{\mathbf{w}}(t)$ actually is a Gaussian process with mean function

$$\mathbf{m}(t) = \int_0^t 0 \, ds = 0 \quad (\text{C.3})$$

and covariance function

$$k(t, t') = \int_0^t \int_0^{t'} V \delta(s - s') \, ds' \, ds = \int_0^{\min(t, t')} V \, ds = V \min(t, t'), \quad (\text{C.4})$$

which happens to be the covariance function of a Brownian motion. (Note that $k(t, t) = Vt$, like we saw earlier.) So if we brush complaints from formal mathematicians aside, both methods come down to exactly the same.

Which method will we use then? Evaluating (C.2) requires the application of Itô integrals, which requires some complicated mathematics. Evaluating (C.1) can be done in a more straightforward manner. As such, we will stick with the linear system (C.1), as is used by most control engineers.

C.1.2. EVOLUTION OF THE SYSTEM STATE

Consider system (C.1). Suppose that we put the system in an *initial state* $\mathbf{x}(0) \equiv \mathbf{x}_0$. If we know the process noise $\mathbf{v}(t)$ deterministically at each point in time, how will the state $\mathbf{x}(t)$ then evolve? That is outlined by the following theorem.

Theorem C.1. Consider the linear differential equation (C.1), with $\mathbf{v}(t)$ known. Its solution $\mathbf{x}(t)$ is given by

$$\mathbf{x}(t) = e^{At} \mathbf{x}_0 + \int_0^t e^{A(t-s)} \mathbf{v}(s) \, ds. \quad (\text{C.5})$$

Proof. We solve this using the method of the integrating factor. We multiply both sides of the system equation by e^{-At} . This turns it into

$$e^{-At} \dot{\mathbf{x}}(t) - e^{-At} A\mathbf{x}(t) = e^{-At} \mathbf{v}(t). \quad (\text{C.6})$$

The left side can be rewritten as a derivative, according to

$$\frac{d}{dt} (e^{-At} \mathbf{x}(t)) = e^{-At} \mathbf{v}(t). \quad (\text{C.7})$$

Let's replace t by s . If we subsequently integrate over s , from 0 to t , we find that

$$[e^{-As} \mathbf{x}(s)]_0^t = \int_0^t e^{-As} \mathbf{v}(s) ds. \quad (\text{C.8})$$

This can be extended into

$$e^{-At} \mathbf{x}(t) - \mathbf{x}(0) = \int_0^t e^{-As} \mathbf{v}(s) ds. \quad (\text{C.9})$$

C

Solving for $\mathbf{x}(t)$ by left-multiplying by e^{At} will result in (C.5). \square

Now let's suppose that we do not know the initial state $\underline{\mathbf{x}}_0$ exactly. Instead, we will treat it as a random variable $\underline{\mathbf{x}}_0$, which we assume to be Gaussian. We also do not know the process noise $\underline{\mathbf{v}}(t)$, except for its general properties. In this case the state $\underline{\mathbf{x}}(t)$ also becomes a random variable $\underline{\mathbf{x}}(t)$ at each point in time. What can we now say about its properties?

Theorem C.2. *Consider the linear differential equation (C.1). If the initial state satisfies $\underline{\mathbf{x}}_0 \sim \mathcal{N}(\underline{\boldsymbol{\mu}}_0, \Sigma_0)$ and $\underline{\mathbf{v}}(t)$ is zero-mean Gaussian white noise with intensity V , then $\underline{\mathbf{x}}(t)$ has a Gaussian distribution $\mathcal{N}(\underline{\boldsymbol{\mu}}(t), \Sigma(t))$ at each point in time, with*

$$\underline{\boldsymbol{\mu}}(t) = e^{At} \underline{\boldsymbol{\mu}}_0, \quad (\text{C.10})$$

$$\Sigma(t) = e^{At} (\Sigma_0 - X^V) e^{A^T t} + X^V, \quad (\text{C.11})$$

and where X^V is the solution to the Lyapunov equation $AX^V + X^V A^T + V = 0$. (For further details on X^V , see Appendix A.4.)

Proof. Our starting point here is Theorem C.1. We know that both $\underline{\mathbf{x}}_0$ and $\underline{\mathbf{v}}(t)$ are Gaussian parameters. Since adding up Gaussian parameters will result in a new Gaussian parameter (this follows from Theorem B.13), $\underline{\mathbf{x}}(t)$ will be Gaussian at each point in time. And to fully specify $\underline{\mathbf{x}}(t)$, we only need to find the mean vector and the covariance matrix for each time t .

To find the mean, we take the expectation of (C.5). This results in

$$\underline{\boldsymbol{\mu}}(t) = \mathbb{E}[\underline{\mathbf{x}}(t)] = \mathbb{E}\left[e^{At} \underline{\mathbf{x}}_0 + \int_0^t e^{A(t-s)} \underline{\mathbf{v}}(s) ds\right]. \quad (\text{C.12})$$

The expectation operator is a linear operator. (See for instance Theorems B.3 and B.4.) As such, we can apply it separately to both of the above terms, as well as pull it within the integral. This gives us

$$\underline{\boldsymbol{\mu}}(t) = e^{At} \mathbb{E}[\underline{\mathbf{x}}_0] + \int_0^t e^{A(t-s)} \mathbb{E}[\underline{\mathbf{v}}(s)] ds = e^{At} \underline{\boldsymbol{\mu}}_0, \quad (\text{C.13})$$

where we have used $\mathbb{E}[\underline{\mathbf{v}}(t)] = 0$.

To find the covariance, we apply similar steps. We start with

$$\Sigma(t) = \mathbb{E}\left[(\underline{\mathbf{x}}(t) - \mathbb{E}[\underline{\mathbf{x}}(t)]) (\underline{\mathbf{x}}(t) - \mathbb{E}[\underline{\mathbf{x}}(t)])^T\right]. \quad (\text{C.14})$$

Next, from (C.5) and (C.10) it follows that

$$\underline{x}(t) - \mathbb{E}[\underline{x}(t)] = e^{At}(\underline{x}(0) - \boldsymbol{\mu}(0)) + \int_0^t e^{A(t-s)} \underline{v}(s) ds. \quad (\text{C.15})$$

We should keep in mind here that $\underline{v}(s)$ is white noise. This means that it is not correlated with any past states $\underline{x}(t)$ with $t \leq s$. (If $t > s$, then $\underline{x}(t)$ will of course directly depend on $\underline{v}(s)$.) Hence, $\mathbb{E}[\underline{v}(t)(\underline{x}(0) - \boldsymbol{\mu}(0))^T] = 0$. Using this, we can turn (C.14) into

$$\begin{aligned} \Sigma(t) &= \mathbb{E}\left[e^{At}(\underline{x}(0) - \boldsymbol{\mu}(0))(\underline{x}(0) - \boldsymbol{\mu}(0))^T e^{A^T t}\right] + \left(\int_0^t e^{A(t-s)} \underline{v}(s) ds\right) \left(\int_0^t e^{A(t-s)} \underline{v}(s) ds\right)^T \\ &= e^{At} \mathbb{E}\left[(\underline{x}(0) - \boldsymbol{\mu}(0))(\underline{x}(0) - \boldsymbol{\mu}(0))^T\right] e^{A^T t} \\ &\quad + \int_0^t \int_0^t e^{A(t-s_1)} \mathbb{E}[\underline{v}(s_1) \underline{v}^T(s_2)] e^{A^T(t-s_2)} ds_2 ds_1. \end{aligned} \quad (\text{C.16})$$

We can immediately note that the first of these expectations equals the initial state covariance matrix Σ_0 . The second expectation $\mathbb{E}[\underline{v}(s_1) \underline{v}^T(s_2)]$ per definition equals $V\delta(s_1 - s_2)$. This term only has a value whenever $s_1 = s_2$, and in this case the integral over $\delta(s_1 - s_2)$ equals one. As such, we can rewrite the above to

$$\Sigma(t) = e^{At} \Sigma_0 e^{A^T t} + \int_0^t e^{A(t-s_1)} V e^{A^T(t-s_1)} ds_1. \quad (\text{C.17})$$

For the mathematicians: this reduction of $\mathbb{E}[\underline{v}(s_1) \underline{v}^T(s_2)]$ to $V\delta(s_1 - s_2)$ is formally an application of the *Itô isometry*, as explained in Øksendal (1985).

Within the above integral we can substitute s_1 for $t - s$. This means that $ds_1 = -ds$ which adds a minus sign to the integral. However, while s_1 ranges from 0 to t , we have s ranging from t to 0. So the integral limits are reversed. Reversing the integral limits is equivalent to adding a minus sign to the integral, which tells us that

$$\int_0^t e^{A(t-s_1)} V e^{A^T(t-s_1)} ds_1 = \int_t^0 -e^{As} V e^{A^Ts} ds = \int_0^t e^{As} V e^{A^Ts} ds. \quad (\text{C.18})$$

This integral per definition equals $X^V(t)$ (see definition (A.119)) and Theorem A.26 claims that this in turn equals $X^V - e^{At} X^V e^{A^T t}$. As a result, we have

$$\Sigma(t) = e^{At} \Sigma_0 e^{A^T t} + X^V - e^{At} X^V e^{A^T t}. \quad (\text{C.19})$$

Rewriting this would immediately turn it into (C.11), completing the proof. \square

The above theorem tells us something interesting. If the system is stable, then as $t \rightarrow \infty$ we have $e^{At} \rightarrow 0$. In other words, $\Sigma(t) \rightarrow X^V$. The Lyapunov solution X^V is therefore known as the *steady-state state covariance matrix*. Given process noise with intensity V , the state of a stable system will eventually have a covariance of X^V .

The time-dependent matrix $\Sigma(t)$ itself also satisfies an interesting differential equation, as shown by the next theorem.

C

Theorem C.3. *The state covariance $\Sigma(t)$ satisfies*

$$\dot{\Sigma}(t) = A\Sigma(t) + \Sigma(t)A^T + V. \quad (\text{C.20})$$

Proof. We prove this by taking the derivative of (C.11). If we apply the relation $\frac{d}{dt}e^{At} = Ae^{At} = e^{At}A$, then

$$\dot{\Sigma}(t) = Ae^{At}(\Sigma_0 - X^V)e^{A^T t} + e^{At}(\Sigma_0 - X^V)e^{A^T t}A^T. \quad (\text{C.21})$$

C

The term $e^{At}(\Sigma_0 - X^V)e^{A^T t}$ equals $\Sigma(t) - X^V$. Applying this gives us

$$\begin{aligned}\dot{\Sigma}(t) &= A(\Sigma(t) - X^V) + (\Sigma(t) - X^V)A^T \\ &= A\Sigma(t) + \Sigma(t)A^T - (AX^V + X^VA^T).\end{aligned} \quad (\text{C.22})$$

If we then note that, per definition, $AX^V + X^VA^T + V = 0$, we directly find (C.20). \square

Finally, we can also define the covariance between $\underline{x}(t_1)$ and $\underline{x}(t_2)$ as

$$\Sigma(t_1, t_2) \equiv \mathbb{E}[(\underline{x}(t_1) - \mathbb{E}[\underline{x}(t_1)])(\underline{x}(t_2) - \mathbb{E}[\underline{x}(t_2)])^T]. \quad (\text{C.23})$$

How to calculate this covariance is explained by the following theorem.

Theorem C.4. *For $t_1 \leq t_2$, the state covariance $\Sigma(t_1, t_2)$ defined by (C.23) satisfies*

$$\Sigma(t_1, t_2) = e^{At_1}(\Sigma_0 - X^V)e^{A^T t_2} + X^Ve^{A^T(t_2-t_1)}. \quad (\text{C.24})$$

In addition, we have $\Sigma(t_1, t_2) = \Sigma^T(t_2, t_1)$ and $\Sigma(t, t) = \Sigma(t)$.

Proof. The proof of this is nearly identical to that of Theorem C.2. That is, we can find in an identical way that

$$\Sigma(t_1, t_2) = e^{At_1}\Sigma_0e^{A^T t_2} + \int_0^{t_1} \int_0^{t_2} e^{A(t_1-s_1)}Ve^{A^T(t_2-s_2)}ds_2 ds_1. \quad (\text{C.25})$$

We know that $\delta(s_1 - s_2)$ only has a nonzero value when $s_1 = s_2$. And because $0 \leq s_1 \leq t_1$ and $0 \leq s_2 \leq t_2$, this only occurs whenever $s_1 = s_2 \leq \min(t_1, t_2)$. We have assumed that $t_1 \leq t_2$, so $\min(t_1, t_2) = t_1$. This means that we do not need to consider values of $s_1 = s_2$ larger than t_1 in either of our integrals. If we then also solve the inner integral, we find

$$\Sigma(t_1, t_2) = e^{At_1}\Sigma_0e^{A^T t_2} + \int_0^{t_1} e^{A(t_1-s_1)}Ve^{A^T(t_2-s_1)}ds_1. \quad (\text{C.26})$$

Similarly to (C.18), we can rewrite the integral in the above expression to

$$\begin{aligned}\int_0^{t_1} e^{A(t_1-s_1)}Ve^{A^T(t_2-s_1)}ds_1 &= \left(\int_0^{t_1} e^{A(t_1-s_1)}Ve^{A^T(t_1-s_1)}ds_1 \right) e^{A(t_2-t_1)} \\ &= \left(\int_0^{t_1} e^{As}Ve^{A^Ts}ds \right) e^{A(t_2-t_1)} = X^V(t_1)e^{A(t_2-t_1)}.\end{aligned} \quad (\text{C.27})$$

Applying Theorem A.26 once more to rewrite $X^V(t_1)$ as $X^V - e^{At_1}X^Ve^{A^Tt_1}$ will turn the expression for $\Sigma(t_1, t_2)$ into

$$\Sigma(t_1, t_2) = e^{At_1}\Sigma_0e^{A^Tt_2} + X^Ve^{A(t_2-t_1)} - e^{At_1}X^Ve^{A^Tt_2}. \quad (\text{C.28})$$

Rewriting this further will give us (C.24), proving the first claim of this theorem.

The two other claims from the theorem, $\Sigma(t_1, t_2) = \Sigma^T(t_2, t_1)$ and $\Sigma(t, t) = \Sigma(t)$, both follow directly from definition (C.23). The first of these two claims does tell us what to do when $t_1 \geq t_2$. In that case, we can find $\Sigma(t_2, t_1)$ and transpose the result. Hence, for $t_2 \geq t_1$, we have

$$\Sigma(t_1, t_2) = e^{At_1}(\Sigma_0 - X^V)e^{A^Tt_2} + e^{A(t_1-t_2)}X^V. \quad (\text{C.29})$$

Although we could have also derived this in the same way as how we found (C.28). \square

Finally, there is another quantity which we will often see. It is the *expected squared state*, defined through

$$\Psi(t) \equiv \mathbb{E}[\mathbf{x}(t)\mathbf{x}^T(t)] = \Sigma(t) + \boldsymbol{\mu}(t)\boldsymbol{\mu}^T(t), \quad (\text{C.30})$$

$$\Psi(t_1, t_2) \equiv \mathbb{E}[\mathbf{x}(t_1)\mathbf{x}^T(t_2)] = \Sigma(t_1, t_2) + \boldsymbol{\mu}(t_1)\boldsymbol{\mu}^T(t_2). \quad (\text{C.31})$$

It satisfies many of the expressions which the covariance matrix $\Sigma(t)$ also satisfies.

Theorem C.5. *The expected squared state $\Psi(t)$, defined by (C.30), satisfies*

$$\Psi(t) = e^{At}(\Psi_0 - X^V)e^{A^Tt} + X^V, \quad (\text{C.32})$$

$$\dot{\Psi}(t) = A\Psi(t) + \Psi(t)A^T + V, \quad (\text{C.33})$$

$$\Psi(t_1, t_2) = e^{At_1}(\Psi_0 - X^V)e^{A^Tt_2} + X^Ve^{A^T(t_2-t_1)}. \quad (\text{C.34})$$

where the last expression only holds when $t_1 \leq t_2$. For $t_1 \geq t_2$, we have $\Psi(t_1, t_2) = \Psi^T(t_2, t_1)$.

Proof. We will start with the last of the three expressions in the theorem. Using definition (C.31) and Theorem C.4, we discover that for $t_1 \leq t_2$ we have

$$\begin{aligned} \Psi(t_1, t_2) &= \Sigma(t_1, t_2) + \boldsymbol{\mu}(t_1)\boldsymbol{\mu}^T(t_2) \\ &= e^{At_1}(\Sigma_0 - X^V)e^{A^Tt_2} + X^Ve^{A^T(t_2-t_1)} + e^{At_1}\boldsymbol{\mu}_0\boldsymbol{\mu}_0^Te^{A^Tt_2} \\ &= e^{At_1}(\Sigma_0 + \boldsymbol{\mu}_0\boldsymbol{\mu}_0^T - X^V)e^{A^Tt_2} + X^Ve^{A^T(t_2-t_1)}, \end{aligned} \quad (\text{C.35})$$

which equals (C.34). When $t_1 \geq t_2$ we can use $\Psi(t_1, t_2) = \Psi^T(t_2, t_1)$, which follows directly from definition (C.31). To prove (C.32) we can insert $t_1 = t_2 = t$ into the above expression and apply $\Psi(t, t) = \Psi(t)$. Finally, we can prove (C.33) in an identical way as we proved Theorem C.3, except with Σ replaced by Ψ . \square

Now that we know everything about how the state evolves, it is time to add a cost function and then add input to the system allowing us to minimize this cost function.

C.2. THE EXPECTED COST

In the section after this one we will add control to our linear system. We then want to find the optimal controller. But what makes a controller optimal?

The answer is that it minimizes a cost function, and that is why we study cost functions first. In this function we will look at various cost functions and find expressions for the mean values.

We first do so for the infinite-time cost function (Section C.2.1). We will then see that the noise causes this cost to become infinitely large. We work around this by either looking at a system without noise (Section C.2.2), looking at a finite-time cost function (Section C.2.3) or looking at a discounted cost function (Section C.2.4).

C.2.1. THE INFINITE-TIME COST FUNCTION

In this thesis we will always use a quadratic cost function. That is,

$$\underline{J} = \int_0^\infty \underline{x}^T(t) Q \underline{x}(t) dt. \quad (\text{C.36})$$

Here \underline{J} is the cost and Q is the *state penalty matrix*. It is a positive semi-definite matrix, ensuring that \underline{J} cannot be negative.

It is important to note that the state $\underline{x}(t)$ is a random variable. After all, its initial state $\underline{x}_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ has a Gaussian distribution, and it is continually disturbed by noise. This means that \underline{J} is a random variable too. Its value is not set in stone based on initial conditions but depends on what initial state and what noise we actually get. We can calculate its expected value $\mathbb{E}[\underline{J}]$ though.

To do so, we will use the trace operator. (For background on the trace operator, see Appendix A.1.1.) Note that, because \underline{J} is a scalar, we have $\underline{J} = \text{tr}(\underline{J})$. Also note that the trace operator, the expectation operator and the integration operator are linear operators, meaning we can interchange the order in which they are applied. This tells us that

$$\begin{aligned} \mathbb{E}[\underline{J}] &= \mathbb{E}\left[\int_0^\infty \text{tr}(\underline{x}^T(t) Q \underline{x}(t)) dt\right] \\ &= \int_0^\infty \mathbb{E}[\text{tr}(\underline{x}(t) \underline{x}^T(t) Q)] dt \\ &= \int_0^\infty \text{tr}(\mathbb{E}[\underline{x}(t) \underline{x}^T(t)] Q) dt \\ &= \int_0^\infty \text{tr}(\Psi(t) Q) dt. \end{aligned} \quad (\text{C.37})$$

Applying Theorem C.5 will turn this into

$$\begin{aligned} \mathbb{E}[\underline{J}] &= \int_0^\infty \text{tr}\left(\left(e^{At}(\Psi_0 - X^V)e^{A^T t} + X^V\right) Q\right) dt \\ &= \int_0^\infty \left(\text{tr}(X^V Q) + \text{tr}\left((\Psi_0 - X^V)e^{A^T t} Q e^{At}\right)\right) dt. \end{aligned} \quad (\text{C.38})$$

There is one problem in this expression. The first term within the above integral is constant. And when we integrate over a constant term, for an infinitely long duration, we get an infinite cost. What is going on here?

The idea is that the state $\underline{\mathbf{x}}$ is perpetually disturbed by the noise $\underline{\nu}$. It does not converge to zero, but it will wind up with a steady-state covariance matrix $\Sigma(t) \rightarrow X^V$. It hence perpetually contributes to the cost.

Optimizing a cost which is infinite does not work, so we need to fix this issue. There are multiple ways to do so. We could assume that there is no noise ($V = 0$), we could look at the finite-time cost or we could introduce a discount exponent. We will look at each of these cases one by one.

C.2.2. THE COST OF A SYSTEM WITHOUT NOISE

When there is no noise, the only randomness is caused by the initial state $\mathbf{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$. The resulting expected cost is then given by the following theorem.

Theorem C.6. *Assume that A is stable and that there is no noise ($V = 0$). The expected infinite-time cost (C.36) is given by*

$$\mathbb{E}[\underline{J}] = \text{tr}(\Psi_0 \bar{X}^Q). \quad (\text{C.39})$$

Proof. Our starting point is (C.38). When $V = 0$ we also have $X^V = 0$ and as a result we find that

$$\mathbb{E}[\underline{J}] = \int_0^\infty \text{tr}\left(\Psi_0 e^{A^T t} Q e^{At}\right) dt = \text{tr}\left(\Psi_0 \int_0^\infty e^{A^T t} Q e^{At} dt\right) = \text{tr}(\Psi_0 \bar{X}^Q), \quad (\text{C.40})$$

where in the last step we have applied Theorem A.25. Note that \bar{X}^Q per definition is the solution to the alternate Lyapunov equation $A^T \bar{X}^Q + \bar{X}^Q A + Q = 0$. (See (A.115).) \square

We can also rewrite (C.39) into

$$\mathbb{E}[\underline{J}] = \text{tr}(\Sigma_0 \bar{X}^Q) + \boldsymbol{\mu}_0^T \bar{X}^Q \boldsymbol{\mu}_0. \quad (\text{C.41})$$

This shows how the initial state mean $\boldsymbol{\mu}_0$ and the initial state covariance Σ_0 separately contribute to the expected cost $\mathbb{E}[\underline{J}]$.

C.2.3. THE FINITE-TIME COST FUNCTION

Instead of letting the cost integral run up to $t = \infty$, we can also let it run up to a finite time T . In that case we get the *finite-time cost function*

$$\underline{J}_T = \int_0^T \underline{\mathbf{x}}^T(t) Q \underline{\mathbf{x}}(t) dt. \quad (\text{C.42})$$

The expected cost is now given by the following theorem.

Theorem C.7. *Assume that A is Sylvester. The expected finite-time cost (C.42) is given by*

$$\mathbb{E}[\underline{J}_T] = \text{tr}((\Psi_0 - \Psi(T) + TV) \bar{X}^Q). \quad (\text{C.43})$$

Proof. Our starting point again is (C.38). With the finite-time integral, it now equals

$$\begin{aligned}\mathbb{E}[\underline{J}_T] &= \int_0^T \text{tr}(X^V Q) dt + \text{tr}\left((\Psi_0 - X^V)\left(\int_0^T e^{A^T t} Q e^{At} dt\right)\right) \\ &= \text{tr}(X^V Q) T + \text{tr}((\Psi_0 - X^V) \bar{X}^Q(T)),\end{aligned}\quad (\text{C.44})$$

where we have applied definition (A.120) to turn the integral into $\bar{X}^Q(T)$. We can expand $\bar{X}^Q(T)$ using Theorem A.26. If we also use Theorem A.29 to rewrite the first term, the above turns into

$$\begin{aligned}\mathbb{E}[\underline{J}_T] &= \text{tr}(V \bar{X}^Q) T + \text{tr}\left((\Psi_0 - X^V)(\bar{X}^Q - e^{A^T T} \bar{X}^Q e^{AT})\right) \\ &= \text{tr}(TV \bar{X}^Q) + \text{tr}\left((\Psi_0 - X^V - e^{AT} (\Psi_0 - X^V) e^{AT}) \bar{X}^Q\right) \\ &= \text{tr}(TV \bar{X}^Q) + \text{tr}((\Psi_0 - \Psi(T)) \bar{X}^Q),\end{aligned}\quad (\text{C.45})$$

which equals (C.43). \square

It is interesting to see the effect of both Ψ_0 and V . If we start in the steady-state distribution, with $\Psi_0 = X^V$, then $\Psi(T)$ will also remain equal to X^V . (Keep in mind that $\Psi(T) \rightarrow X^V$ as $T \rightarrow \infty$.) The result will be a cost of $\text{tr}(TV \bar{X}^Q)$. This is the direct cost due to the noise.

If we start in a different initial state, this effectively changes the cost by an amount of $\text{tr}((\Psi_0 - \Psi(T)) \bar{X}^Q)$. This change can increase the cost (when $\Psi_0 > X^V$) or decrease the cost (when $\Psi_0 < X^V$).

Also note that, as $T \rightarrow \infty$, we have $\mathbb{E}[\underline{J}_T] \rightarrow \infty$ as well. The cost becomes infinite. An interesting topic here is the rate at which it becomes infinite, especially after the first transients of \underline{x} have disappeared. This *steady-state cost rate* is defined as

$$\lim_{T \rightarrow \infty} \frac{d\underline{J}_T}{dT} = \lim_{T \rightarrow \infty} \frac{d}{dT} \int_0^T \underline{x}(t) Q \underline{x}(t) dt = \lim_{T \rightarrow \infty} \underline{x}(T) Q \underline{x}(T).\quad (\text{C.46})$$

We can now see, both from the above expression and from (C.43), that the expected steady-state cost rate equals

$$\mathbb{E}\left[\lim_{T \rightarrow \infty} \frac{d\underline{J}_T}{dT}\right] = \lim_{T \rightarrow \infty} \frac{d\mathbb{E}[\underline{J}_T]}{dT} = \lim_{T \rightarrow \infty} \text{tr}(\Psi(T) Q) = \text{tr}(V \bar{X}^Q) = \text{tr}(X^V Q),\quad (\text{C.47})$$

where in the last step we have applied Theorem A.29.

C.2.4. THE DISCOUNTED COST FUNCTION

Another way to ensure that we get a finite cost is to add a discount exponent. This results in the *discounted cost function*

$$\underline{J} = \int_0^\infty e^{2\alpha t} \underline{x}^T(t) Q \underline{x}(t) dt,\quad (\text{C.48})$$

$$\underline{J}_T = \int_0^T e^{2\alpha t} \underline{x}^T(t) Q \underline{x}(t) dt.\quad (\text{C.49})$$

The parameter α can be positive or negative. If it is negative, like in many applications related to reinforcement learning in which future costs/rewards are discounted, it is known as the *discount exponent*. (See for instance [Sutton and Barto \(1998\)](#), [Bertsekas and Tsitsiklis \(1996\)](#).) If it is positive, like in various linear systems theory applications, it is called the *prescribed degree of stability*. (For more background on this, see [Anderson and Moore \(1990\)](#), [Bosgra et al. \(2008\)](#).) Why it is called the prescribed degree of stability will become clear later, after Theorem C.13.

It is important to note that the discounted cost function is actually a generalization of the regular cost function. After all, when we set $\alpha = 0$, we wind up with the regular one. In the rest of this appendix, we will always indicate which cost function we work with and under what assumptions.

So given this new cost function, what can we say about the expected cost? That is explained by the following two theorems, the first for the finite-time cost and the second for the infinite-time cost.

Theorem C.8. *Assume that $\alpha \neq 0$ and that A and A_α are both Sylvester. The expected discounted finite-time cost (C.49) is given by*

$$\mathbb{E}[\underline{J}] = \text{tr}\left(\left(\Psi_0 - e^{2\alpha T} \Psi(T) + (1 - e^{2\alpha T})\left(\frac{-V}{2\alpha}\right)\right) \bar{X}_\alpha^Q\right). \quad (\text{C.50})$$

Proof. We can prove this theorem in the same way as Theorem C.7, albeit with more bookkeeping. However, for fun we will prove it in a very different and slightly more elegant way.

Our starting point now is (C.37), which we adjust by taking into account the factor $e^{2\alpha t}$ and the upper integral limit T . We will write it as

$$\mathbb{E}[I_T] = \text{tr}\left(\left(\int_0^T e^{2\alpha t} \Psi(t) dt\right) Q\right) = \text{tr}(YQ), \quad (\text{C.51})$$

where we have defined the integral as Y . The key now is to find Y . We do so by using (C.33) from Theorem C.5. We multiply it by $e^{2\alpha t}$ and integrate it to find

$$\begin{aligned} \int_0^T e^{2\alpha t} \dot{\Psi}(t) dt &= A \left(\int_0^T e^{2\alpha t} \Psi(t) dt \right) + \left(\int_0^T e^{2\alpha t} \Psi(t) dt \right) A^T + \int_0^T e^{2\alpha t} V dt \\ &= AY + YA^T + (1 - e^{2\alpha T}) \left(\frac{-V}{2\alpha}\right). \end{aligned} \quad (\text{C.52})$$

The left part of the above equation can also be solved through integration by parts. This turns it into

$$\int_0^T e^{2\alpha t} \dot{\Psi}(t) dt = [e^{2\alpha t} \Psi(t)]_0^T - 2\alpha \int_0^T e^{2\alpha t} \Psi(t) dt = e^{2\alpha T} \Psi(T) - \Psi_0 - 2\alpha Y. \quad (\text{C.53})$$

By merging the above two expressions, using $A_\alpha \equiv A + \alpha I$, we wind up with

$$A_\alpha Y + YA_\alpha^T + \Psi_0 - e^{2\alpha T} \Psi(T) + (1 - e^{2\alpha T}) \left(\frac{-V}{2\alpha}\right) = 0. \quad (\text{C.54})$$

This is a Lyapunov equation which we can solve for Y . It follows that Y equals the Lyapunov solution $X_\alpha^{\Psi_0 - e^{2\alpha T} \Psi(T) + (1 - e^{2\alpha t}) \left(\frac{-V}{2\alpha}\right)}$. Since this term is a nightmare to write, we rewrite it using Theorems A.27 and A.28 into

$$Y = X_\alpha^{\Psi_0 - e^{2\alpha T} \Psi(T) + (1 - e^{2\alpha t}) \left(\frac{-V}{2\alpha}\right)} = X_\alpha^{\Psi_0} - e^{2\alpha T} X_\alpha^{\Psi(T)} + (1 - e^{2\alpha t}) \frac{-X_\alpha^V}{2\alpha}. \quad (\text{C.55})$$

Through Theorem A.29 we can now write our solution for $\mathbb{E}[\underline{J}]$ as

$$\begin{aligned} \mathbb{E}[\underline{J}] &= \text{tr} \left(\left(X_\alpha^{\Psi_0} - e^{2\alpha T} X_\alpha^{\Psi(T)} + (1 - e^{2\alpha t}) \frac{-X_\alpha^V}{2\alpha} \right) Q \right) \\ &= \text{tr} \left(\left(\Psi_0 - e^{2\alpha T} \Psi(T) + (1 - e^{2\alpha t}) \left(\frac{-V}{2\alpha}\right) \right) \bar{X}_\alpha^Q \right), \end{aligned} \quad (\text{C.56})$$

which completes our proof. \square

We have actually solved the more difficult problem of the finite-time cost first. This allows us to treat the more simple infinite-time cost as a special case of the finite-time cost.

Theorem C.9. *Assume that $\alpha < 0$ and that A_α is stable. The expected discounted infinite-time cost (C.48) is given by*

$$\mathbb{E}[\underline{J}] = \text{tr} \left(\left(\Psi_0 - \frac{V}{2\alpha} \right) \bar{X}_\alpha^Q \right). \quad (\text{C.57})$$

Proof. We consider Theorem C.8 as $T \rightarrow \infty$. Because $\alpha < 0$ we have $e^{2\alpha T} \rightarrow 0$. In addition, for stable A_α it also holds that

$$e^{2\alpha T} \Psi(T) = e^{A_\alpha T} (\Psi_0 - X^V) e^{A_\alpha^T T} + e^{2\alpha T} X^V \rightarrow 0. \quad (\text{C.58})$$

This implies that (C.50) directly turns into (C.57), completing the proof. \square

It is interesting to note that Theorem C.8 also turns into Theorem C.7 when $\alpha \rightarrow 0$. To see why, we should realize that, according to l'Hôpital's rule, we have

$$\lim_{\alpha \rightarrow 0} \frac{1 - e^{2\alpha T}}{2\alpha} = \lim_{\alpha \rightarrow 0} \frac{\frac{d}{d\alpha}(1 - e^{2\alpha T})}{\frac{d}{d\alpha}(2\alpha)} = \lim_{\alpha \rightarrow 0} \frac{-2Te^{2\alpha T}}{2} = -T. \quad (\text{C.59})$$

It is also fun to insert the steady-state state distribution $\Psi_0 = X^V$ into Theorem C.9 and see what happens. In this case, using $V = -AX^V - X^V A^T$, we find that

$$\begin{aligned} \mathbb{E}[\underline{J}] &= \text{tr} \left(\left(X^V - \frac{V}{2\alpha} \right) \bar{X}_\alpha^Q \right) \\ &= \frac{1}{2\alpha} \text{tr} \left((2\alpha X^V + AX^V + X^V A^T) \bar{X}_\alpha^Q \right) \\ &= \frac{1}{2\alpha} \text{tr} \left((A_\alpha X^V + X^V A_\alpha^T) \bar{X}_\alpha^Q \right) \\ &= \frac{1}{2\alpha} \text{tr} \left(X^V (A_\alpha^T \bar{X}_\alpha^Q + \bar{X}_\alpha^Q A_\alpha) \right) \\ &= -\frac{1}{2\alpha} \text{tr} (X^V Q), \end{aligned} \quad (\text{C.60})$$

which is what we can expect when $\Psi(t) = X^V$ for any t .

So now we know how to find the mean of the cost for various different cost functions. It is also possible to find the variance of the cost, which we will look at in Section C.4. But first we will add an input to the system and see how we can use it to minimize the (expected) cost.

C.3. LINEAR QUADRATIC GAUSSIAN CONTROL

It is time to add an input to the system. This requires us to come up with a control law too. The fundamental question is ‘Which control law can minimize the (expected) cost?’

To investigate this, we start by examining a system without any process noise. We first do this for the non-discounted cost function (Section C.3.1) and then extend the ideas to the discounted cost function (Section C.3.2). Then we reintroduce the process noise, as well as an uncertain initial state, and look at how this affects the situation (Section C.3.3). Finally we add measurement noise as well. We look at how we can then estimate the state (Section C.3.4) and then use this estimate to optimally control the system (Section C.3.5).

C.3.1. THE INPUT THAT OPTIMIZES THE COST FUNCTION

We start off by considering a relatively simple case. Consider the linear system with a *system input* $\mathbf{u}(t)$ but without process noise

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t). \quad (\text{C.61})$$

We also assume that we know the initial state \mathbf{x}_0 . We want to control this system in some optimal way, with optimal meaning it minimizes the cost J . Here, we also want to penalize excessive inputs, and so we use the cost function

$$J = \int_0^\infty (\mathbf{x}^T(t)Q\mathbf{x}(t) + \mathbf{u}^T(t)R\mathbf{u}(t)) dt. \quad (\text{C.62})$$

We have already seen the positive semi-definite state penalty matrix Q . Now we also have a positive definite *input penalty matrix* R . Both matrices are symmetric.

To optimally control the system, we now need to find a control law $\mathbf{u}(t) = \boldsymbol{\pi}^*(\mathbf{x}(t))$ which always results in a cost that is lower or equal to the cost resulting from any other control law $\boldsymbol{\pi}(\mathbf{x}(t))$, irrespective of which initial state \mathbf{x}_0 the system starts from. What would such an optimal control law look like? The following theorem tells us that it at least must be linear in the state.

Theorem C.10. *Consider the linear system (C.61). Assume that there is a control law $\mathbf{u}(t) = \boldsymbol{\pi}(\mathbf{x}(t))$ which can stabilize the system. Then there exists at least one optimal control law $\mathbf{u}(t) = \boldsymbol{\pi}^*(\mathbf{x}(t))$ which minimizes the quadratic cost function (C.62). Furthermore, of all the optimal control laws, there is always at least one which is linear in $\mathbf{x}(t)$; so of the form $\boldsymbol{\pi}^*(\mathbf{x}(t)) = -F\mathbf{x}(t)$. The resulting optimal cost function J^* is quadratic in the initial state \mathbf{x}_0 ; so of the form $J^*(\mathbf{x}_0) = \mathbf{x}_0^T \bar{X} \mathbf{x}_0$.*

Proof. The first part of the theorem claims that, for a stabilizable system, there is an optimal control law. The reason for this is that, for a stabilizing control law $\boldsymbol{\pi}(\mathbf{x}(t))$, the cost $J(\mathbf{x}_0)$ is finite but positive. As such, it must have a minimum, and the corresponding

control law $\pi^*(\mathbf{x}(t))$ resulting in this minimum is the optimal control law. Of course it may happen that there are multiple control laws resulting in the same optimal cost function $J^*(\mathbf{x}_0)$.

To prove that the cost is quadratic in the initial state, we are going to do a thought experiment. This thought experiment actually consists of three simulation runs, which are visualized in the left part of figure C.1.

C

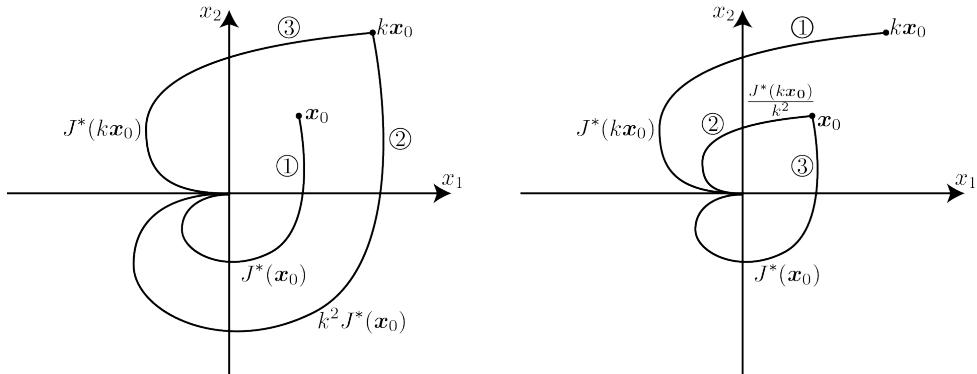


Figure C.1: A graphical illustration of the thought experiment which proves the cost function is quadratic in the initial state.

1. Suppose that we know some optimal control law $\mathbf{u}(t) = \pi^*(\mathbf{x}(t))$, which is not necessarily linear. For our first simulation run, we put the system in some initial state \mathbf{x}_0 and run this control law π^* . We keep track of the state and denote the resulting state progression by $\mathbf{x}_1(t)$, with $\mathbf{x}_1(0) = \mathbf{x}_0$. We also remember exactly which input $\mathbf{u}_1(t)$ we applied at each time t . At the end of our experiment, we have accumulated the (optimal) cost $J_1 = J^*(\mathbf{x}_0)$.
2. For our second experiment, we are going to scale the previous experiment. That is, we are going to start in an initial state $\mathbf{x}_2(0) = k\mathbf{x}_0$, with k a nonzero number. We then apply the control input $\mathbf{u}_2(t) = k\mathbf{u}_1(t)$. Now something interesting happens. Because the system is linear, we will have $\mathbf{x}_2(t) = k\mathbf{x}_1(t)$ for all future times t . In other words, everything is k times as large! As a result, we know that the cost J_2 which we accumulate will equal $k^2 J_1 = k^2 J(\mathbf{x}_0)$.
3. For our third experiment, we again start in $\mathbf{x}_3(0) = k\mathbf{x}_0$, yet this time we simply apply our optimal control law $\mathbf{u} = \pi^*(\mathbf{x})$. The resulting cost will necessarily be optimal and will equal $J_3 = J^*(k\mathbf{x}_0)$.

Now compare experiments 2 and 3. Both experiments had the same initial state, and in experiment 3 the cost was optimal. This means that we must have $J_2 \geq J_3$, or

$$k^2 J^*(\mathbf{x}_0) \geq J^*(k\mathbf{x}_0). \quad (\text{C.63})$$

Next, we can do another set of three experiments, but now with the set-up as shown in the right part of figure C.1. That is, we first start in $\mathbf{x}_1(0) = k\mathbf{x}_0$ and apply $\mathbf{u}_1(t) = \pi^*(\mathbf{x}(t))$.

Then we start in $\mathbf{x}_2(0) = \mathbf{x}_0$ and apply $\mathbf{u}_2(t) = \frac{1}{k}\mathbf{u}_1(t)$. Finally we start in $\mathbf{x}_3(0) = \mathbf{x}_0$ and apply $\mathbf{u}(t) = \boldsymbol{\pi}^*(\mathbf{x}(t))$. This set of experiments tells us that

$$\frac{1}{k^2} J^*(k\mathbf{x}_0) \geq J^*(\mathbf{x}_0). \quad (\text{C.64})$$

If we combine the above two equations, then we find that equality must hold. That is,

$$k^2 J^*(\mathbf{x}_0) = J^*(k\mathbf{x}_0). \quad (\text{C.65})$$

From this we can conclude that, when the initial state \mathbf{x}_0 becomes k times as large, then the optimal cost J^* becomes k^2 as large. In other words, the cost function is quadratic in \mathbf{x}_0 . (Technically, for the multivariate case, there is an extra condition which must be met. For details on this, see [Anderson and Moore \(1990\)](#), section 2.3.)

In addition, we have seen that when we take linear combinations of an optimal control law, we still wind up with an optimal control law. We can use this to show that when there is an optimal control law, there is also at least one optimal control law which is linear. To do that, we can actually take an optimal control law $\boldsymbol{\pi}^*(\mathbf{x}(t))$ and construct a linear optimal control law from it. First, we define the *unit vectors* $\mathbf{e}_1, \mathbf{e}_2, \dots$ as

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}, \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}, \dots \quad (\text{C.66})$$

We then define the matrix F as

$$F = -[\boldsymbol{\pi}^*(\mathbf{e}_1) \quad \boldsymbol{\pi}^*(\mathbf{e}_2) \quad \dots \quad \boldsymbol{\pi}^*(\mathbf{e}_n)], \quad (\text{C.67})$$

and use the linear control law $\boldsymbol{\pi}(\mathbf{x}(t)) = -F\mathbf{x}(t)$. Because linear combinations of optimal control laws also result in optimal control laws, this control law must be optimal. This proves that there is an optimal control law which is linear. \square

The above theorem has told us that, to find the optimal control law, it is enough to focus on control laws of the form $\mathbf{u}(t) = -F\mathbf{x}(t)$, with F the *feedback matrix*. But what is the *optimal feedback matrix* \check{F} , minimizing the cost function?

Theorem C.11. *Consider the linear system $\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$ with feedback law $\mathbf{u}(t) = -F\mathbf{x}(t)$. Assume that it is stabilizable: there is a matrix F for which $A - BF$ is stable. The optimal feedback matrix \check{F} minimizing the cost function (C.62) is given by*

$$\check{F} \equiv R^{-1}B^T\check{X}, \quad (\text{C.68})$$

where the optimal cost matrix \check{X} is the solution to the Riccati equation

$$A^T\check{X} + \check{X}A + Q - \check{X}BR^{-1}B^T\check{X} = 0. \quad (\text{C.69})$$

The resulting cost of the system subject to an initial state \mathbf{x}_0 equals

$$J = \mathbf{x}_0^T\check{X}\mathbf{x}_0. \quad (\text{C.70})$$

Proof. Subject to the feedback law, the system behaves as $\dot{\mathbf{x}}(t) = (A - BF)\mathbf{x}(t)$. The solution for $\mathbf{x}(t)$ will hence be

$$\mathbf{x}(t) = e^{(A-BF)t} \mathbf{x}_0. \quad (\text{C.71})$$

The cost function now becomes

$$\begin{aligned} J &= \int_0^\infty \mathbf{x}^T(t)(Q + F^T RF)\mathbf{x}(t) dt \\ &= \mathbf{x}_0^T \left(\int_0^\infty e^{(A-BF)^T t} (Q + F^T RF) e^{(A-BF)t} dt \right) \mathbf{x}_0. \end{aligned} \quad (\text{C.72})$$

C

We know from Theorem A.25 that, when $A - BF$ is stable, the above integral equals $\bar{X}_{A-BF}^{Q+F^T RF}$, which is per definition the solution to the Lyapunov equation

$$(A - BF)^T \bar{X}_{A-BF}^{Q+F^T RF} + \bar{X}_{A-BF}^{Q+F^T RF} (A - BF) + (Q + F^T RF) = 0. \quad (\text{C.73})$$

For ease of notation, we will write $\bar{X}_{A-BF}^{Q+F^T RF}$ just as \bar{X} . The cost J now equals

$$J = \mathbf{x}_0^T \bar{X} \mathbf{x}_0. \quad (\text{C.74})$$

The matrix \bar{X} here is paramount to the cost function J , which is why we call it the *cost matrix*. We now want to find for which feedback matrix F the above cost is minimized. We should realize here that, if we choose F , we directly also find \bar{X} from (C.73).

Now let's ask ourselves, for which F do we get a cost matrix \bar{X} such that $F = R^{-1}B^T\bar{X}$? To answer this question, we rewrite (C.73) by completing the squares with respect to F . (See Theorem C.12 for details on how to do this.) The result becomes

$$(F - R^{-1}B^T\bar{X})^T R(F - R^{-1}B^T\bar{X}) + Q + A^T\bar{X} + \bar{X}A - \bar{X}BR^{-1}B^T\bar{X} = 0. \quad (\text{C.75})$$

If we have picked our feedback matrix F such that $F = R^{-1}B^T\bar{X}$, then the first term will be zero. It follows that \bar{X} satisfies the Riccati equation (C.69) and hence $\bar{X} = \check{X}$, implying that $F = \check{F} = R^{-1}B^T\check{X}$.

But what if $F \neq R^{-1}B^T\bar{X}$? In this case the first term is not zero but, because R is positive definite, it must be positive definite or positive semi-definite matrix. The crucial thing though, is that this matrix will be directly added to Q . So picking the feedback matrix F to be unequal to $R^{-1}B^T\bar{X}$ is equivalent to increasing the state penalty matrix Q by a positive semi-definite matrix. Naturally, increasing the state penalty matrix Q by a positive semi-definite matrix cannot decrease the cost. It only increases it for many (if not all) initial states \mathbf{x}_0 .

From this we conclude that the optimal F must satisfy $F = R^{-1}B^T\bar{X}$. The corresponding \bar{X} must then satisfy the Riccati equation (C.69). So if we take the solution \check{X} from the Riccati equation and set up $\check{F} = R^{-1}B^T\check{X}$, we get an optimal feedback matrix. \square

Theorem C.12. *The Lyapunov equation*

$$(A - BF)^T \bar{X} + \bar{X}(A - BF) + (Q + F^T RF) = 0 \quad (\text{C.76})$$

can be rewritten, by completing the squares with respect to F , to

$$(F - R^{-1}B^T\bar{X})^T R(F - R^{-1}B^T\bar{X}) + A^T\bar{X} + \bar{X}A + Q - \bar{X}BR^{-1}B^T\bar{X} = 0. \quad (\text{C.77})$$

Proof. We want to rewrite the first equation to a quadratic form

$$(F - T_1)^T T_2 (F - T_1) + T_3 = 0, \quad (\text{C.78})$$

where T_1 , T_2 and T_3 are terms which we still need to find, but none of them have F in them. To find T_1 , T_2 and T_3 , we expand the above to

$$F^T T_2 F - F^T T_2 T_1 - T_1^T T_2 F + T_1^T T_2 T_1 + T_3 = 0. \quad (\text{C.79})$$

Now we will compare equations (C.76) and (C.79). By looking at all the terms that have two F 's in them, we can immediately see that $T_2 = R$. Then, by comparing all terms that have only one F in them, we find that

$$T_1^T T_2 F = \bar{X} B F. \quad (\text{C.80})$$

Although there might be multiple values of T_1 which satisfy this equation, we are sure that one of them equals

$$T_1 = (\bar{X} B T_2^{-1})^T = R^{-1} B^T \bar{X}. \quad (\text{C.81})$$

Remember that both R and \bar{X} are assumed to be symmetric, while B is not.

Finally, by looking at all terms that are so far unaccounted for, we find that

$$T_1^T T_2 T_1 + T_3 = A^T \bar{X} + \bar{X} A + Q. \quad (\text{C.82})$$

This implies that

$$T_3 = A^T \bar{X} + \bar{X} A + Q - T_1^T T_2 T_1 = A^T \bar{X} + \bar{X} A + Q - \bar{X} B R^{-1} R R^{-1} B^T \bar{X}. \quad (\text{C.83})$$

Now that we have T_1 , T_2 and T_3 , we can plug them into equation (C.78) to get (C.77). \square

It is important to note the order of the steps when finding F and \bar{X} . If we choose any (sub-optimal) feedback matrix F , then we can find the cost matrix \bar{X} within the cost function $J = \mathbf{x}_0^T \bar{X} \mathbf{x}_0$ by solving the Lyapunov equation (C.73). However, if we want to find the optimal control law, we first have to find the optimal cost matrix \check{X} and then use (C.68) to find the optimal feedback matrix \check{F} . For this feedback matrix \check{F} , the optimal cost matrix \check{X} of course also satisfies the Lyapunov matrix (C.73).

Also, keep in mind that this is the feedback matrix minimizing the *infinite-time* cost function. It is not the feedback matrix minimizing the *finite-time* cost function. In fact, optimizing the cost for a finite-time experiment is a more complicated problem, because you also need to take into account the time you still have left in the experiment. If you are just starting your experiment, it is worthwhile to apply significant inputs to improve the state. However, when you are nearing the end, applying inputs will mostly be useless. It will cost you due to R , but there will not be enough time to reap the benefits from the improved state. We will not consider the problem of analytically optimizing the finite-time cost.

C.3.2. DIFFERENCES FOR THE DISCOUNTED COST FUNCTION

In Section C.2.4 we have studied the discounted cost function (C.48). Suppose that we would also introduce such a discount here, getting the cost function

$$J = \int_0^\infty e^{2\alpha t} (\mathbf{x}^T(t) Q \mathbf{x}(t) + \mathbf{u}^T(t) R \mathbf{u}(t)) dt. \quad (\text{C.84})$$

How would this affect the optimal feedback matrix \check{F} ?

Theorem C.13. Consider the linear system $\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$ with feedback law $\mathbf{u}(t) = -F\mathbf{x}(t)$. Assume that it is stabilizable up to degree α : there is a matrix F_α for which $A_\alpha - BF_\alpha$ is stable. The optimal feedback matrix \check{F}_α minimizing the discounted cost function (C.84) is given by

$$\check{F}_\alpha \equiv R^{-1} B^T \check{X}_\alpha, \quad (\text{C.85})$$

where the optimal discounted cost matrix \check{X}_α is the solution to the Riccati equation

$$A_\alpha^T \check{X}_\alpha + \check{X}_\alpha A_\alpha + Q - \check{X}_\alpha B R^{-1} B^T \check{X}_\alpha = 0. \quad (\text{C.86})$$

The resulting optimal cost of the system subject to an initial state \mathbf{x}_0 equals

$$J = \mathbf{x}_0^T \check{X}_\alpha \mathbf{x}_0. \quad (\text{C.87})$$

Proof. Similarly to what was done in (C.72), we can find that the discounted cost is

$$\begin{aligned} J &= \int_0^\infty e^{2\alpha t} \mathbf{x}^T(t) (Q + F^T RF) \mathbf{x}(t) dt \\ &= \mathbf{x}_0^T \left(\int_0^\infty e^{2\alpha t} e^{(A-BF)^T t} (Q + F^T RF) e^{(A-BF)t} dt \right) \mathbf{x}_0 \\ &= \mathbf{x}_0^T \left(\int_0^\infty e^{(A+\alpha I-BF)^T t} (Q + F^T RF) e^{(A+\alpha I-BF)t} dt \right) \mathbf{x}_0. \end{aligned} \quad (\text{C.88})$$

Now we can see that this equation is identical to what we had in (C.72), except that A is replaced by A_α . As a result, the outcome is also the same, except with A replaced by A_α in the derivation of the optimal cost matrix \check{X} and the optimal feedback matrix \check{F} . \square

It is interesting to note how α affects the stability of the controlled system $A_\alpha - BF_\alpha$. The main idea is that we always find a feedback matrix F_α (when possible) such that the controlled system $A_\alpha - BF_\alpha$ is stable. In other words, the real parts of the eigenvalues of $A_\alpha - BF_\alpha$ will be smaller than zero. Or equivalently, the real parts of the eigenvalues of $A - BF_\alpha$ will be smaller than $-\alpha$.

Interestingly this means that, if $\alpha < 0$, the controlled system is not necessarily stable. We just know that the system state $\mathbf{x}(t)$ diverges less quickly than that the weight factor $e^{\alpha t}$ converges to zero, such that $e^{2\alpha t} \mathbf{x}^T(t) Q \mathbf{x}(t)$ still converges to zero. Controllers resulting from negative values of α are therefore often lazy. They prefer to incur the cost of a future bad state, rather than applying an input now to prevent that state.

Similarly, if $\alpha > 0$, the controlled system is most certainly stable, because all the eigenvalues of the controlled system are prescribed to be smaller than $-\alpha$. This is why α is in this case known as the prescribed degree of stability. It results in a more aggressive control law to ensure this stability requirement. Such a control law prefers to apply an input now to prevent a future bad state.

C.3.3. REINTRODUCING PROCESS NOISE

Next we will add input noise again. So we consider the system

$$\dot{\underline{x}}(t) = A\underline{x}(t) + B\underline{u}(t) + \underline{v}(t) = (A - BF)\underline{x}(t) + \underline{v}(t), \quad (\text{C.89})$$

where we assume that we fully know the state $\underline{x}(t)$ when determining the input $\underline{u}(t)$. What is the optimal way to control this system?

Before we start, we need to make a few important realizations. First of all we need to realize that, due to the noise, all parameters became random variables again, including the cost \underline{J} . So we need to work with the expectation $\mathbb{E}[\underline{J}]$ again. Secondly, because of the noise, the cost \underline{J} has become infinite, as we discovered in Section C.2.1. We need to work around that.

There are two ways to do so. We could minimize the expected steady-state cost rate (C.47), or use a negative discount exponent α like in Theorem C.9. We will consider the steady-state cost rate first. When we do, we actually find exactly the same feedback matrix as in Theorem C.11.

Theorem C.14. *Consider the linear system $\dot{\underline{x}}(t) = A\underline{x}(t) + B\underline{u}(t) + \underline{v}(t)$ with feedback law $\underline{u}(t) = -F\underline{x}(t)$. Assume that it is stabilizable: there is a matrix F for which $A - BF$ is stable. The optimal feedback matrix \check{F} minimizing the expected steady-state cost rate (C.47) (with $\alpha = 0$) is the same as the one defined in (C.68). It results in an optimal steady-state cost rate of*

$$\lim_{T \rightarrow \infty} \frac{d\mathbb{E}[\underline{J}_T]}{dT} = \text{tr}(V \check{X}). \quad (\text{C.90})$$

Proof. We need to minimize the expected steady-state cost rate (C.47). We should keep in mind here that we are using a system

$$\dot{\underline{x}}(t) = (A - BF)\underline{x}(t) + \underline{v}(t) = \tilde{A}\underline{x}(t) + \underline{v}(t), \quad (\text{C.91})$$

as well as a cost function

$$\underline{J} = \int_0^\infty \underline{x}^T(t) (Q + F^T RF) \underline{x}(t) dt = \int_0^\infty \underline{x}^T(t) \tilde{Q} \underline{x}(t) dt. \quad (\text{C.92})$$

It follows that the expected steady-state cost rate we need to minimize equals

$$\lim_{T \rightarrow \infty} \frac{d\mathbb{E}[\underline{J}_T]}{dT} = \text{tr}(V \tilde{X}_{\tilde{A}}^{\tilde{Q}}). \quad (\text{C.93})$$

By using Theorem A.25 we can rewrite this to

$$\lim_{T \rightarrow \infty} \frac{d\mathbb{E}[\underline{J}_T]}{dT} = \text{tr} \left(V \int_0^\infty e^{(A-BF)^T t} (Q + F^T RF) e^{(A-BF)t} dt \right). \quad (\text{C.94})$$

Previously, in Theorem C.11, we needed to minimize the cost (C.72) which equaled

$$\begin{aligned} J &= \underline{x}_0^T \left(\int_0^\infty e^{(A-BF)^T t} (Q + F^T RF) e^{(A-BF)t} dt \right) \underline{x}_0 \\ &= \text{tr} \left(\Psi_0 \left(\int_0^\infty e^{(A-BF)^T t} (Q + F^T RF) e^{(A-BF)t} dt \right) \right). \end{aligned} \quad (\text{C.95})$$

So we see that these problems are actually exactly the same, except that Ψ_0 is replaced by V . Since both Ψ_0 and V are just constants, the optimal F is exactly the same. To minimize the steady-state cost rate, we therefore need to choose \check{F} from Theorem C.11, which turns the integral into \check{X} . The result is a steady-state cost rate of (C.90). \square

Do things change if we use a discount exponent? At this point it probably will not surprise you that the answer is ‘No’. We get the optimal feedback matrix \check{F}_α from Theorem C.13.

C

Theorem C.15. Consider the linear system $\dot{\underline{x}}(t) = A\underline{x}(t) + B\underline{u}(t) + \underline{v}(t)$ with feedback law $\underline{u}(t) = -F\underline{x}(t)$. Assume that it is stabilizable up to degree α : there is a matrix F_α for which $A_\alpha - BF_\alpha$ is stable. The optimal feedback matrix \check{F}_α minimizing the discounted cost function (C.84) is the same as the one defined in (C.85). It results in an optimal expected cost of

$$\mathbb{E}[\underline{J}] = \text{tr}\left(\left(\Psi_0 - \frac{V}{2\alpha}\right)\check{X}_\alpha\right). \quad (\text{C.96})$$

Proof. We know from Theorem (C.9) that the discounted cost equals

$$\begin{aligned} \mathbb{E}[\underline{J}] &= \text{tr}\left(\left(\Psi_0 - \frac{V}{2\alpha}\right)\check{X}_{A_\alpha - BF}^{Q+F^T RF}\right) \\ &= \text{tr}\left(\left(\Psi_0 - \frac{V}{2\alpha}\right)\left(\int_0^\infty e^{(A+\alpha I-BF)^T t} (Q + F^T RF)e^{(A+\alpha I-BF)t} dt\right)\right). \end{aligned} \quad (\text{C.97})$$

In Theorem C.13 we needed to optimize (C.88), which equaled

$$J = \text{tr}\left(\Psi_0\left(\int_0^\infty e^{(A+\alpha I-BF)^T t} (Q + F^T RF)e^{(A+\alpha I-BF)t} dt\right)\right). \quad (\text{C.98})$$

Here we can see that these problems are exactly the same, albeit that Ψ_0 has been replaced by $(\Psi_0 - \frac{V}{2\alpha})$. As such, they also have the same solution \check{F}_α from (C.85), which turns the integral into \check{X}_α . The resulting expected cost $\mathbb{E}[\underline{J}]$ equals (C.96). \square

We can conclude that adding process noise does not really affect our optimal control strategy. We had a strategy which was optimal at getting rid of initial disturbances \underline{x}_0 . This same strategy is also optimal at getting rid of new disturbances caused by process noise $\underline{v}(t)$. But does the same hold when we introduce measurement noise?

C.3.4. ESTIMATING THE STATE FROM NOISY MEASUREMENTS

We will now make the problem quite a bit harder by adding a measurement equation to our system. This turns our system into

$$\begin{aligned} \dot{\underline{x}}(t) &= A\underline{x}(t) + B\underline{u}(t) + \underline{v}(t), \\ \underline{y}(t) &= C\underline{x}(t) + \underline{w}(t), \end{aligned} \quad (\text{C.99})$$

where $\underline{w}(t)$ is the *measurement noise*. Similarly to $\underline{v}(t)$, we assume $\underline{w}(t)$ is zero-mean Gaussian white noise with intensity W .

We also assume that we cannot measure the full state $\underline{x}(t)$ anymore. So we cannot use it to determine the input $\mathbf{u}(t)$. At time t we do know the output $\mathbf{y}(t)$, but this is of course corrupted by measurement noise.

To solve this, we will introduce an *observer*. This observer uses a *state estimate* $\hat{\mathbf{x}}(t)$ which approximates $\underline{x}(t)$. We initialize this state estimate at what we expect the state $\underline{x}(t)$ to be. That is, $\hat{\mathbf{x}}_0 = \mathbb{E}[\underline{x}(0)] = \boldsymbol{\mu}_0$. We then update it through the *estimate update law*

$$\dot{\hat{\mathbf{x}}}(t) = A\hat{\mathbf{x}}(t) + B\mathbf{u}(t) + K(\mathbf{y}(t) - C\hat{\mathbf{x}}(t)). \quad (\text{C.100})$$

The idea behind this update law is that $\hat{\mathbf{x}}(t)$ acts just like $\underline{x}(t)$. It is only when the measurement $\mathbf{y}(t) = C\underline{x}(t)$ is unequal to $C\hat{\mathbf{x}}(t)$ that we adjust $\hat{\mathbf{x}}(t)$ to correspond more to the measurements. The amount by which we do this depends on the *observer gain matrix* K .

We now want to choose K such that it minimizes the *state estimation error* $\underline{\mathbf{e}}(t) \equiv \hat{\mathbf{x}}(t) - \underline{x}(t)$. This error is a random variable as well, which makes it hard to minimize. The best we can do is minimize its variance, which is what the next theorem is all about.

Theorem C.16. *Consider the linear system (C.99) with state estimator (C.100). Assume that it is detectable: there is a matrix K for which $A - KC$ is stable. The minimum steady-state error covariance matrix \check{E} equals the solution of the Riccati equation*

$$A\check{E} + \check{E}A^T + V - \check{E}C^TW^{-1}C\check{E} = 0, \quad (\text{C.101})$$

where the corresponding optimal observer gain matrix \check{K} equals

$$\check{K} = \check{E}C^TW^{-1}. \quad (\text{C.102})$$

Proof. To prove this, we will analyze the behavior of $\underline{\mathbf{e}}(t)$. We can find that $\dot{\underline{\mathbf{e}}}(t)$ equals

$$\begin{aligned} \dot{\underline{\mathbf{e}}}(t) &= \dot{\hat{\mathbf{x}}}(t) - \dot{\underline{x}}(t) \\ &= A\hat{\mathbf{x}}(t) + B\mathbf{u}(t) + K(\mathbf{y}(t) - C\hat{\mathbf{x}}(t)) - A\underline{x}(t) - B\mathbf{u}(t) - \underline{\mathbf{v}}(t) \\ &= A(\hat{\mathbf{x}}(t) - \underline{x}(t)) + K(C\underline{x}(t) + \underline{\mathbf{w}}(t) - C\hat{\mathbf{x}}(t)) - \underline{\mathbf{v}}(t) \\ &= (A - KC)\underline{\mathbf{e}}(t) + K\underline{\mathbf{w}}(t) - \underline{\mathbf{v}}(t). \end{aligned} \quad (\text{C.103})$$

We can now use Theorem C.2 to find how the mean $\boldsymbol{\mu}_{\underline{\mathbf{e}}}(t)$ and the covariance matrix $\Sigma_{\underline{\mathbf{e}}}(t)$ of the error $\underline{\mathbf{e}}(t)$ vary over time. We know that, because $\hat{\mathbf{x}}_0 = \mathbb{E}[\underline{x}_0]$, the initial mean of $\underline{\mathbf{e}}$ equals

$$\boldsymbol{\mu}_{\underline{\mathbf{e}}}(0) \equiv \mathbb{E}[\underline{\mathbf{e}}(0)] = \mathbb{E}[\hat{\mathbf{x}}_0 - \underline{x}_0] = \mathbb{E}[\mathbb{E}[\underline{x}_0] - \underline{x}_0] = \mathbf{0}. \quad (\text{C.104})$$

It follows from Theorem C.2 that

$$\boldsymbol{\mu}_{\underline{\mathbf{e}}}(t) \equiv \mathbb{E}[\underline{\mathbf{e}}(t)] = e^{(A - KC)t}\boldsymbol{\mu}_{\underline{\mathbf{e}}}(0) = \mathbf{0}. \quad (\text{C.105})$$

The covariance matrix $\Sigma_{\underline{\mathbf{e}}}(t)$ will not be zero. Instead, the initial covariance matrix $\Sigma_{\underline{\mathbf{e}}}(0)$ equals

$$\begin{aligned} \Sigma_{\underline{\mathbf{e}}}(0) &\equiv \mathbb{E}\left[(\underline{\mathbf{e}}(0) - \mathbb{E}[\underline{\mathbf{e}}(0)])(\underline{\mathbf{e}}(0) - \mathbb{E}[\underline{\mathbf{e}}(0)])^T\right] = \mathbb{E}[\underline{\mathbf{e}}(0)\underline{\mathbf{e}}^T(0)] \\ &= \mathbb{E}\left[(\underline{x}_0 - \mathbb{E}[\underline{x}_0])(\underline{x}_0 - \mathbb{E}[\underline{x}_0])^T\right] = \Sigma_0. \end{aligned} \quad (\text{C.106})$$

The covariance of $\underline{\mathbf{e}}(t)$ now develops according to

$$\Sigma_e(t) = e^{(A-KC)t} (\Sigma_e(0) - E) e^{(A-KC)^T t} + E, \quad (\text{C.107})$$

where E is the solution to the Lyapunov equation

$$(A - KC)E + E(A - KC)^T + (V + KWK^T) = 0. \quad (\text{C.108})$$

This means that, as long as $A - KC$ is stable, the error covariance $\Sigma_e(t)$ will converge to the steady-state error covariance E . And our job is to choose the matrix K so as to minimize E .

We have actually already seen this exact problem before, in the proof of Theorem C.11. Just like we did there, we can complete the squares with respect to W to find that

$$(K - EC^T W^{-1})W(K - EC^T W^{-1})^T + V + AE + EA^T - EC^T W^{-1}CE = 0. \quad (\text{C.109})$$

Through a similar argument, we now find that K can only minimize E when it satisfies $K = EC^T W^{-1}$. It follows that E must satisfy the Riccati equation (C.101), proving the theorem. \square

C.3.5. OPTIMAL CONTROL BASED ON THE STATE ESTIMATE

Now that we have an estimate $\hat{\mathbf{x}}(t)$ of the state, how do we use this to control the system?

The main idea here is to pretend that the estimate $\hat{\mathbf{x}}(t)$ is the true state. In other words, we replace our old control law $\mathbf{u}(t) = -F\mathbf{x}(t)$ by the new law $\mathbf{u}(t) = -F\hat{\mathbf{x}}(t)$. This turns the system equations into

$$\dot{\underline{\mathbf{x}}}(t) = A\underline{\mathbf{x}}(t) - BF\hat{\mathbf{x}}(t) + \underline{\mathbf{v}}(t), \quad (\text{C.110})$$

$$\dot{\hat{\mathbf{x}}}(t) = A\hat{\mathbf{x}}(t) - BF\hat{\mathbf{x}}(t) + K(C\underline{\mathbf{x}}(t) + \underline{\mathbf{w}}(t) - C\hat{\mathbf{x}}(t)). \quad (\text{C.111})$$

We can plug all this in a matrix form according to

$$\begin{bmatrix} \dot{\underline{\mathbf{x}}}(t) \\ \dot{\hat{\mathbf{x}}}(t) \end{bmatrix} = \begin{bmatrix} A & -BF \\ KC & A - BF - KC \end{bmatrix} \begin{bmatrix} \underline{\mathbf{x}}(t) \\ \hat{\mathbf{x}}(t) \end{bmatrix} + \begin{bmatrix} \underline{\mathbf{v}}(t) \\ K\underline{\mathbf{w}}(t) \end{bmatrix}. \quad (\text{C.112})$$

Instead of writing the system using $\underline{\mathbf{x}}(t)$ and $\hat{\mathbf{x}}(t)$ in the state, we can also write it using $\underline{\mathbf{x}}(t)$ and $\underline{\mathbf{e}}(t)$ in the state. This time, the system equations are given by

$$\dot{\underline{\mathbf{x}}}(t) = (A - BF)\underline{\mathbf{x}}(t) - BF\underline{\mathbf{e}}(t) + \underline{\mathbf{v}}(t), \quad (\text{C.113})$$

$$\dot{\underline{\mathbf{e}}}(t) = (A - KC)\underline{\mathbf{e}}(t) + K\underline{\mathbf{w}}(t) - \underline{\mathbf{v}}(t). \quad (\text{C.114})$$

If we put this in a matrix form, we get

$$\begin{bmatrix} \dot{\underline{\mathbf{x}}}(t) \\ \dot{\underline{\mathbf{e}}}(t) \end{bmatrix} = \begin{bmatrix} A - BF & -BF \\ 0 & A - KC \end{bmatrix} \begin{bmatrix} \underline{\mathbf{x}}(t) \\ \underline{\mathbf{e}}(t) \end{bmatrix} + \begin{bmatrix} \underline{\mathbf{v}}(t) \\ K\underline{\mathbf{w}}(t) - \underline{\mathbf{v}}(t) \end{bmatrix}. \quad (\text{C.115})$$

The interesting point is that, in both of these cases, we wind up with a system of the form $\dot{\underline{\mathbf{x}}}(t) = \tilde{A}\underline{\mathbf{x}}(t) + \tilde{\mathbf{v}}(t)$, for some new state vector $\underline{\mathbf{x}}(t)$, some system matrix \tilde{A} and some

zero-mean Gaussian white noise $\underline{\tilde{v}}(t)$ with intensity \tilde{V} . For instance, in the last case we have

$$\underline{\tilde{x}}(t) = \begin{bmatrix} \underline{x}(t) \\ \underline{e}(t) \end{bmatrix}, \quad \tilde{A} = \begin{bmatrix} A - BF & -BF \\ 0 & A - KC \end{bmatrix} \quad \text{and} \quad \tilde{V} = \begin{bmatrix} V & -V \\ -V & KWK^T + V \end{bmatrix}. \quad (\text{C.116})$$

In addition, using $\underline{u}(t) = -F\underline{\hat{x}}(t) = -F(\underline{x}(t) + \underline{e}(t))$ we can write the cost as

$$\begin{aligned} \underline{J} &= \int_0^\infty e^{2\alpha t} \left(\underline{x}^T(t) Q \underline{x}(t) + (\underline{x}(t) + \underline{e}(t))^T F^T R F (\underline{x}(t) + \underline{e}(t)) \right) dt \\ &= \int_0^\infty e^{2\alpha t} \begin{bmatrix} \underline{x}(t) \\ \underline{e}(t) \end{bmatrix}^T \begin{bmatrix} Q + F^T R F & F^T R F \\ F^T R F & F^T R F \end{bmatrix} \begin{bmatrix} \underline{x}(t) \\ \underline{e}(t) \end{bmatrix} dt = \int_0^\infty e^{2\alpha t} \underline{\tilde{x}}^T(t) \tilde{Q} \underline{\tilde{x}}(t) dt. \end{aligned} \quad (\text{C.117})$$

Given the feedback matrix F and the observer gain matrix K , it is always possible to write the system as $\dot{\underline{\tilde{x}}}(t) = \tilde{A}\underline{\tilde{x}}(t) + \underline{\tilde{v}}(t)$, with corresponding state penalty matrix \tilde{Q} . So when analyzing the properties of \underline{J} , we only need to consider systems of this form.

The main question we still have to ask is: which combination of matrices F and K is optimal for our full system? Fascinatingly, it turns out that our previous optimal matrices \check{F} and \check{K} , when applied together, are still optimal in this situation. This means that we can set up our optimal controller and observer separately, and then connect them to wind up with an optimal system. This is known as the *separation principle*.

Theorem C.17. Consider the linear system (C.115). Assume that there are matrices F and K for which $A - BF$ and $A - KC$ are stable. The expected steady-state cost rate (C.47) (with $\alpha = 0$) is minimized by the feedback matrix \check{F} from (C.68) and the observer gain matrix \check{K} from (C.102). The resulting expected steady-state cost rate equals

$$\lim_{T \rightarrow \infty} \frac{d\mathbb{E}[\underline{J}_T]}{dT} = \text{tr}(\check{X} \check{K} W \check{K}^T) + \text{tr}(\check{E} Q) = \text{tr}(\check{X} V) + \text{tr}(\check{E} \check{F}^T R \check{F}). \quad (\text{C.118})$$

Proof. We start by rewriting the system. We already saw two ways of writing the system, being (C.112) and (C.115). We will instead use

$$\begin{bmatrix} \dot{\underline{\tilde{x}}}(t) \\ \dot{\underline{e}}(t) \end{bmatrix} = \begin{bmatrix} A - BF & -KC \\ 0 & A - KC \end{bmatrix} \begin{bmatrix} \underline{\tilde{x}}(t) \\ \underline{e}(t) \end{bmatrix} + \begin{bmatrix} K \underline{w}(t) \\ K \underline{w}(t) + \underline{v}(t) \end{bmatrix}. \quad (\text{C.119})$$

You may have noted that we write the state estimate $\underline{\tilde{x}}(t)$ as a random variable too now. It is true that we know the estimate $\underline{\hat{x}}(t)$ deterministically at time t , when it follows from the estimate update law (C.100). However, before then it will still depend on what exact noise we will get, and so it can take a variety of values. As such, we must treat it as a random variable.

For the above system, the equivalent system vectors/matrices are

$$\begin{aligned} \underline{\tilde{x}}(t) &= \begin{bmatrix} \underline{\hat{x}}(t) \\ \underline{e}(t) \end{bmatrix}, & \tilde{A} &= \begin{bmatrix} A - BF & -KC \\ 0 & A - KC \end{bmatrix}, \\ \tilde{V} &= \begin{bmatrix} KWK^T & KWK^T \\ KWK^T & KWK^T + V \end{bmatrix}, & \tilde{Q} &= \begin{bmatrix} Q + F^T R F & -Q \\ -Q & Q \end{bmatrix}. \end{aligned} \quad (\text{C.120})$$

Subject to these system matrices, we want to optimize the expected steady-state cost rate

$$\lim_{T \rightarrow \infty} \frac{d\mathbb{E}[J_T]}{dT} = \lim_{T \rightarrow \infty} \mathbb{E}[\tilde{\mathbf{x}}^T(T) \tilde{Q} \tilde{\mathbf{x}}(T)] = \text{tr}(\tilde{V} \tilde{X}_{\tilde{A}}^{\tilde{Q}}) = \text{tr}(X_{\tilde{A}}^{\tilde{V}} \tilde{Q}), \quad (\text{C.121})$$

where we have applied (C.47). Note that we have two expressions for the cost rate. Which one is better to use?

Earlier we saw that this depends on whether we want to optimize F or K . Keep in mind here that \tilde{A} and \tilde{Q} depend on F , while \tilde{V} does not. This means that, when optimizing F , it is better to use the first relation, because then we only have to optimize $\tilde{X}_{\tilde{A}}^{\tilde{Q}}$. In fact, this is what we did at Theorem C.11. On the flip side, \tilde{A} and \tilde{V} depend on K , while \tilde{Q} does not. So when optimizing K , it is better to use the second relation. This then comes down to optimizing $X_{\tilde{A}}^{\tilde{V}}$, which we did at Theorem C.16.

But what do we do when we want to optimize K and F simultaneously? In this case, why not try picking K first, and see how that affects our optimization problem for F ? So we want to optimize $X_{\tilde{A}}^{\tilde{V}}$. We know that this matrix satisfies $\tilde{A}X_{\tilde{A}}^{\tilde{V}} + X_{\tilde{A}}^{\tilde{V}}\tilde{A}^T + \tilde{V} = 0$, which we can write as

$$\begin{bmatrix} A - BF & -KC \\ 0 & A - KC \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix} + \begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix} \begin{bmatrix} (A - BF)^T & 0 \\ -C^T K^T & (A - KC)^T \end{bmatrix} + \begin{bmatrix} KWK^T & KWK^T \\ KWK^T & KWK^T + V \end{bmatrix} = 0. \quad (\text{C.122})$$

It is worthwhile to note that $X_{\tilde{A}}^{\tilde{V}}$ is the steady-state covariance of $\tilde{\mathbf{x}}(t)$. In other words, we have

$$X_{11} = \lim_{t \rightarrow \infty} \mathbb{E}[\tilde{\mathbf{x}}(t)\tilde{\mathbf{x}}^T(t)], \quad (\text{C.123})$$

$$X_{22} = \lim_{t \rightarrow \infty} \mathbb{E}[\underline{\mathbf{e}}(t)\underline{\mathbf{e}}^T(t)], \quad (\text{C.124})$$

$$X_{12} = \lim_{t \rightarrow \infty} \mathbb{E}[\tilde{\mathbf{x}}(t)\underline{\mathbf{e}}^T(t)]. \quad (\text{C.125})$$

Expanding the matrix equation now results in four separate equations, being

$$(A - BF)X_{11} - KCX_{12}^T + X_{11}(A - BF)^T - X_{12}C^T K^T + KWK^T = 0, \quad (\text{C.126})$$

$$(A - KC)X_{12}^T + X_{12}^T(A - BF)^T - X_{22}C^T K^T + KWK^T = 0, \quad (\text{C.127})$$

$$(A - BF)X_{12} - KCX_{22} + X_{12}(A - KC)^T + KWK^T = 0, \quad (\text{C.128})$$

$$(A - KC)X_{22} + X_{22}(A - KC)^T + KWK^T + V = 0. \quad (\text{C.129})$$

The expected steady-state cost rate follows as

$$\lim_{T \rightarrow \infty} \frac{d\mathbb{E}[J_T]}{dT} = \text{tr}(X_{\tilde{A}}^{\tilde{V}} \tilde{Q}) = \text{tr}(X_{11}(Q + F^T RF) - X_{12}Q - X_{12}^T Q + X_{22}Q). \quad (\text{C.130})$$

So how shall we pick K ? What we could do is choose the value of K which minimizes the term $\text{tr}(X_{22}Q)$. After all, neither X_{22} nor Q depends on F and (C.129) fully specifies

X_{22} , given K . This does not guarantee that K is optimal, because we are only optimizing a part of the complete cost expression. Though later we will see that it is indeed the optimal observer gain matrix.

The resulting optimization problem is one which we have seen before. It is the exact same problem we faced at Theorem C.16. As a result, the solution must be the same optimal observer gain matrix $\check{K} = \check{E}C^T W^{-1}$, where $\check{E} = X_{22}$ is the resulting steady-state error covariance.

Next, let's look at how this affects the rest of the problem. We start at either (C.127) or (C.128); these two equations are equivalent. For the resulting value of K and X_{22} we have $KCX_{22} = KWK^T$. It follows that $X_{12} = 0$, which in turn reduces (C.126) to the Lyapunov equation

$$(A - BF)X_{11} + X_{11}(A - BF)^T + \check{K}W\check{K}^T = 0. \quad (\text{C.131})$$

In other words, X_{11} equals the Lyapunov solution $X_{A-BF}^{\check{K}W\check{K}^T}$ and we need to choose F so as to optimize $\text{tr}(X_{11}(Q + F^T RF))$. We can also write this as $\text{tr}(\check{K}W\check{K}^T \tilde{X}_{A-BF}^{Q+F^T RF})$ (see Theorem A.29) where we per definition must have

$$(A - BF)^T \tilde{X}_{A-BF}^{Q+F^T RF} + \tilde{X}_{A-BF}^{Q+F^T RF}(A - BF) + Q + F^T RF = 0. \quad (\text{C.132})$$

We need to optimize this with respect to F . Luckily, we have already seen this exact problem before at (C.73). The resulting solution equaled \check{X} from Theorem C.11. Using this result, we can write the expected steady-state cost rate as the first expression in (C.118).

But wait a second. What we have done so far is pick a value $K = \check{K}$ and then optimize F to \check{F} . Who says that this is the joint optimum for K and F together? To prove that it is, we can also do this exact derivation in a different set-up. Instead of putting $\hat{x}(t)$ and $e(t)$ into $\tilde{x}(t)$, we now use $\underline{x}(t)$ and $\underline{e}(t)$. That is, we use (C.116) and (C.117). We then follow the exact same steps, with main exception that we do not optimize $\text{tr}(X_{\tilde{A}}^{\tilde{V}} \tilde{Q})$ but instead use $\text{tr}(\tilde{V} \tilde{X}_{\tilde{A}}^{\tilde{Q}})$.

Because of this change, we now first have to pick $F = \check{F}$. Optimizing for K would then result in $K = \check{K}$. So using $F = \check{F}$ implies $K = \check{K}$ is optimal, while using $K = \check{K}$ implies using $F = \check{F}$ is optimal. This important result means that the combination (\check{F}, \check{K}) is at least a local optimum of the problem. But since the problem is quadratic in F and K , it must also be the global optimum.

An interesting side-effect of the above derivation is that we do wind up with a different expression for the expected steady-state cost rate. It now equals the second expression from (C.118). \square

An interesting result from the previous theorem is that (C.125) equals zero. In other words, the steady state error $e(t)$ and estimate $\hat{x}(t)$ are uncorrelated and hence – since they are Gaussian – independent. As a result, we can expand (C.121) into

$$\lim_{T \rightarrow \infty} \frac{d\mathbb{E}[J_T]}{dT} = \lim_{T \rightarrow \infty} \mathbb{E}[\hat{x}^T(T)(Q + F^T RF)\hat{x}(T)] + \lim_{T \rightarrow \infty} \mathbb{E}[e^T(T)Qe(T)]. \quad (\text{C.133})$$

The cost rate required just to the control the state estimate $\hat{x}(t)$ now equals $\text{tr}(\check{X}V)$, while the cost rate only due to the estimation error $e(t)$ is $\text{tr}(\check{E}Q)$. But the sum of these two terms is less than the total expected cost rate (C.118).

The reason why we eventually wind up with more costs is because there is still coupling between the two parameters $\underline{e}(t)$ and $\hat{\underline{x}}(t)$. To be precise, the error affects the state estimate due to the term $-KC$ within \tilde{A} . Intuitively this means that, when there is an error $\underline{e}(t)$, this error results in inputs $\underline{u}(t)$ to control the estimate $\hat{\underline{x}}(t)$, even though that estimate turns out to be wrong. Luckily $\hat{\underline{x}}(t)$ does not influence $\underline{e}(t)$, which is the reason why we can still design our controller and observer separately. But the cost we wind up with is higher than the cost we would expect to get from each individual part.

The final question we should ask ourselves is whether this result also holds for the discounted cost function. The answer is that it does. There is one difference though. The costs only due to the error, given by

$$\int_0^\infty e^{2\alpha t} \underline{e}^T(t) Q \underline{e}(t) dt, \quad (\text{C.134})$$

are now not minimized anymore by $\check{K} = \check{E} C^T W^{-1}$. Instead, we need a new observer gain matrix \check{K}_α .

Theorem C.18. Consider the linear system (C.115). Assume there are matrices F_α and K_α for which $A_\alpha - BF_\alpha$ and $A_\alpha - K_\alpha C$ are stable. The expected discounted cost (C.84) with $\alpha < 0$ is minimized by the feedback matrix \check{F}_α from (C.85) and the observer gain matrix \check{K}_α given by

$$\check{K}_\alpha = \check{E}_\alpha C^T W^{-1}, \quad (\text{C.135})$$

where \check{E}_α is the solution to the Riccati equation

$$A_\alpha \check{E}_\alpha + \check{E}_\alpha A_\alpha^T + (V - 2\alpha \Sigma_0) - \check{E}_\alpha C^T W^{-1} C \check{E}_\alpha = 0. \quad (\text{C.136})$$

The resulting expected cost equals

$$\begin{aligned} \mathbb{E}[\underline{J}] &= \text{tr}\left(\left(\frac{-\check{X}_\alpha}{2\alpha}\right)(\check{K}W\check{K}^T - 2\alpha \boldsymbol{\mu}_0 \boldsymbol{\mu}_0^T)\right) + \text{tr}\left(\left(\frac{-\check{E}_\alpha}{2\alpha}\right)Q\right) \\ &= \text{tr}\left(\left(\frac{-\check{X}_\alpha}{2\alpha}\right)(V - 2\alpha \Psi_0)\right) + \text{tr}\left(\left(\frac{-\check{E}_\alpha}{2\alpha}\right)\check{F}^T R \check{F}\right). \end{aligned} \quad (\text{C.137})$$

Proof. We can derive this result in almost the exact same way as we derived Theorem C.17. We just need to pay attention to the minor differences.

First of all, the initial distribution of $\hat{\underline{x}}(t)$ suddenly has become important. It equals

$$\tilde{\underline{x}}_0 = \begin{bmatrix} \hat{\underline{x}}_0 \\ \underline{e}_0 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_0 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & \Sigma_0 \end{bmatrix}\right) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_0, \tilde{\Sigma}_0). \quad (\text{C.138})$$

According to Theorem C.9, the cost which we want to minimize now has turned from $\text{tr}(\tilde{V} \tilde{X}_{\tilde{A}}^{\tilde{Q}})$ into

$$\mathbb{E}[\underline{J}] = \text{tr}\left(\left(\tilde{\Psi}_0 - \frac{\tilde{V}}{2\alpha}\right)\tilde{X}_{\tilde{A}_\alpha}^{\tilde{Q}}\right). \quad (\text{C.139})$$

So we need to replace \tilde{A} by \tilde{A}_α , or equivalently A by A_α . Similarly, we need to replace \tilde{V} by $\left(\tilde{\Psi}_0 - \frac{\tilde{V}}{2\alpha}\right)$, or equivalently

$$\begin{bmatrix} KWK^T & KWK^T \\ KWK^T & KWK^T + V \end{bmatrix} \text{ by } \begin{bmatrix} \boldsymbol{\mu}_0 \boldsymbol{\mu}_0^T & 0 \\ 0 & \Sigma_0 \end{bmatrix} - \frac{1}{2\alpha} \begin{bmatrix} KWK^T & KWK^T \\ KWK^T & KWK^T + V \end{bmatrix} \quad (\text{C.140})$$

$$= \begin{bmatrix} \boldsymbol{\mu}_0 \boldsymbol{\mu}_0^T + K \left(\frac{-W}{2\alpha} \right) K^T & K \left(\frac{-W}{2\alpha} \right) K^T \\ K \left(\frac{-W}{2\alpha} \right) K^T & K \left(\frac{-W}{2\alpha} \right) K^T + \left(\Sigma_0 - \frac{V}{2\alpha} \right) \end{bmatrix}.$$

That is, we need to replace W by $\frac{-W}{2\alpha}$ and V by $\left(\Sigma_0 - \frac{V}{2\alpha}\right)$.

Picking K now can be done in the same way. We optimize the equivalent of (C.129), which equals

$$(A_\alpha - KC) X_{22} + X_{22} (A_\alpha - KC)^T + K \left(\frac{-W}{2\alpha} \right) K^T + \left(\Sigma_0 - \frac{V}{2\alpha} \right) = 0. \quad (\text{C.141})$$

Let's write the optimal value of X_{22} as \check{E}'_α . It will be the solution of the Riccati equation

$$A_\alpha \check{E}'_\alpha + \check{E}'_\alpha A_\alpha^T + \left(\Sigma_0 - \frac{V}{2\alpha} \right) - \check{E}'_\alpha C^T \left(\frac{-W}{2\alpha} \right)^{-1} C \check{E}'_\alpha = 0. \quad (\text{C.142})$$

So solving the Riccati equation gives us \check{E}'_α . The corresponding optimal observer gain matrix \check{K}_α then equals

$$\check{K}_\alpha = \check{E}'_\alpha C^T \left(\frac{-W}{2\alpha} \right)^{-1}. \quad (\text{C.143})$$

The only downside is that these equations do not work as $\alpha \rightarrow 0$, because then $\check{E}'_\alpha \rightarrow \infty$. We fix this by multiplying (C.142) by -2α and subsequently defining $\check{E}_\alpha = -2\alpha \check{E}'_\alpha$. This results in the more familiar Riccati equation (C.136), while the optimal observer gain matrix can be written as (C.135).

The result of this choice of K is that $X_{12} = 0$ again and that, equivalently to (C.131), X_{11} becomes the solution of the Lyapunov equation

$$(A_\alpha - BF) X_{11} + X_{11} (A_\alpha - BF)^T + \boldsymbol{\mu}_0 \boldsymbol{\mu}_0^T + K \left(\frac{-W}{2\alpha} \right) K^T = 0. \quad (\text{C.144})$$

We now want to choose F to minimize the expected cost

$$\begin{aligned} \mathbb{E} [\underline{J}] &= \text{tr} (X_{11} (Q + F^T RF) + \check{E}'_\alpha Q) \\ &= \text{tr} \left(\left(\boldsymbol{\mu}_0 \boldsymbol{\mu}_0^T + K \left(\frac{-W}{2\alpha} \right) K^T \right) \bar{X}_{A_\alpha - BF}^{Q+F^T RF} + \check{E}'_\alpha Q \right), \end{aligned} \quad (\text{C.145})$$

where we have again applied Theorem A.29. Note that only the Lyapunov solution term depends on F , and we already know how to find its optimum. To be precise, its optimum is the same as previously, except that A has been replaced by A_α . It is hence the solution \check{X}_α to the Riccati equation (C.86). The corresponding value of F equals $\check{F}_\alpha = R^{-1} B^T \check{X}_\alpha$. Rewriting the expected cost (C.145) will then result in the first expression from (C.137). To find the second expression from (C.137) we should have used a different joint vector $\underline{x}(t)$, identically to what was mentioned at the end of Theorem C.17. \square

We know how α affects the feedback matrix \check{F}_α . If α becomes negative, we get a lazy controller. So how does α affect the observer gain matrix \check{K}_α ?

To see how this works, we just need to compare the two discounted Riccati equations (C.86) and (C.136). They are nearly the same. C replaces B , W replaces R , $(V - 2\alpha\Sigma_0)$ replaces Q , \check{X}_α replaces \check{E}_α and \check{F}_α replaces \check{K}_α . If a negative value of α will cause \check{F}_α to be small, then the same will hold for \check{K}_α .

C Intuitively you can see this as follows. When α is highly negative, the current time is far more important than the future; even the near future. So the controller will think, ‘Aggressively adjusting the state to any process noise will cost effort now, and only slowly improve the state over time. That doesn’t seem worth it, especially since the state will soon be unimportant anyway.’ Similarly, the observer will think, ‘Aggressively adjusting the state estimate to any measurement noise will cost effort now, and only slowly improve this estimate over time. That doesn’t seem worth it, especially since any errors will soon be unimportant anyway.’ The two thoughts are equivalent too.

The result of such a lazy observer is that the estimate $\hat{x}(t)$ will not fluctuate too much along with measurement noise. As such, the total input applied to the system will be less. However, it also means that any deviations of the observation error $\underline{e}(t)$ that may be present due to noise will be damped out less quickly, resulting in a larger steady-state error covariance $\lim_{t \rightarrow \infty} \Sigma_e(t)$.

An interesting fact is that, when we use $\alpha > 0$, and we get a more aggressive observer, then the steady-state error covariance $\lim_{t \rightarrow \infty} \Sigma_e(t)$ will also be larger. Feel free to reason out for yourself why this is the case.

C.4. THE VARIANCE OF THE LQG COST

In Section C.2 we have derived expressions for the mean LQG cost $\mathbb{E}[J]$. In this section we expand on that by deriving expressions for the variance $\mathbb{V}[J]$. We should note here that the cost J does *not* have a Gaussian distribution, so the mean and variance do not tell us everything about the distribution of J . (Look up Figure C.2 on page 330 to see what the distribution does look like.) Nevertheless, the variance does give some information about how spread out it will be.

We start by studying two special cases: the infinite-time case $T \rightarrow \infty$ (Section C.4.1) and the non-discounted case $\alpha = 0$. (Section C.4.2). This will let us know the general method with which we can derive expressions for the variance. We then expand this method to the general case with finite T and negative α (Section C.4.3). Finally, we derive our expressions all over again, but then using matrix exponentials instead of Lyapunov solutions (Section C.4.4).

Most of the proofs you find in this section can also be found in Bijl et al. (2016), albeit in an abbreviated form and with a slightly adjusted notation. So for the summary, you can look up that paper. For the more elaborate and comprehensive version, keep on reading here.

C.4.1. THE INFINITE-TIME CASE

We first investigate the infinite-time discounted cost function C.48. The variance of the cost is given by

$$\mathbb{V}[\underline{J}] \equiv \mathbb{E}[(\underline{J} - \mathbb{E}[\underline{J}])^2] = \mathbb{E}[\underline{J}^2] - \mathbb{E}[\underline{J}]^2. \quad (\text{C.146})$$

We already know how to calculate $\mathbb{E}[\underline{J}]$, but how do we find $\mathbb{E}[\underline{J}^2]$, given the system parameters A , V , Q , R , μ_0 and Σ_0 ? (Or alternatively using $\Psi_0 = \Sigma_0 + \mu_0 \mu_0^T$ instead of Σ_0 ?) That is the key question. It will be answered by the following theorem, which uses the notation from Appendix A.4.1.

Theorem C.19. Consider the system $\dot{\underline{x}}(t) = A\underline{x}(t) + \underline{v}(t)$. Assume that $\alpha < 0$ and that A_α is stable. The variance $\mathbb{V}[\underline{J}]$ of the infinite-time cost C.48 is then given by

$$\mathbb{V}[\underline{J}] = 2\text{tr}\left(\left(\Psi_0 \bar{X}_\alpha^Q\right)^2\right) - 2\left(\mu_0^T \bar{X}_\alpha^Q \mu_0\right)^2 + 4\text{tr}\left(\left(X_{2\alpha}^{\Psi_0} - \frac{X_{2\alpha}^V}{4\alpha}\right) \bar{X}_\alpha^Q V \bar{X}_\alpha^Q\right). \quad (\text{C.147})$$

Proof. We start off by evaluating $\mathbb{E}[\underline{J}^2]$. This equals

$$\begin{aligned} \mathbb{E}[\underline{J}^2] &= \mathbb{E}\left[\left(\int_0^\infty e^{2\alpha t} \underline{x}^T(t) Q \underline{x}(t) dt\right)^2\right] \\ &= \mathbb{E}\left[\left(\int_0^\infty e^{2\alpha t_1} \underline{x}^T(t_1) Q \underline{x}(t_1) dt\right)\left(\int_0^\infty e^{2\alpha t_2} \underline{x}^T(t_2) Q \underline{x}(t_2) dt\right)\right] \\ &= \mathbb{E}\left[\int_0^\infty \int_0^\infty e^{2\alpha(t_1+t_2)} \underline{x}^T(t_1) Q \underline{x}(t_1) \underline{x}^T(t_2) Q \underline{x}(t_2) dt_2 dt_1\right]. \end{aligned} \quad (\text{C.148})$$

The above quantity is a scalar, which means we can take the trace of it. This allows us to apply Theorem B.20, resulting in

$$\begin{aligned} \mathbb{E}[\underline{J}^2] &= \mathbb{E}\left[\int_0^\infty \int_0^\infty \text{tr}(e^{2\alpha(t_1+t_2)} \underline{x}^T(t_1) Q \underline{x}(t_1) \underline{x}^T(t_2) Q \underline{x}(t_2)) dt_2 dt_1\right] \\ &= \int_0^\infty \int_0^\infty (\text{tr}(e^{2\alpha t_1} \Psi(t_1) Q) \text{tr}(e^{2\alpha t_2} \Psi(t_2) Q) + 2\text{tr}(e^{2\alpha(t_1+t_2)} \Psi(t_2, t_1) Q \Psi(t_1, t_2) Q) \\ &\quad - 2e^{2\alpha(t_1+t_2)} \mu^T(t_1) Q \mu(t_1) \mu^T(t_2) Q \mu(t_2)) dt_2 dt_1. \end{aligned} \quad (\text{C.149})$$

We know $\mu(t)$ from Theorem C.2 and both $\Psi(t)$ and $\Psi(t_1, t_2)$ from Theorem C.5. That allows us to expand the above expression. This will give us overly long equations though, so to prevent that, we will consider each of the three terms from the above equation separately. That is, we write $\mathbb{E}[\underline{J}^2] = T_1 + T_2 + T_3$, where

$$T_1 = \int_0^\infty \int_0^\infty \text{tr}(e^{2\alpha t_1} \Psi(t_1) Q) \text{tr}(e^{2\alpha t_2} \Psi(t_2) Q) dt_2 dt_1, \quad (\text{C.150})$$

$$T_2 = 2 \int_0^\infty \int_0^\infty \text{tr}(e^{2\alpha(t_1+t_2)} \Psi(t_2, t_1) Q \Psi(t_1, t_2) Q) dt_2 dt_1, \quad (\text{C.151})$$

$$T_3 = -2 \int_0^\infty \int_0^\infty e^{2\alpha(t_1+t_2)} \mu^T(t_1) Q \mu(t_1) \mu^T(t_2) Q \mu(t_2) dt_2 dt_1. \quad (\text{C.152})$$

We can see right away (see for instance (C.37)) that $T_1 = \mathbb{E}[\underline{J}]^2$. It now follows directly from (C.146) that $\mathbb{V}[\underline{J}] = T_2 + T_3$. So we only need to solve T_2 and T_3 .

We start with T_3 . We can apply Theorem C.2 to expand $\boldsymbol{\mu}(t)$ and then use Theorem A.25 to solve the resulting integral. This will give us

$$\begin{aligned} T_3 &= -2 \left(\int_0^\infty e^{2\alpha t} \boldsymbol{\mu}^T(t) Q \boldsymbol{\mu}(t) dt \right)^2 \\ &= -2 \left(\int_0^\infty e^{2\alpha t} \boldsymbol{\mu}_0^T e^{A^T t} Q e^{At} \boldsymbol{\mu}_0 dt \right)^2 \\ &= -2 \left(\boldsymbol{\mu}_0^T \left(\int_0^\infty e^{A_\alpha^T t} Q e^{A_\alpha t} dt \right) \boldsymbol{\mu}_0 \right)^2 \\ &= -2 \left(\boldsymbol{\mu}_0^T \bar{X}_\alpha^Q \boldsymbol{\mu}_0 \right)^2. \end{aligned} \quad (\text{C.153})$$

C

Next up is T_2 . This is the most tricky term to solve. The first reason why this is tricky is because the expression (C.34) for $\Psi(t_1, t_2)$ is only valid for $t_1 \leq t_2$. Luckily, we can notice that the integrand of (C.151) is symmetric in t_1 and t_2 . That is, if we interchange t_1 and t_2 , we get exactly the same integrand; except transposed, but the integrand is a scalar anyway. Because of this, we do not have to integrate over the complete integration area – all possible values of t_1 and t_2 . We could also integrate over half the integration area – only over the values $t_1 \leq t_2$. By doing this, we get half the outcome, so we should multiply the outcome by two. This gives us

$$T_2 = 4 \int_0^\infty \int_{t_1}^\infty \text{tr} \left(e^{2\alpha(t_1+t_2)} \Psi^T(t_1, t_2) Q \Psi(t_1, t_2) Q \right) dt_2 dt_1. \quad (\text{C.154})$$

This allows us to apply (C.34). If we then also define the shorthand notation $\Delta = \Psi_0 - X^V$, it follows that

$$\begin{aligned} T_2 &= 4 \int_0^\infty \int_{t_1}^\infty \text{tr} \left(e^{2\alpha(t_1+t_2)} \left(e^{At_1} (\Psi_0 - X^V) e^{A^T t_2} + X^V e^{A^T(t_2-t_1)} \right)^T Q \right. \\ &\quad \left. \left(e^{At_1} (\Psi_0 - X^V) e^{A^T t_2} + X^V e^{A^T(t_2-t_1)} \right) Q \right) dt_2 dt_1 \\ &= 4 \int_0^\infty \int_{t_1}^\infty \text{tr} \left(e^{2\alpha(t_1+t_2)} e^{At_2} \Delta e^{A^T t_1} Q e^{At_1} \Delta e^{A^T t_2} Q + e^{2\alpha(t_1+t_2)} e^{A(t_2-t_1)} X^V Q X^V e^{A^T(t_2-t_1)} Q \right. \\ &\quad \left. + 2e^{2\alpha(t_1+t_2)} e^{A(t_2-t_1)} X^V Q e^{At_1} \Delta e^{A^T t_2} Q \right) dt_2 dt_1. \end{aligned} \quad (\text{C.155})$$

There are once more three terms in this expression, which we will denote by $T_{2,1}$, $T_{2,2}$ and $T_{2,3}$, respectively. To be precise,

$$T_{2,1} = 4 \int_0^\infty \int_{t_1}^\infty \text{tr} \left(e^{2\alpha(t_1+t_2)} e^{At_2} \Delta e^{A^T t_1} Q e^{At_1} \Delta e^{A^T t_2} Q \right) dt_2 dt_1, \quad (\text{C.156})$$

$$T_{2,2} = 4 \int_0^\infty \int_{t_1}^\infty \text{tr} \left(e^{2\alpha(t_1+t_2)} e^{A(t_2-t_1)} X^V Q X^V e^{A^T(t_2-t_1)} Q \right) dt_2 dt_1, \quad (\text{C.157})$$

$$T_{2,3} = 8 \int_0^\infty \int_{t_1}^\infty \text{tr} \left(e^{2\alpha(t_1+t_2)} e^{A(t_2-t_1)} X^V Q e^{At_1} \Delta e^{A^T t_2} Q \right) dt_2 dt_1. \quad (\text{C.158})$$

We start with $T_{2,1}$. We can see that the integrand here is once more symmetric in t_1 and t_2 . That means we can apply the inverse trick of what we just did. This allows us to solve the integrals according to

$$\begin{aligned} T_{2,1} &= 2\text{tr}\left(\int_0^\infty \int_0^\infty e^{2\alpha(t_1+t_2)} \Delta e^{A^T t_1} Q e^{At_1} \Delta e^{A^T t_2} Q e^{At_2} dt_2 dt_1\right) \\ &= 2\text{tr}\left(\left(\int_0^\infty e^{2\alpha t} \Delta e^{A^T t} Q e^{At} dt\right)^2\right) \\ &= 2\text{tr}\left(\left(\Delta \bar{X}_\alpha^Q\right)^2\right). \end{aligned} \quad (\text{C.159})$$

The next term, $T_{2,2}$ is not symmetric in t_1 and t_2 . We do want both lower integration bounds to be zero though. To fix this, we define $s = t_2 - t_1$. If we would integrate over s instead of t_2 , then $ds = dt_2$ while the integral bounds would run from 0 to ∞ . This results in

$$\begin{aligned} T_{2,2} &= 4 \int_0^\infty \int_0^\infty \text{tr}\left(e^{2\alpha(2t_1+s)} X^V Q X^V e^{A^T s} Q e^{As}\right) ds dt_1 \\ &= 4 \int_0^\infty \text{tr}\left(e^{4\alpha t_1} X^V Q X^V \bar{X}_\alpha^Q\right) dt_1 \\ &= 4\text{tr}\left(X^V Q \left(\frac{-X^V}{4\alpha}\right) \bar{X}_\alpha^Q\right). \end{aligned} \quad (\text{C.160})$$

For $T_{2,3}$ we can apply the same substitution of $s = t_2 - t_1$. This turns it into

$$\begin{aligned} T_{2,3} &= 8 \int_0^\infty \int_0^\infty \text{tr}\left(e^{2\alpha(2t_1+s)} X^V Q e^{At_1} \Delta e^{A^T(s+t_1)} Q e^{As}\right) ds dt_1 \\ &= 8 \int_0^\infty \text{tr}\left(e^{4\alpha t_1} X^V Q e^{At_1} \Delta e^{A^T t_1} \bar{X}_\alpha^Q\right) dt_1 \\ &= 8\text{tr}\left(X^V Q X_{2\alpha}^\Delta \bar{X}_\alpha^Q\right). \end{aligned} \quad (\text{C.161})$$

Putting everything together results in the cost variance

$$\mathbb{V}[J] = 2\text{tr}\left(\left(\Delta \bar{X}_\alpha^Q\right)^2\right) - 2\left(\boldsymbol{\mu}_0^T \bar{X}_\alpha^Q \boldsymbol{\mu}_0\right)^2 + 4\text{tr}\left(X^V Q \left(2X_{2\alpha}^\Delta - \frac{X^V}{4\alpha}\right) \bar{X}_\alpha^Q\right). \quad (\text{C.162})$$

This seems like a pretty nice and easy-to-use expression, but it is not equal to (C.147), which is still a slight bit more elegant. Luckily we can rewrite one into the other. This is a more elaborate process than you might at first expect, so I have made a separate theorem out of that. Theorem C.20 completes this proof. \square

Theorem C.20. *The two expressions (C.147) and (C.162) are equivalent.*

Proof. We will start by rewriting $2\text{tr}\left(\Delta \bar{X}_\alpha^Q\right)^2$. It equals

$$\begin{aligned} 2\text{tr}\left(\left(\Delta \bar{X}_\alpha^Q\right)^2\right) &= 2\text{tr}\left(\left((\Psi_0 - X^V) \bar{X}_\alpha^Q\right)^2\right) \\ &= 2\text{tr}\left(\left(\Psi_0 \bar{X}_\alpha^Q\right)^2 + \left(X^V \bar{X}_\alpha^Q\right)^2 - 2\Psi_0 \bar{X}_\alpha^Q X^V \bar{X}_\alpha^Q\right). \end{aligned} \quad (\text{C.163})$$

Next, we briefly define the shorthand $P = 2X_{2\alpha}^\Delta - \frac{X^V}{4\alpha}$ and rewrite $4\text{tr}(X^V Q P \bar{X}_\alpha^Q)$. Due to the definition of \bar{X}_α^Q , it must equal

$$4\text{tr}(X^V Q P \bar{X}_\alpha^Q) = 4\text{tr}\left(X^V \left(-A_\alpha^T \bar{X}_\alpha^Q - \bar{X}_\alpha^Q A_\alpha\right) P \bar{X}_\alpha^Q\right). \quad (\text{C.164})$$

We know that $\text{tr}(S) = \text{tr}(S^T)$ for any matrix S , and as a result $\text{tr}(S) = \frac{1}{2}\text{tr}(S + S^T)$. If we apply this, we find that

$$\begin{aligned} 4\text{tr}(X^V Q P \bar{X}_\alpha^Q) &= 2\text{tr}\left(-X^V A_\alpha^T \bar{X}_\alpha^Q P \bar{X}_\alpha^Q - X^V \bar{X}_\alpha^Q A_\alpha P \bar{X}_\alpha^Q\right. \\ &\quad \left.- A_\alpha X^V \bar{X}_\alpha^Q P \bar{X}_\alpha^Q - X^V \bar{X}_\alpha^Q P A_\alpha^T \bar{X}_\alpha^Q\right). \end{aligned} \quad (\text{C.165})$$

Per definition we have $A_\alpha = A + \alpha I$. Using this, we can see that the above is equivalent to

$$\begin{aligned} 4\text{tr}(X^V Q P \bar{X}_\alpha^Q) &= 2\text{tr}\left(-X^V A^T \bar{X}_\alpha^Q P \bar{X}_\alpha^Q - X^V \bar{X}_\alpha^Q A_{2\alpha} P \bar{X}_\alpha^Q\right. \\ &\quad \left.- A X^V \bar{X}_\alpha^Q P \bar{X}_\alpha^Q - X^V \bar{X}_\alpha^Q P A_{2\alpha}^T \bar{X}_\alpha^Q\right) \\ &= 2\text{tr}\left(\left(-AX^V - X^V A^T\right) \bar{X}_\alpha^Q P \bar{X}_\alpha^Q + X^V \bar{X}_\alpha^Q \left(-A_{2\alpha} P - PA_{2\alpha}^T\right) \bar{X}_\alpha^Q\right). \end{aligned} \quad (\text{C.166})$$

We know that $-AX^V - X^V A^T$ equals V , but what is $-A_{2\alpha} P - PA_{2\alpha}^T$ equal to? Applying the definition $\Delta = \Psi_0 - X^V$, we find that

$$\begin{aligned} -A_{2\alpha} P - PA_{2\alpha}^T &= -2(A_{2\alpha} X_{2\alpha}^\Delta - X_{2\alpha}^\Delta A_{2\alpha}^T) + (A + 2\alpha I) \frac{X^V}{4\alpha} + \frac{X^V}{4\alpha} (A + 2\alpha I)^T \\ &= 2\Delta + \frac{1}{4\alpha} (AX^V + X^V A^T) + 4\alpha \frac{X^V}{4\alpha} \\ &= 2\Psi_0 - 2X^V + \frac{-V}{4\alpha} + X^V = 2\Psi_0 - X^V - \frac{V}{4\alpha}. \end{aligned} \quad (\text{C.167})$$

We can also rewrite P itself. If we first apply Theorem A.27, followed by Theorem A.30, it turns into

$$P = 2X_{2\alpha}^{\Psi_0 - X^V} - \frac{X^V}{4\alpha} = 2\left(X_{2\alpha}^{\Psi_0} - X_{2\alpha}^{X^V}\right) - \frac{X^V}{4\alpha} = 2\left(X_{2\alpha}^{\Psi_0} - \frac{X_{2\alpha}^V - X^V}{4\alpha}\right) - \frac{X^V}{4\alpha}. \quad (\text{C.168})$$

As a result of this, we wind up with

$$\begin{aligned} 4\text{tr}(X^V Q P \bar{X}_\alpha^Q) &= 2\text{tr}\left(V \bar{X}_\alpha^Q \left(2\left(X_{2\alpha}^{\Psi_0} - \frac{X_{2\alpha}^V - X^V}{4\alpha}\right) - \frac{X^V}{4\alpha}\right) \bar{X}_\alpha^Q + X^V \bar{X}_\alpha^Q \left(2\Psi_0 - X^V - \frac{V}{4\alpha}\right) \bar{X}_\alpha^Q\right) \\ &= 4\text{tr}\left(V \bar{X}_\alpha^Q \left(X_{2\alpha}^{\Psi_0} - \frac{X_{2\alpha}^V}{4\alpha}\right) \bar{X}_\alpha^Q\right) + 2\text{tr}\left(2X^V \bar{X}_\alpha^Q \Psi_0 \bar{X}_\alpha^Q - \left(X^V \bar{X}_\alpha^Q\right)^2\right). \end{aligned} \quad (\text{C.169})$$

By combining (C.163) and (C.169), we now see that

$$2\text{tr}\left(\left(\Delta \bar{X}_\alpha^Q\right)^2\right) + 4\text{tr}(X^V Q P \bar{X}_\alpha^Q) = 2\text{tr}\left(\left(\Psi_0 \bar{X}_\alpha^Q\right)^2\right) + 4\text{tr}\left(\left(X_{2\alpha}^{\Psi_0} - \frac{X_{2\alpha}^V}{4\alpha}\right) \bar{X}_\alpha^Q V \bar{X}_\alpha^Q\right), \quad (\text{C.170})$$

which proves that (C.147) and (C.162) are indeed equivalent. \square

C.4.2. THE NON-DISCOUNTED CASE

The cost for the non-discounted case can be derived in a very similar way. The details are all just different. If we once more use $\Delta = \Psi_0 - X^V$, we wind up with the following theorem.

Theorem C.21. Consider the system $\dot{\underline{x}}(t) = A\underline{x}(t) + \underline{v}(t)$. Assume that A is Sylvester. The variance $\mathbb{V}[\underline{J}_T]$ of the finite-time cost (C.42) (with $\alpha = 0$) is then given by

$$\begin{aligned}\mathbb{V}[\underline{J}_T] &= 2\text{tr}((\Delta \bar{X}^Q(T))^2) - 2(\boldsymbol{\mu}_0^T \bar{X}^Q(T) \boldsymbol{\mu}_0)^2 + 4\text{tr}\left(X^V Q \left(X^V \left(T \bar{X}^Q - \bar{X}^{\bar{X}^Q}(T)\right) + 2X^\Delta \bar{X}^Q(T) - 2\bar{X}^{X^\Delta e^{A^T} Q}(T)\right)\right). \quad (\text{C.171})\end{aligned}$$

Proof. We will follow the same set-up as the proof of Theorem C.19. Make sure you have read that proof before. Just like previously, we have $\mathbb{V}[\underline{J}_T] = T_2 + T_3$, where the terms T_2 and T_3 follow from (C.151) and (C.152). The main difference is that now the integrals run up to time T and α has been set to zero.

For T_3 this results in

$$T_3 = -2 \left(\boldsymbol{\mu}_0^T \left(\int_0^T e^{A^T t} Q e^{A t} dt \right) \boldsymbol{\mu}_0 \right) = -2(\boldsymbol{\mu}_0^T \bar{X}^Q(T) \boldsymbol{\mu}_0), \quad (\text{C.172})$$

where we have used definition (A.119) of $\bar{X}^Q(T)$. Note that $\bar{X}^Q(T)$ can be found through Theorem A.26.

We once more split T_2 up into the three terms $T_{2,1}$, $T_{2,2}$ and $T_{2,3}$. $T_{2,1}$ now equals

$$T_{2,1} = 2\text{tr}\left(\left(\int_0^T \Delta e^{A^T t} Q e^{A t} dt\right)^2\right) = 2\text{tr}\left((\Delta \bar{X}^Q(T))^2\right). \quad (\text{C.173})$$

For $T_{2,2}$ we apply the same substitution of $s = t_2 - t_1$. This results in

$$\begin{aligned}T_{2,2} &= 4 \int_0^T \int_{t_1}^T \text{tr}\left(e^{A(t_2-t_1)} X^V Q X^V e^{A^T(t_2-t_1)} Q\right) dt_2 dt_1 \\ &= 4 \int_0^T \int_0^{T-t_1} \text{tr}\left(X^V Q X^V e^{A^T s} Q e^{As}\right) ds dt_1.\end{aligned} \quad (\text{C.174})$$

Solving this is relatively easy if we first interchange the integrals. When doing so, we should keep the integration area the same. This integration area is defined by $0 \leq t_1 \leq T$ and $0 \leq s \leq T - t_1$, or alternatively as $0 \leq s \leq T$ and $0 \leq s \leq T - s$. (If you draw this area out in a plane, you will find that these integration bounds are equivalent.) So we get

$$\begin{aligned}T_{2,2} &= 4 \int_0^T \int_0^{T-s} \text{tr}\left(X^V Q X^V e^{A^T s} Q e^{As}\right) dt_1 ds \\ &= 4 \int_0^T (T-s) \text{tr}\left(X^V Q X^V e^{A^T s} Q e^{As}\right) ds.\end{aligned} \quad (\text{C.175})$$

This gives us two integrals – one with T and one with s . The first can be solved directly, while the second requires Theorem A.31. The solution will equal

$$T_{2,2} = 4\text{tr}\left(X^V Q X^V \left(T \bar{X}^Q(T) - \bar{X}^{\bar{X}^Q}(T)\right)\right). \quad (\text{C.176})$$

We have saved the worst for last: the term $T_{2,3}$. If we apply the same substitution and interchanging of integrals, then similarly to (C.161) we get

$$\begin{aligned} T_{2,3} &= 8 \int_0^T \int_0^{T-s} \text{tr} \left(X^V Q e^{At_1} \Delta e^{A^T(s+t_1)} Q e^{As} \right) dt_1 ds \\ &= 8 \int_0^T \text{tr} \left(X^V Q X^\Delta (T-s) e^{A^T s} Q e^{As} \right) ds. \end{aligned} \quad (\text{C.177})$$

C

We can expand $X^\Delta(T-s)$ using Theorem A.26, giving us

$$\begin{aligned} T_{2,3} &= 8 \int_0^T \text{tr} \left(X^V Q \left(X^\Delta - e^{A(T-s)} X^\Delta e^{A^T(T-s)} \right) e^{A^T s} Q e^{As} \right) ds \\ &= 8 \int_0^T \text{tr} \left(X^V Q X^\Delta e^{A^T s} Q e^{As} - X^V Q e^{A(T-s)} X^\Delta e^{A^T(T-s)} e^{A^T s} Q e^{As} \right) ds \\ &= 8 \text{tr} \left(X^V Q X^\Delta \bar{X}^Q(T) - X^V Q \left(\int_0^T e^{A(T-s)} X^\Delta e^{A^T s} Q e^{As} ds \right) \right). \end{aligned} \quad (\text{C.178})$$

The integral in the above expression is per definition (A.121) equal to $\tilde{X}^{X^\Delta e^{A^T} Q}(T)$. It can be found through Theorem A.39. Using it, we now know that $T_{2,3}$ equals

$$T_{2,3} = 8 \text{tr} \left(X^V Q X^\Delta \bar{X}^Q(T) - X^V Q \tilde{X}^{X^\Delta e^{A^T} Q}(T) \right). \quad (\text{C.179})$$

If we now merge all our results together, we directly find (C.171). \square

When $T \rightarrow \infty$, the variance of the cost (when $\alpha = 0$) naturally goes to infinity. However, it would be interesting to calculate the variance of the steady-state cost rate. When A is stable, then this will be finite. Its value can be found through the following theorem.

Theorem C.22. Consider the system $\dot{\underline{x}}(t) = A\underline{x}(t) + \underline{v}(t)$. Assume that A is stable. The variance of the steady-state cost rate (C.46) is then given by

$$\mathbb{V} \left[\lim_{T \rightarrow \infty} \frac{dJ_T}{dT} \right] = 2 \text{tr} \left((X^V Q)^2 \right). \quad (\text{C.180})$$

Proof. By taking the variance of the steady-state cost rate (C.46) we get

$$\begin{aligned} \mathbb{V} \left[\lim_{T \rightarrow \infty} \frac{dJ_T}{dT} \right] &= \lim_{T \rightarrow \infty} \mathbb{V} [\underline{x}^T(T) Q \underline{x}(T)] \\ &= \lim_{T \rightarrow \infty} \mathbb{E} \left[(\underline{x}^T(T) Q \underline{x}(T))^2 \right] - \mathbb{E} [\underline{x}^T(T) Q \underline{x}(T)]^2. \end{aligned} \quad (\text{C.181})$$

We already know the expected steady-state cost rate $\mathbb{E} [\underline{x}^T(T) Q \underline{x}(T)]$ from (C.47). By using Theorem B.19 we can also find that

$$\mathbb{E} \left[(\underline{x}^T(T) Q \underline{x}(T))^2 \right] = \text{tr} (\Psi(T) Q)^2 + 2 \text{tr} ((\Psi(T) Q)^2) - 2 (\boldsymbol{\mu}^T(T) Q \boldsymbol{\mu}(T)). \quad (\text{C.182})$$

In the limit of $T \rightarrow \infty$, we have $\boldsymbol{\mu}(T) \rightarrow \mathbf{0}$ and $\Psi(T) \rightarrow X^V$. As a result, the variance immediately reduces to

$$\mathbb{V} \left[\lim_{T \rightarrow \infty} \frac{dJ_T}{dT} \right] = 2\text{tr}((X^V Q)^2), \quad (\text{C.183})$$

which equals (C.180). Do note here that $2\text{tr}((X^V Q)^2)$ is not the same as $2\text{tr}((V \bar{X}^Q)^2)$, so there is no easy alternate version for this expression. \square

C.4.3. THE GENERAL CASE

We have saved the hardest and most general case for last. Let's consider the variance of the finite-time discounted cost function.

Theorem C.23. Consider the system $\dot{\underline{x}}(t) = A\underline{x}(t) + \underline{v}(t)$. Assume that $A_{-\alpha}$, A , A_α and $A_{2\alpha}$ are Sylvester. The variance $\mathbb{V}[J_T]$ of the finite-time cost (C.49) is then given by

$$\begin{aligned} \mathbb{V}[J_T] &= 2\text{tr}((\Delta \bar{X}_\alpha^Q(T))^2) - 2(\boldsymbol{\mu}_0^T \bar{X}_\alpha^Q(T) \boldsymbol{\mu}_0)^2 + 4\text{tr}\left(X^V Q \left(X^V \frac{e^{4\alpha T} \bar{X}_{-\alpha}^Q(T) - \bar{X}_\alpha^Q(T)}{4\alpha}\right.\right. \\ &\quad \left.\left.+ 2X_{2\alpha}^\Delta \bar{X}_\alpha^Q(T) - 2\tilde{X}_{2\alpha}^\Delta e^{A_{2\alpha}^T} Q(T)\right)\right). \end{aligned} \quad (\text{C.184})$$

Proof. For this proof we will use the same set-up as we did in Theorems C.19 and C.21. Make sure you are familiar with that first.

We once more find the term T_3 , which now equals

$$T_3 = -2\left(\boldsymbol{\mu}_0^T \left(\int_0^T e^{2\alpha t} e^{A^T t} Q e^{At} dt\right) \boldsymbol{\mu}_0\right) = -2(\boldsymbol{\mu}_0^T \bar{X}_\alpha^Q(T) \boldsymbol{\mu}_0). \quad (\text{C.185})$$

For $T_{2,1}$ we find that

$$T_{2,1} = 2\text{tr}\left(\left(\int_0^T e^{2\alpha t} \Delta e^{A^T t} Q e^{At} dt\right)^2\right) = 2\text{tr}((\Delta \bar{X}_\alpha^Q(T))^2). \quad (\text{C.186})$$

Our starting point for $T_{2,2}$, similarly to (C.175), is

$$\begin{aligned} T_{2,2} &= 4 \int_0^T \int_0^{T-s} \text{tr}\left(e^{2\alpha(2t_1+s)} X^V Q X^V e^{A^T s} Q e^{As}\right) dt_1 ds \\ &= 4 \int_0^T \left(\frac{e^{4\alpha(T-s)} - 1}{4\alpha}\right) \text{tr}\left(e^{2\alpha s} X^V Q X^V e^{A^T s} Q e^{As}\right) ds. \end{aligned} \quad (\text{C.187})$$

This integral once more has two parts, which we can solve through

$$\begin{aligned} T_{2,2} &= 4 \int_0^T \frac{1}{4\alpha} \text{tr}\left(e^{4\alpha T} e^{-2\alpha s} X^V Q X^V e^{A^T s} Q e^{As} - e^{2\alpha s} X^V Q X^V e^{A^T s} Q e^{As}\right) ds \\ &= 4\text{tr}\left(X^V Q X^V \frac{e^{4\alpha T} \bar{X}_{-\alpha}^Q(T) - \bar{X}_\alpha^Q(T)}{4\alpha}\right). \end{aligned} \quad (\text{C.188})$$

Finally there is $T_{2,3}$. Similarly to (C.177) we get

$$\begin{aligned} T_{2,3} &= 8 \int_0^T \int_0^{T-s} \text{tr} \left(e^{2\alpha(2t_1+s)} X^V Q e^{At_1} \Delta e^{A^T(s+t_1)} Q e^{As} \right) dt_1 ds \\ &= 8 \int_0^T \text{tr} \left(e^{2\alpha s} X^V Q X_{2\alpha}^\Delta(T-s) e^{A^T s} Q e^{As} \right) ds. \end{aligned} \quad (\text{C.189})$$

Expanding $X_{2\alpha}^\Delta(T-t_2)$ through Theorem A.26 and subsequently applying definition (A.121) turns this into

$$\begin{aligned} T_{2,3} &= 8 \int_0^T \text{tr} \left(e^{2\alpha s} X^V Q \left(X_{2\alpha}^\Delta - e^{A_{2\alpha}(T-s)} X_{2\alpha}^\Delta e^{A_{2\alpha}^T(T-s)} \right) e^{A^T s} Q e^{As} \right) ds \\ &= 8 \int_0^T \text{tr} \left(e^{2\alpha s} X^V Q X_{2\alpha}^\Delta e^{A^T s} Q e^{As} - e^{2\alpha(2T-s)} X^V Q e^{A(T-s)} X_{2\alpha}^\Delta e^{A^T T} Q e^{As} \right) ds \\ &= 8 \text{tr} \left(X^V Q \left(X_{2\alpha}^\Delta \left(\int_0^T e^{A_\alpha^T s} Q e^{A_\alpha s} ds \right) - \left(\int_0^T e^{A_{2\alpha}(T-s)} X_{2\alpha}^\Delta e^{A_{2\alpha}^T T} Q e^{As} ds \right) \right) \right) \\ &= 8 \text{tr} \left(X^V Q \left(X_{2\alpha}^\Delta \bar{X}_\alpha^Q(T) - \tilde{X}_{2\alpha}^{X_{2\alpha}^\Delta e^{A_{2\alpha}^T T} Q}(T) \right) \right). \end{aligned} \quad (\text{C.190})$$

Adding up all the intermediate results now directly gives us (C.184). \square

The nice part is that Theorem C.23 reduces to Theorems C.19 and C.21 in the cases of $T \rightarrow \infty$ and $\alpha \rightarrow 0$, respectively.

When $T \rightarrow \infty$, then for stable A_α and $\alpha < 0$ we will have $e^{A_\alpha T} \rightarrow 0$ and $e^{4\alpha T} \rightarrow 0$. In addition, $\bar{X}_\alpha^Q(T) \rightarrow \bar{X}_\alpha^Q$, which means that (C.184) reduces to (C.162).

When $\alpha \rightarrow 0$, something similar happens. Although here we first have to combine Theorems A.26 and A.30 to find that

$$\begin{aligned} \bar{X}_\alpha^{\bar{X}_\alpha^Q}(T) &= \bar{X}_\alpha^{\bar{X}_\alpha^Q} - e^{A_\alpha^T T} \bar{X}_\alpha^{\bar{X}_\alpha^Q} e^{A_\alpha T} = \frac{\bar{X}_\alpha^Q - \bar{X}_{-\alpha}^Q}{4\alpha} - e^{A_\alpha^T T} \frac{\bar{X}_\alpha^Q - \bar{X}_{-\alpha}^Q}{4\alpha} e^{A_\alpha T} \\ &= \left(\frac{\bar{X}_\alpha^Q - e^{A_\alpha^T T} \bar{X}_\alpha^Q e^{A_\alpha T}}{4\alpha} \right) - e^{4\alpha T} \left(\frac{\bar{X}_{-\alpha}^Q - e^{A_{-\alpha}^T T} \bar{X}_{-\alpha}^Q e^{A_{-\alpha} T}}{4\alpha} \right) + \frac{e^{4\alpha T} - 1}{4\alpha} \bar{X}_{-\alpha}^Q \\ &= \frac{\bar{X}_\alpha^Q(T) - e^{4\alpha T} \bar{X}_{-\alpha}^Q(T)}{4\alpha} + \frac{e^{4\alpha T} - 1}{4\alpha} \bar{X}_{-\alpha}^Q. \end{aligned} \quad (\text{C.191})$$

In addition, l'Hôpital's rule also tells us that

$$\lim_{\alpha \rightarrow 0} \frac{e^{4\alpha T} - 1}{4\alpha} = \lim_{\alpha \rightarrow 0} \frac{\frac{d}{d\alpha}(e^{4\alpha T} - 1)}{\frac{d}{d\alpha}(4\alpha)} = \lim_{\alpha \rightarrow 0} \frac{4Te^{4\alpha T}}{4} = T. \quad (\text{C.192})$$

Combining these two results implies that, as $\alpha \rightarrow 0$, we have

$$\frac{e^{4\alpha T} \bar{X}_{-\alpha}^Q(T) - \bar{X}_\alpha^Q(T)}{4\alpha} \rightarrow T \bar{X}^Q - \bar{X}^{\bar{X}^Q}(T). \quad (\text{C.193})$$

Through this, (C.184) immediately reduces to (C.171) as $\alpha \rightarrow 0$.

C.4.4. SOLUTIONS USING MATRIX EXPONENTIALS

The method of using Lyapunov solutions to find $\mathbb{E}[\underline{J}_T]$ and $\mathbb{V}[\underline{J}_T]$ has a significant downside: if A or A_α is not Sylvester, the theorems do not hold. By solving integrals using matrix exponentials, using the theorems from Appendix A.5, we can work around that problem.

Theorem C.24. *If we define the matrix C as*

$$C = \begin{bmatrix} -A_{2\alpha}^T & Q & 0 & 0 & 0 \\ 0 & A & V & 0 & 0 \\ 0 & 0 & -A^T & Q & 0 \\ 0 & 0 & 0 & A_{2\alpha} & V \\ 0 & 0 & 0 & 0 & -A_{-2\alpha}^T \end{bmatrix}, \quad (\text{C.194})$$

and write $C^e \equiv e^{CT}$ as

$$e^{CT} \equiv C^e = \begin{bmatrix} C_{11}^e & \cdots & C_{15}^e \\ \vdots & \ddots & \vdots \\ C_{51}^e & \cdots & C_{55}^e \end{bmatrix}, \quad (\text{C.195})$$

then we can find $\mathbb{E}[\underline{J}_T]$ and $\mathbb{V}[\underline{J}_T]$ through

$$\mathbb{E}[\underline{J}_T] = \text{tr}((C_{44}^e)^T (C_{12}^e \Psi_0 + C_{13}^e)), \quad (\text{C.196})$$

$$\begin{aligned} \mathbb{V}[\underline{J}_T] &= 2\text{tr}\left(\left((C_{44}^e)^T (C_{12}^e \Psi_0 + C_{13}^e)\right)^2 - 2(C_{44}^e)^T (C_{14}^e \Psi_0 + C_{15}^e) \right. \\ &\quad \left. - 2(\boldsymbol{\mu}_0^T (C_{44}^e)^T C_{12}^e \boldsymbol{\mu}_0\right)^2. \end{aligned} \quad (\text{C.197})$$

Proof. We first prove the expression for $\mathbb{E}[\underline{J}_T]$. We know from (C.51) that

$$\mathbb{E}[\underline{J}_T] = \text{tr}\left(\left(\int_0^T e^{2\alpha t} \Psi(t) dt\right) Q\right). \quad (\text{C.198})$$

We also know that $\Sigma(t)$ satisfies (C.17), and identically that $\Psi(t)$ satisfies

$$\Psi(t) = e^{At} \Psi_0 e^{A^T t} + \int_0^t e^{A(t-s)} V e^{A^T(t-s)} ds. \quad (\text{C.199})$$

Inserting this into (C.198) will give us

$$\begin{aligned} \mathbb{E}[\underline{J}_T] &= \text{tr}\left(\int_0^T e^{2\alpha t} e^{At} \Psi_0 e^{A^T t} Q dt + \int_0^T \int_0^t e^{2\alpha t} e^{A(t-s)} V e^{A^T(t-s)} Q ds dt\right) \\ &= \text{tr}\left(\int_0^T e^{A_{2\alpha}^T t} Q e^{At} \Psi_0 dt + \int_0^T \int_0^t e^{A_{2\alpha}^T t} Q e^{A(t-s)} V e^{-A^T s} ds dt\right) \\ &= \text{tr}\left(e^{A_{2\alpha}^T T} \int_0^T e^{-A_{2\alpha}^T(T-t)} Q e^{At} \Psi_0 dt + e^{A_{2\alpha}^T T} \int_0^T \int_0^t e^{-A_{2\alpha}^T(T-t)} Q e^{A(t-s)} V e^{-A^T s} ds dt\right). \end{aligned} \quad (\text{C.200})$$

This may seem like a completely haphazard way of rewriting the equation. It makes sense when we add in Theorems A.34 through A.37. These theorems tell us that

$$C_{44}^e = e^{A_{2\alpha} T}, \quad (\text{C.201})$$

$$C_{12}^e = \int_0^T e^{-A_{2\alpha}^T(T-t)} Q e^{At} dt, \quad (\text{C.202})$$

$$C_{13}^e = \int_0^T \int_0^t e^{-A_{2\alpha}^T(T-t)} Q e^{A(t-s)} V e^{-A^T s} ds dt. \quad (\text{C.203})$$

C

Applying them immediately results in (C.196).

Proving the expression for $\mathbb{V}[\underline{J}_T]$ is done similarly, but it will obviously be more work. First of all, we should note that C_{14}^e and C_{15}^e equal

$$C_{14}^e = \int_0^T \int_0^t \int_0^s e^{-A_{2\alpha}^T(T-t)} Q e^{A(t-s)} V e^{-A^T(s-r)} Q e^{A_{2\alpha} r} dr ds dt, \quad (\text{C.204})$$

$$C_{15}^e = \int_0^T \int_0^t \int_0^s \int_0^r e^{-A_{2\alpha}^T(T-t)} Q e^{A(t-s)} V e^{-A^T(s-r)} Q e^{A_{2\alpha}(r-q)} V e^{-A_{-2\alpha}^T q} dq dr ds dt.$$

Using this, we will find the terms T_3 and T_2 from the proof of Theorem C.23. After all, together these two terms equal $\mathbb{V}[\underline{J}_T]$. We can directly see from (C.185) that T_3 equals

$$T_3 = -2 \left(\boldsymbol{\mu}_0^T \left(\int_0^T e^{2\alpha t} e^{A^T t} Q e^{At} dt \right) \boldsymbol{\mu}_0 \right) = -2 (\boldsymbol{\mu}_0^T (C_{44}^e)^T C_{12}^e \boldsymbol{\mu}_0)^2. \quad (\text{C.205})$$

Next, we will find T_2 . This will be done very differently than previously. We begin all the way at the original definition (C.151) of T_2 , which is for the finite-time case equal to

$$T_2 = 2 \int_0^T \int_0^T \text{tr} (e^{2\alpha(t_1+t_2)} \Psi(t_2, t_1) Q \Psi(t_1, t_2) Q) dt_2 dt_1. \quad (\text{C.206})$$

For $\Psi(t_1, t_2)$ we now use a relation similar to (C.199). We can derive, similarly to (C.25), that

$$\begin{aligned} \Psi(t_1, t_2) &= e^{At_1} \Psi_0 e^{A^T t_2} + \int_0^{t_1} \int_0^{t_2} e^{A(t_1-s_1)} V \delta(s_1 - s_2) e^{A^T(t_2-s_2)} ds_2 ds_1 \\ &= e^{At_1} \Psi_0 e^{A^T t_2} + \int_0^{\min(t_1, t_2)} e^{A(t_1-s)} V e^{A^T(t_2-s)} ds. \end{aligned} \quad (\text{C.207})$$

Keep in mind here that $\Psi(t_1, t_2) = \Psi^T(t_2, t_1)$. To keep our equation manageable, we will write $\Psi(t_1, t_2)$ as $\Psi_a + \Psi_b$, where we define

$$\Psi_a = e^{At_1} \Psi_0 e^{A^T t_2}, \quad (\text{C.208})$$

$$\Psi_b = \int_0^{\min(t_1, t_2)} e^{A(t_1-s)} V e^{A^T(t_2-s)} ds. \quad (\text{C.209})$$

This turns (C.206) into three parts, being $T_2 = T_{2,aa} + T_{2,ab} + T_{2,bb}$, where each of these is defined as

$$T_{2,aa} = 2 \int_0^T \int_0^T \text{tr}(e^{2\alpha(t_1+t_2)} \Psi_a^T Q \Psi_a Q) dt_2 dt_1, \quad (\text{C.210})$$

$$T_{2,ab} = 4 \int_0^T \int_0^T \text{tr}(e^{2\alpha(t_1+t_2)} \Psi_a^T Q \Psi_b Q) dt_2 dt_1, \quad (\text{C.211})$$

$$T_{2,bb} = 2 \int_0^T \int_0^T \text{tr}(e^{2\alpha(t_1+t_2)} \Psi_b^T Q \Psi_b Q) dt_2 dt_1. \quad (\text{C.212})$$

We just need to rewrite each of these terms in a format which we can solve through matrix exponentials. For $T_{2,aa}$ this goes according to

$$\begin{aligned} T_{2,aa} &= 2 \int_0^T \int_0^T \text{tr}\left(e^{2\alpha(t_1+t_2)} e^{At_2} \Psi_0 e^{A^T t_1} Q e^{At_1} \Psi_0 e^{A^T t_2} Q\right) dt_2 dt_1 \\ &= 2 \text{tr}\left(\left(\int_0^T e^{2\alpha t} e^{At} Q e^{At} \Psi_0\right)^2\right) \\ &= 2 \text{tr}\left(\left((C_{44}^e)^T C_{12}^e \Psi_0\right)^2\right). \end{aligned} \quad (\text{C.213})$$

For $T_{2,ab}$ we get an extra integral, which means we wind up with

$$T_{2,ab} = 4 \int_0^T \int_0^T \int_0^{\min(t_1, t_2)} \text{tr}\left(e^{2\alpha(t_1+t_2)} e^{At_2} \Psi_0 e^{At_1} Q e^{A(t_1-s)} V e^{A^T(t_2-s)} Q\right) ds dt_2 dt_1. \quad (\text{C.214})$$

The main problem here is the order of integration. We want to cycle the multiplication of the elements in the integrand to have Ψ_0 at the end. When we do, we see that we encounter t_1 in the exponents first, then s and then t_2 . So we want the integration order to become $dt_2 ds dt_1$. How do we accomplish this?

We can rearrange the integrals as long as we keep the integration area the same. This area is described by $0 \leq s \leq (t_1, t_2) \leq T$. Given the order of integration that we want, we can turn the above into

$$\begin{aligned} T_{2,ab} &= 4 \int_0^T \int_0^{t_1} \int_s^T \dots dt_2 ds dt_1 \\ &= 4 \int_0^T \int_0^{t_1} \int_0^T \dots dt_2 ds dt_1 - 4 \int_0^T \int_0^{t_1} \int_0^s \dots dt_2 ds dt_1. \end{aligned} \quad (\text{C.215})$$

In the last part we have split the integral up into two integrals. The good news is that we can solve both integrals, because they are in the right order and their integration bounds are zero. We can also even split up the first integral. This turns $T_{2,ab}$ into

$$\begin{aligned} T_{2,ab} &= 4 \text{tr}\left(\left(\int_0^T \int_0^{t_1} e^{2\alpha t_1} e^{At_1} Q e^{A(t_1-s)} V e^{-At_1} ds dt_1\right) \left(\int_0^T e^{2\alpha t_2} e^{At_2} Q e^{At_2} dt_2\right) \Psi_0\right) \\ &\quad - 4 \text{tr}\left(\int_0^T \int_0^{t_1} \int_0^s e^{2\alpha(t_1+t_2)} e^{At_1} Q e^{A(t_1-s)} V e^{A^T(t_2-s)} Q e^{At_2} \Psi_0 dt_2 ds dt_1\right) \\ &= 4 \text{tr}\left((C_{44}^e)^T C_{13}^e (C_{44}^e)^T C_{12}^e \Psi_0 - (C_{44}^e)^T C_{14}^e \Psi_0\right). \end{aligned} \quad (\text{C.216})$$

That leaves us with $T_{2,bb}$ to solve. It equals

$$T_{2,bb} = 2 \int_0^T \int_0^T \int_0^{\min(t_1, t_2)} \int_0^{\min(t_1, t_2)} \text{tr} \left(e^{2\alpha(t_1+t_2)} e^{A(t_2-s_1)} V e^{A^T(t_1-s_1)} Q e^{A(t_1-s_2)} V e^{A^T(t_2-s_2)} Q \right) ds_2 ds_1 dt_2 dt_1. \quad (\text{C.217})$$

The integration area is now given by $0 \leq (s_1, s_2) \leq (t_1, t_2) \leq T$. As integration order, I will want $ds_1 dt_2 ds_2 dt_1$. This can be obtained through

$$\begin{aligned} T_{2,bb} &= 2 \int_0^T \int_0^{t_1} \int_{s_2}^T \int_0^{\min(t_1, t_2)} \dots ds_1 dt_2 ds_2 dt_1 \\ &= 2 \int_0^T \int_0^{t_1} \int_0^T \int_0^{\min(t_1, t_2)} \dots ds_1 dt_2 ds_2 dt_1 - 2 \int_0^T \int_0^{t_1} \int_0^{s_2} \int_0^{\min(t_1, t_2)} \dots ds_1 dt_2 ds_2 dt_1. \end{aligned} \quad (\text{C.218})$$

The tricky part now lies in the $\min(t_1, t_2)$ integral bound. For the bounds of the second integral, we have $t_2 \leq s_2 \leq t_1$, which means that $\min(t_1, t_2)$ reduces to t_2 . For the left-hand side integral we have no such luck. However, we can readjust the integral order to $ds_2 dt_1 ds_1 dt_2$. When we do, we can again pull off the same trick, resulting in

$$\begin{aligned} T_{2,bb} &= 2 \int_0^T \int_0^{t_2} \int_{s_1}^T \int_0^{t_1} \dots ds_2 dt_1 ds_1 dt_2 - 2 \int_0^T \int_0^{t_1} \int_0^{s_2} \int_0^{t_2} \dots ds_1 dt_2 ds_2 dt_1 \\ &= 2 \int_0^T \int_0^{t_2} \int_0^T \int_0^{t_1} \dots ds_2 dt_1 ds_1 dt_2 - 2 \int_0^T \int_0^{t_2} \int_0^{s_1} \int_0^{t_1} \dots ds_2 dt_1 ds_1 dt_2 \\ &\quad - 2 \int_0^T \int_0^{t_1} \int_0^{s_2} \int_0^{t_2} \dots ds_1 dt_2 ds_2 dt_1. \end{aligned} \quad (\text{C.219})$$

We can also note that the integrand (which we have replaced by dots for now) is symmetric with respect to both t_1 and t_2 as well as s_1 and s_2 . That means that the last two integrals from the above expression are equal. It also allows us to split up the first integral. This tells us that

$$\begin{aligned} T_{2,bb} &= 2 \text{tr} \left(\left(\int_0^T \int_0^t e^{2\alpha t} e^{A^T t} Q e^{A(t-s)} V e^{-A^T s} ds dt \right)^2 \right) - 4 \int_0^T \int_0^{t_1} \int_0^{s_2} \int_0^{t_2} \\ &\quad \text{tr} \left(e^{2\alpha(t_1+t_2)} e^{A(t_2-s_1)} V e^{A^T(t_1-s_1)} Q e^{A(t_1-s_2)} V e^{A^T(t_2-s_2)} Q \right) ds_1 dt_2 ds_2 dt_1 \\ &= 2 \text{tr} \left(\left(e^{A_{2\alpha}^T T} \int_0^T \int_0^t e^{(-A_{2\alpha}^T)(T-t)} Q e^{A(t-s)} V e^{(-A^T)s} ds dt \right)^2 \right) - 4 \text{tr} \left(e^{A_{2\alpha}^T T} \int_0^T \int_0^{t_1} \int_0^{s_2} \int_0^{t_2} \right. \\ &\quad \left. e^{(-A_{2\alpha}^T)(T-t_1)} Q e^{A(t_1-s_2)} V e^{(-A^T)(s_2-t_2)} Q e^{A_{2\alpha}(t_2-s_1)} V e^{(-A_{-2\alpha})^T s_1} ds_1 dt_2 ds_2 dt_1 \right) \\ &= 2 \text{tr} \left(((C_{44}^e)^T C_{13}^e)^2 - 2(C_{44}^e)^T C_{15}^e \right). \end{aligned} \quad (\text{C.220})$$

By putting all results together, we find (C.197). \square

You may be wondering which method works better to calculate the cost mean and variance: using Lyapunov solutions or matrix exponentials? There are a few cases when

the choice is obvious. For infinite-time problems we have to use Lyapunov solutions, while for non-Sylvester matrices A we have to use matrix exponentials.

For the rest, the same issues hold as are discussed in Appendix A.5.3. For small time steps T (smaller than roughly a few seconds) using the matrix exponential method is often better, but for larger times the matrix exponential method of Theorem C.24 becomes numerically very inaccurate. In that case using Lyapunov solutions is the obvious choice.

C.5. APPLICATIONS OF THE DERIVED EXPRESSION

We have derived methods to calculate the mean and the variance of the LQG cost \underline{J} . In this section we will look at two questions: are these equations correct and how can we use them? The first question will be answered in Section C.5.1 while we look at the second one in Section C.5.2.

C.5.1. A SIMULATION VERIFYING THE DERIVED EQUATIONS

We will set up an example LQG system

$$\dot{\underline{x}}(t) = \begin{bmatrix} 1/4 & 1/2 \\ -1/7 & -1/3 \end{bmatrix} \underline{x}(t) + \underline{v}(t), \quad (\text{C.221})$$

which has interesting eigenvalues at 0.075 and -0.16 . As distribution of the initial state \underline{x}_0 we use

$$\underline{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0) = \mathcal{N}\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}\right). \quad (\text{C.222})$$

As cost function weight we use $Q = I$, and we choose a noise intensity V of

$$V = \begin{bmatrix} 3 & -2 \\ -2 & 8 \end{bmatrix} \quad (\text{C.223})$$

Finally we use a simulation time of $T = 3$ s to let the noise and the initial state contribute roughly equally to the cost.

For this system, we run $n = 1\,000\,000$ numerical simulations with time step $dt = 0.01$ s. The mean and standard deviation of the cost \underline{J}_T , for various values of α , is shown in Table C.1. This table compares these experimental results with the theoretical results given by (C.43), (C.50), (C.171) and (C.184). Alternatively, we could of course have used the matrix exponential equations (C.196) and (C.197), but these equations give (as expected) exactly the same result, barring minor numerical differences.

In Table C.1 we can see that the experimental and the theoretical results are nearly identical. There are minor differences, but these can be explained partly by statistical variations and partly by the inaccuracies in the numerical cost integration. As a result, we can conclude that the derived equations are very likely correct.

It is also interesting to note that the mean and the standard deviation of \underline{J}_T are of the same order of magnitude. This seems strange, because for positive (semi-)definite Q and R the cost \underline{J}_T cannot be negative. The reason for this can be seen if we examine the probability density function of \underline{J}_T , which is shown in Figure C.2. Here we see that most of the time the cost \underline{J}_T is lower than the mean $\mathbb{E}[\underline{J}_T]$. However, when the cost is

Table C.1: The mean $\mathbb{E}[\underline{J}_T]$ and standard deviation $\sqrt{\text{V}[\underline{J}_T]}$ of the cost after 1000000 experiments. Experimental results are compared to the theoretical expressions given by (C.43), (C.50), (C.171) and (C.184).

Discount exponent α	0.05	0	-0.05	-0.1
Experiment mean	97.1	80.2	66.6	55.7
Theoretical mean	97.6	80.5	66.9	55.9
Experiment standard deviation	95.8	78.4	64.5	53.4
Theoretical standard deviation	94.6	77.3	63.5	52.6

C

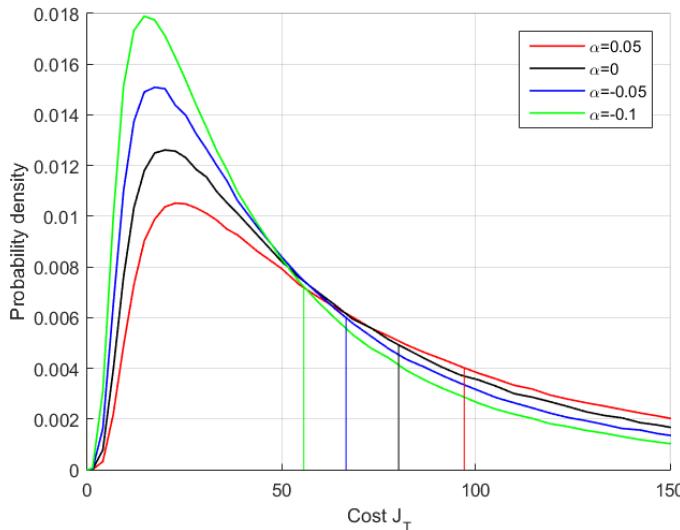


Figure C.2: The probability density function of the cost \underline{J}_T for various values of α . The vertical lines denote the means. This plot was generated using a histogram of the cost of 1000000 experiments performed with system (C.221).

higher than this mean, then it may be a lot higher, increasing the standard deviation of the distribution.

C.5.2. A SIMULATION APPLYING THE DERIVED EQUATIONS

We will now apply the derived equations. One application, in which we analyze the approximated cost function of a Gaussian process regression algorithm, can be found in Section 3.5. Here we will apply the derived expressions for controller synthesis.

In literature, people almost always use the controller that minimizes the expected value of the cost. This is done irrespective of the variance of the cost. However, if the goal is to keep the cost below a certain threshold, then this is not always the best approach.

Consider the system

$$\dot{\underline{x}}(t) = \begin{bmatrix} 1 & 0 \\ 1/20 & 1 \end{bmatrix} \underline{x}(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \underline{u}(t) + \underline{v}(t), \quad (\text{C.224})$$

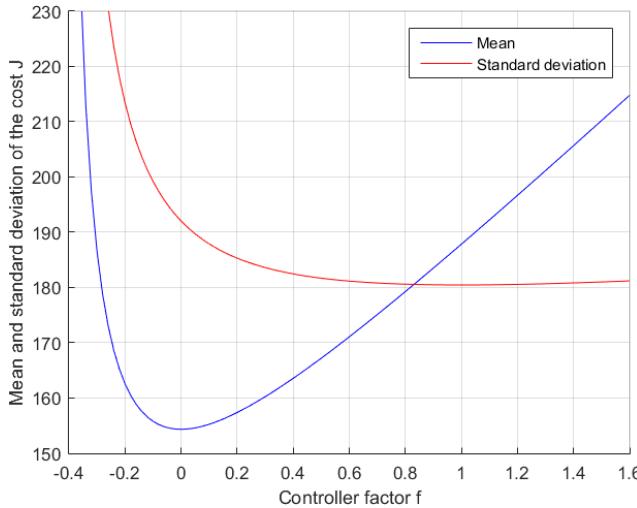


Figure C.3: The cost mean $\mathbb{E}[J]$ and standard deviation $\sqrt{\mathbb{V}[J]}$ with respect to the control matrix $F = (1 - f)F_{\text{opt}} + fF_{\text{mv}}$, for varying f , when applying the control law $\underline{u}(t) = -F\underline{x}(t)$ in system (C.224). It is clear that minimizing the mean cost does not give the same control law as minimizing the cost variance. In general, using the minimum variance controller results in more safe (stable) controllers, at the cost of larger applied inputs.

where only the first state can be controlled directly. Furthermore, the coupling between the first and the second state is very weak. This is troublesome, as the noise \underline{v} with intensity $V = I$ does excite this second state, and with $Q = I$ there will be penalties. Fixing the problem can be expensive though, because $R = I$. However, to prevent the cost from becoming very high, we use a strongly negative discount exponent $\alpha = -0.8$.

As control law we use a linear controller $\underline{u}(t) = -F\underline{x}(t)$. The optimal control matrix, found through Theorem C.15, equals $F_{\text{opt}} = \begin{bmatrix} 1.6 & 9.9 \end{bmatrix}$. It minimizes $\mathbb{E}[J]$ at $[J(F_{\text{opt}})] = 154.4$ and standard deviation $\sqrt{\mathbb{V}[J(F_{\text{opt}})]} = 192.0$. However, we can also minimize $\mathbb{V}[J]$ using a gradient descent method or a similar method. This gives the minimum-variance control matrix $F_{\text{mv}} = \begin{bmatrix} 4.4 & 30.0 \end{bmatrix}$ with mean cost $\mathbb{E}[J(F_{\text{mv}})] = 187.5$ and standard deviation $\sqrt{\mathbb{V}[J(F_{\text{mv}})]} = 180.5$. This mean cost is significantly larger than $\mathbb{E}[J]_{\text{opt}}$, making it seem as if this is a significantly worse control matrix. This is also illustrated in Figure C.3, which shows that using F_{mv} (at $f = 1$) results in a significantly higher mean cost than using F_{opt} (at $f = 0$).

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 $\bar{J} = 1500$, which is roughly ten times the mean. Using 500 000 numerical simulations, with $T = 20$ s and $dt = 0.01$ s, we have found that

$$p(J(F_{\text{opt}}) > \bar{J}) \approx 0.093\%, \quad (\text{C.225})$$

$$p(J(F_{\text{mv}}) > \bar{J}) \approx 0.062\%. \quad (\text{C.226})$$

Hence the optimal controller has about half as many threshold-violating cases as the

minimum-variance control law, which is a significantly worse result. This shows that it is sometimes useful to go for a control law which reduces the variance of the cost, and such a control law can be found using the equations we derived.

C.6. OVERVIEW OF LITERATURE AND CONTRIBUTIONS

Most of the theory in this chapter is not new. The basic control system notation of Section C.1 was set up by [Skogestad and Postlethwaite \(2005\)](#), while the more formal system notation came from [Øksendal \(1985\)](#).

In Section C.2, the discounted cost function was mentioned by [Bosgra et al. \(2008\)](#), who also made a few comments on calculating the expected cost. I have not found any resource mentioning the finite-time discounted expected cost (C.50) though.

Most of the theory from Section C.3 can also be found spread out across the book by [Anderson and Moore \(1990\)](#). My own contribution here is Theorem C.18, which I have not managed to find anywhere else either.

However, my main contribution is Section C.4, studying the distribution and specifically the variance of the LQG cost function. There is actually very few literature on this exact topic.

It is known that the LQG cost function is the sum of squared Gaussian parameters, turning it into a generalized noncentral χ^2 distribution. This distribution does not have a known probability density function, although its properties have been studied before in literature, for instance by [Rice \(1944\)](#), [Schwartz \(1970\)](#), [Sain and Liberty \(1971\)](#). Methods to approximate it are discussed by [Mathai and Provost \(1992\)](#), [Davies \(1980\)](#). However, none of this has been applied, or can directly be applied, to the LQG cost function.

When it comes to LQG control, most methods only focus on the expected cost $\mathbb{E}[J]$. Luckily there are a few exceptions to this. For instance, *Minimum Variance Control* (MVC) (see [Åström \(1970\)](#)) minimizes the variance of the output y . It does not focus on the cost function though. On the other hand, *Variance Constrained LQG* (VCLQG) (see [Collins and Selekw \(1999\)](#), [Conway and Horowitz \(2008\)](#)) minimizes the cost function subject to bounds on the variance of the state x and/or the input 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, expressions for the cost variance $\mathbb{V}[J]$ are still not given. As such, the only piece of literature focusing on the mean and variance of the LQG cost function is my own publication of [Bijl et al. \(2016\)](#).

REFERENCES

Here you find all the literature that was referred to in this thesis. I would like to note here that a reference is not the same as a recommendation. In fact, they are completely different concepts. So please never use the number of references that a paper gets as a quality criterion.

REFERENCES

Petter Abrahamsen, *A Review of Gaussian Random Fields and Correlation Functions*, Technical Report 917 (Norwegian Computing Center, Oslo, Norway, 1997).

Fabiano Daher Adegas and Jakob Stoustrup, *Structured control of LPV systems with application to wind turbines*, in *Proceedings of the American Control Conference (ACC), Montreal, Québec* (IEEE, 2012) pages 756–761.

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.

Sacha Alberici, Sil Boeve, Pieter van Breevoort, Yvonne Deng, Sonja Förster, Ann Gardiner, Valentijn van Gastel, Katharina Grave, Heleen Groenenberg, David de Jager, Erik Klaassen, Willemijn Pouwels, Matthew Smith, Erika de Visser, Thomas Winkel and Karlien Wouters, *Subsidies and costs of EU energy*, Technical Report DESNL14583 (Ecofys, 2014).

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, *Optimal Control: Linear Quadratic Methods* (Prentice Hall, 1990).

Athanasis C. Antoulas, *Approximation of Large-Scale Dynamical Systems* (Society for Industrial and Applied Mathematics (SIAM), 2005).

P. Auer, N. Cesa-Bianchi, Y. Freund and R. 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.

Francis R. Bach, *Exploring large feature spaces with hierarchical multiple kernel learning*, in *Advances in Neural Information Processing Systems 21*, edited by D. Koller, D. Schuurmans, Y. Bengio and L. Bottou (Curran Associates, Inc., 2009) pages 105–112.

- 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).
- Santiago Basualdo, *Load alleviation on wind turbine blades using variable airfoil geometry*, *Journal of Wind Engineering* 29, pages 169–182 (2005).
- Richard Bellman, *Introduction to Matrix Analysis* (SIAM, 1997).
- 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).
- James Bergstra and Yoshua Bengio, *Random search for hyper-parameter optimization*, *Journal of Machine Learning Research* 13, pages 281–305 (2012).
- James S. Bergstra, Rémi Bardenet, Yoshua Bengio and Balázs Kégl, *Algorithms for hyper-parameter optimization*, in *Advances in Neural Information Processing Systems*, Volume 24 (2011) pages 2546–2554.
- 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, *Gaussian progress regression techniques source code*, (2016a), <https://github.com/HildoBijl/GPRT>.
- Hildo Bijl, *SONIG source code*, (2016b), <https://github.com/HildoBijl/SONIG>.
- Hildo Bijl, Thomas B. Schön, Jan-Willem van Wingerden and Michel Verhaegen, *Online sparse Gaussian process training with input noise*, *Submitted for publication* (2017a).
- Hildo Bijl, Thomas B. Schön, Jan-Willem van Wingerden and Michel Verhaegen, *A sequential Monte Carlo approach to Thompson sampling for Bayesian optimization*, *Submitted for publication* (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.
- Ake Björck, *Numerical Methods for Least Squares Problems* (SIAM, 1996).
- Manuel Blum and Martin Riedmiller, *Optimization of Gaussian process hyperparameters using Rprop*, in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2013).
- Okko H. Bosgra, Huibert Kwakernaak and Gjerrit Meinsma, *Design Methods for Control Systems* (Dutch Institute of Systems and Control (DISC), 2008).
- 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).
- Colin Campbell and Yiming Ying, *Learning with Support Vector Machines* (Morgan and Claypool, 2011).
- 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.
- Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet and Sayan Mukherjee, *Choosing multiple parameters for support vector machines*, *Machine Learning* 46, pages 131–159 (2002).
- Steven C. Chapra and Raymond P. Canale, *Numerical Methods for Engineers* (McGraw-Hill, 2015).
- 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.
- Robert T. Clemen and Robert L. Winkler, *Advances in decision analysis: From foundations to applications*, (Cambridge University Press, 2007) Chapter Aggregating Probability Distributions, pages 154–176.
- 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).
- Dan Cornford, Ian T. Nabney and Christopher K. I. Williams, *Modelling frontal discontinuities in wind fields*, *Journal of Nonparametric Statistics* 14, pages 43–58 (2002).
- 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).
- Roger Daley, *Atmospheric Data Analysis* (Cambridge University Press, 1991).
- 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 (2016).
- A. M. Davie and A. J. Stothers, *Improved bound for complexity of matrix multiplication*, in *Proceedings of the Royal Society of Edinburgh: Section A Mathematics*, 2, Volume 143 (2013) pages 351–369.

- 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).
- 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.
- Petros Dellaportas and David A. Stephens, *Bayesian analysis of errors-in-variables regression models*, *Biometrics*, 51, pages 1085–1095 (1995).
- 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.
- Zvi Drezner, *Computation of the trivariate normal integral*, *Mathematics of Computation* 62, pages 289–294 (1994).
- David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum and Zoubin Ghahramani, *Structure Discovery in Nonparametric Regression through Compositional Kernel Search*, Technical Report (arXiv.org, 2013).
- 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.
- T. van Engelen and H. 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).
- Nando de Freitas, Alex Smola and Masrour Zoghi, *Regret Bounds for Deterministic Gaussian Process Bandits*, Technical Report (arXiv.org, 2012).
- Yuan Cheng Fung, *An Introduction to the Theory of Aeroelasticity* (Wiley, 1955).
- 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).

- François Le Gall, *Powers of tensors and fast matrix multiplication*, in *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)* (2014) pages 296–303.
- Alan Genz, *Numerical computation of rectangular bivariate and trivariate normal and t probabilities*, *Statistics and Computing* 14, pages 251–260 (2004).
- M. N. Gibbs, *Bayesian Gaussian Processes for Regression and Classification*, Ph.D. thesis, Department of Physics, University of Cambridge (1997).
- 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* (MIT Press, 1998) pages 493–499.
- Michael Green and David J. N. Limebeer, *Linear Robust Control* (Dover Publications, 1995).
- 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).
- Michael U. Gutmann and Jukka Corander, *Bayesian Optimization for Likelihood-Free Inference of Simulator-Based Statistical Models*, Technical Report (arXiv.org, 2015).
- William W. Hager, *Updating the inverse of a matrix*, *SIAM Rev.* 31, pages 221–239 (1989).
- Simon Haykin, *Neural Networks: A Comprehensive Foundation* (Prentice Hall, 1999).
- 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).
- James Hensman, Magnus Rattray and Neil D. Lawrence, *Fast variational inference in the conjugate exponential family*, in *Advances in Neural Information Processing Systems (NIPS)* (2012) pages 2888–2896.

- 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).
- José M. Hernández-Lobato, Matthew W. Hoffman and Zoubin Ghahramani, *Supplementary material for: Predictive Entropy Search for Efficient Global Optimization of Black-box Functions*, Technical Report (Machine Learning Group, Department of Engineering, University of Cambridge, 2014).
- Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms* (SIAM, 2002).
- Geoffrey E. Hinton and Ruslan R Salakhutdinov, *Using deep belief nets to learn covariance kernels for Gaussian processes*, in *Advances in Neural Information Processing Systems 20*, edited by J. C. Platt, D. Koller, Y. Singer and S. T. Roweis (Curran Associates, Inc., 2008) pages 1249–1256.
- Matt Hoffman, David M. Blei, Chong Wang and John Paisley, *Stochastic Variational Inference*, Technical Report (arXiv.org, 2012).
- Matthew Hoffman, Eric Brochu and Nando de Freitas, *Portfolio allocation for Bayesian optimization*, in *Uncertainty in Artificial Intelligence (UAI)* (2011) pages 327–336.
- Mike Holland, Anne Wagner, Joe Spadaro, Trevor Davies and Martin Adams, *Revealing the costs of air pollution from industrial facilities in Europe*, Technical Report (European Environment Agency, 2011).
- Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin, *A Practical Guide to Support Vector Classification*, Technical Report (Department of Computer Science, National Taiwan University, 2003).
- 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).
- A. W. Hulskamp, *The Smart Rotor Concept on Wind Turbines*, Ph.D. thesis, TU Delft (2011).
- Kiyosi Itô, *On stochastic differential equations*, *Memoirs of the American Mathematical Society* 4 (1951).
- Carl Jidling, Niklas Wahlström, Adrian Wills and Thomas B. Schön, *Linearly constrained Gaussian processes*, Technical Report (arXiv.org, 2017).
- Thomas B. Schön Johan Dahlin, Mattias Villani, *Efficient approximate Bayesian inference for models with intractable likelihoods*, Technical Report (arXiv.org, 2015).
- 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).
- A. G. Journel and C. J. Huijbregts, *Mining Geostatistics* (Blackburn Press, 1978).
- John D. Anderson Jr., *Fundamentals of Aerodynamics* (McGraw-Hill, 2010).
- 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).
- C. G. Kaufman and B. A. Shaby, *The role of the range parameter for estimation and prediction in geostatistics*, *Biometrika* 100, pages 473–484 (2013).
- C. T. Kelley, *Iterative Methods for Optimization* (SIAM, 1995).
- David A. Kendrick, *Stochastic Control for Economic Models* (McGraw-Hill, 1981).
- Kristian Kersting, Christian Plagemann, Patrick Pfaff and Wolfram Burgard, *Most likely heteroscedastic Gaussian process regression*, in *Proceedings of the 24th International Conference on Machine Learning* (2007) pages 393–400.
- 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).
- A. N. Kolmogorov, *Interpolation und Extrapolation von stationären zufälligen Folgen*, Bulletin of the Academy of Science (Nauk), USSR 5, pages 3–14 (1941).
- Peng Kou, Feng Gao and Xiaohong Guan, *Sparse online warped Gaussian process for wind power probabilistic forecasting*, *Applied Energy* 108, pages 410–428 (2013).
- H.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).
- Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra and Yoshua Bengio, *An empirical evaluation of deep architectures on problems with many factors of variation*, in *Proceedings of the 24th International Conference on Machine Learning* (2007).
- Jeanette Lawrence, *Introduction to Neural Networks* (California Scientific Software Press, 1994).

- David C. Lay, Steven R. Lay and Judi J. McDonald, *Linear Algebra and Its Applications*, fifth edition (Pearson, 2015).
- Miguel Lazaro-Gredilla and Michalis K. Titsias, *Variational heteroscedastic Gaussian process regression*, in *Proceedings of the 28th International Conference on Machine Learning (ICML)* (2011) pages 841–848.
- 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 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).
- D. J. C. MacKay, *Neural networks and machine learning*, (Springer, 1998) Chapter Introduction to Gaussian processes, pages 133–165.
- David J. C. MacKay, *Comparison of approximate methods for handling hyperparameters*, *Neural Computation* 11, pages 1035–1068 (1999).
- Jan R. Magnus and Heinz Neudecker, *Matrix Differential Calculus with Applications in Statistics and Econometrics* (Wiley, 1999).
- 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).
- K. V. Mardia and R. J. Marshall, *Maximum likelihood estimation of models for residual covariance in spatial regression*, *Biometrika* 71, pages 135–146 (1984).
- Arakaparampil M. Mathai and Serge B. Provost, *Quadratic Forms in Random Variables* (Taylor & Francis, 1992).
- G. Matheron, *The intrinsic random functions and their applications*, *Advances in Applied Prob* 5, pages 439–468 (1973).
- 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.
- 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.
- Peter Mörters and Yuval Peres, *Brownian Motion* (Cambridge University Press, 2010).
- Robb J. Muirhead, *Aspects of Multivariate Statistical Theory* (Wiley, 2005).
- Iain Murray and Ryan Prescott Adams, *Slice sampling covariance hyperparameters of latent Gaussian models*, Technical Report (arXiv.org, 2010).
- S. T. Navalkar, L. O. Bernhammer, J. Sodja, E. van Solingen, G. A. M. van Kuik, and J.-W. 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.
- Radford M. Neal, *Bayesian Learning for Neural Networks* (Springer, 1996).
- 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).
- A. O'Hagan and J. F. C. Kingman, *Curve fitting and optimal design for prediction*, *Journal of the Royal Statistical Society, Series B (Methodological)* 40, pages 1–42 (1978).
- 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).
- 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).
- 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 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).
- 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.
- F.J. Savenije and J.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).
- Bernhard Schölkopf and Alexander J. Smola, *Learning with Kernels* (MIT Press, 2002).
- Thomas B. Schön and Fredrik Lindsten, *Learning of dynamical systems - particle filters and Markov chain methods (draft manuscript)* (2017).
- 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, *Bayesian model selection for support vector machines, Gaussian processes and other kernel classifiers*, in *Advances in Neural Information Processing Systems 12*, edited by S. A. Solla, T. K. Leen and K. Müller (MIT Press, 2000) pages 603–609.
- 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).

- K. Selvam, S. Kanev, J. W. van Wingerden, T. van Engelen and M. Verhaegen, *Feedback-feedforward individual pitch control for wind turbine load reduction*, *International Journal of Robust and Nonlinear Control* (2008), [10.1002/rnc.1324](https://doi.org/10.1002/rnc.1324).
- 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).
- Patricia L. Smith, *Splines as a useful and convenient statistical tool*, *The American Statistician* 33, pages 57–62 (1979).
- 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).
- Jasper Snoek, Hugo Larochelle and Ryan Prescott Adams, *Practical Bayesian optimization of machine learning algorithms*, in *Neural Information Processing Systems* (2012).
- 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).
- Michael L. Stein, *Interpolation of Spatial Data* (Springer, 1999).
- Ingo Steinwart and Andreas Christmann, *Support Vector Machines* (Springer, 2008).
- Tilo Strutz, *Data Fitting and Uncertainty* (Springer, 2016).
- 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.
- S. Sundararajan and S. Sathiya Keerthi, *Predictive approaches for choosing hyperparameters in Gaussian processes*, in *Advances in Neural Information Processing Systems 12*, edited by S. A. Solla, T. K. Leen and K. Müller (MIT Press, 2000) pages 631–637.

- Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 1998).
- Andreas Svensson, Johan Dahlin and Thomas B. Schön, *Marginalizing Gaussian process hyperparameters using sequential monte carlo methods*, in *Proceedings of the IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)* (2015).
- 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).
- 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>.
- Philip Duncan Thompson, *Optimum smoothing of two-dimensional fields*, *Tellus A* 8 (1956).
- 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).
- 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).
- 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).
- Hermann-Josef Wagner and Jyotirmay Mathur, *Introduction to Wind Energy Systems* (Springer, 2013).

- Niklas Wahlström, *Modeling of Magnetic Fields and Extended Objects for Localization Applications*, Ph.D. thesis, Linköping University (2015).
- Niklas Wahlström, Patrik 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).
- Norbert Wiener, *Extrapolation, Interpolation and Smoothing of Stationary Time Series* (MIT Press, 1949).
- Christopher K. I. Williams and Carl E. Rasmussen, *Gaussian processes for regression*, in *Advances in Neural Information Processing Systems*, Volume 8 (MIT Press, 1996) pages 514–520.
- Jan-Willem van Wingerden, *Control of Wind Turbines with 'Smart' Rotors: Proof of Concept & LPV Subspace Identification*, Ph.D. thesis, TU Delft (2008).
- Chang-Hee Won, Cheryl B. Schrader and Anthony N. Michel, *Advances in Statistical Control, Algebraic Systems Theory, and Dynamic Systems Characteristics* (Birkhäuser Boston, 2008).

INDEX

Symbols	
χ^2 -squared distribution	261
A	
Acquisition function	174
Adaptive quadratures	149
Air density	34
Alternate Lyapunov equation	222
Alternate Lyapunov solution	222
Ancestor (particles)	164
Automatic relevance determination	45
Azimuth (Wind turbine)	187
B	
Basis function	25
Bayes' theorem	41
Bayesian optimization	173
Big O notation	75
Blockwise matrix inverse	208
Brownian motion	288
C	
Center of gravity (airfoil)	33
Challenge (particles)	152
Challenger particle	152
Champion particles	152
Cholesky decomposition	256
Cognitive load	xiii
Coleman transformation	187
Complexity penalty	43
Computational requirements	74, 75
Conditional independence	273
Constrained Gaussian process	
regression	55
Constrained Gaussian process	
regression equation	56
Constraint	55
Continuous entropy	177
Continuous Lyapunov equation	222
D	
Controller parameters	57
Controller settings	57
Coppersmith-Winograd algorithm	75
Correlation function	18
Cost matrix	302
Cost rate	296
Covariance function	18, 23
Covariance matrix	255
Cumulative density function	148, 256
Cumulative weight chart	162
Cut-off covariance function	106
Cyclic property	204
E	
Damping coefficients	33
Data fit term	43
Defensive importance sampling	165
Delta distribution	246
Delta function	246, 288
Derivative covariance function	28
Derivative mean function	28
Derivative of a Gaussian process	28
Deterministic training conditional	
assumption	104
Differential entropy	177
Discount exponent	297
Discount factor	58
Discounted cost function	296
Discrete Lyapunov equation	222
Distribution	244
F	
Entropy	110, 177
Entropy search	179
Entropy search acquisition function ..	179
Error (Gaussian process optimization) ..	173
Error function	153
Error minimization	174
Errors-in-variables regression	141

Evidence maximization	92	Hyper-prior	41
Expected improvement acquisition function	176	Hyperparameters	40
Expected posterior variance	93	I	
Expected squared state	293	Importance sampling	159
Expected squared value	261	Independence (random variables)	246
Expexted value acquisition function ..	174	Index variable	23
Exploitation	174	Individual pitch control	7
Exploration	174	Inducing function values	76
Exploration parameter	175	Inducing input points	76
Inducing input set	76	Information matrix	17
Initial state	289	Input penalty matrix	299
Feature function	51	Input vector	25
Feature vector	51	Instantaneous regret	173
Feedback matrix	301	Instantaneous reward	57
Finite-time cost function	295	Integer (Algorithms)	134
FITC algorithm	81	Integral of a Gaussian process	30
FITC training equation	81	Integrity of a prediction	69
Flutter	5, 36	Inverse Coleman transformation	188
Fully independent training conditional algorithm	81	Invertible matrix	222
Fully independent training conditional assumption	81	Itô isometry	291
Function bias	51	Iterative feedback tuning	195
Function offset	51	J	
		Joint probability density function	245
G		K	
Gauss-Jordan elimination	75	Kernel density estimation	159
Gaussian distribution	255	Kernel function	18
Gaussian exponential	212	Kronecker product	206
Gaussian mixture model	110	Kullback-Leibler divergence	178
Gaussian probability density function	13, 255	L	
Gaussian process	23	LDU decomposition	208
Gaussian process optimization	173	Learning index	88
Gaussian process regression	2	Length scale	18
Gaussian process regression equation	22, 74	Length scales	13
Generalized χ -squared distribution	261	Life cycles	189
Gust	6	Lift coefficient	34
H		Likelihood	41
Heteroscedasticity	67	Limit cycle	36
Hidden subsidies	4	Linear covariance function	49
Hurwitz matrix	222	Linear parameter-varying control	8
		Linear quadratic gaussian control	288

Log-likelihood	43	N	
Lyapunov constant	222	NARX system	130
Lyapunov equation	221	Noisy input Gaussian process regression	
Lyapunov solution	222	algorithm	118
Lyapunov solution properties	224	Noncentral χ^2 -squared distribution	261
M		Normal distribution	255
Marginal likelihood	41, 161	Normalization constant	43
Marginalization	245	Null distribution	178, 247
Matlab scripts	xiv	O	
Matrix covariance function	27	Observation likelihood	41, 161
Matrix derivative	205	Observer	307
Matrix exponentials	232	Observer gain matrix	307
Matrix inverse	207	Optimal cost matrix	301
Matrix inversion lemma	209	Optimal discounted cost matrix	304
Matrix of squared length scales	25	Optimal discounted feedback matrix	306
Maximum a posteriori method	42	Optimal feedback matrix	301
Maximum distribution	146	Optimal input	146
Maximum entropy probability distribution	111	Optimal observer gain matrix	307
Maximum likelihood method	42	Optimal output	146
Mean function	23	Output function	161
Mean squared error	133	Overfitting	43
Mean variance	134	P	
Mean vector	255	Parameterized control law	57
Measured function values	20	Partially independent training conditional algorithm	81
Measured value	13	Partially independent training conditional assumption	81
Measurement input set	20	Particle	151, 157
Measurement noise	13, 306	Piecewise smooth covariance function	48
Measurement points	20	PITC algorithm	81
Memory requirement	75	Pitch	32
Miner's rule	191	Pitch angle (blades)	7
minimal cost variance control	330	Pitch-plunge system	32, 59
Minimum variance control	330	Plunge	32
Mixture importance sampling	164	Portfolio methods	179
Moment matching	53, 110	Posterior covariance function	24
Moments (Probability distribution) ...	110	Posterior distribution	14
Monte Carlo	110, 157	Posterior hyperparameter distribution	41
Monte Carlo maximum distribution algorithm	151	Posterior mean function	24
Multi-blade coordinate transformation	187	Posterior sampling	180
Multinomial resampling	162	Posterior state distribution	161
Multiplication matrix	222	Power form	261
Multivariate Gaussian distribution	15	Precision matrix	17
		Predicting	19

Prediction step (regression)	75	Separation principle	309
Predictive entropy search	198	Sequential Monte Carlo samplers	155
Preparation step	75	Simpson's formula	241
Prescribed degree of stability	297	Sparse Gaussian process prediction equation	78
Prior covariance function	24	Sparse Gaussian process regression	74, 78
Prior distribution	12, 252	Sparse Gaussian process training equation	77
Prior hyperparameter distribution	41	Sparse online noisy input Gaussian process regression	124
Prior knowledge	12	Spring constant	32
Prior mean function	24	Squared exponential correlation function	18
Prior state distribution	161	Squared exponential covariance function	19, 25
Probabilistic global optimization	174	Stable matrix	222
Probability density function	12, 244	Standard deviation function	174
Probability matching	180	Standard Gaussian distribution	255
Probability of improvement acquisition function	175	Standard probability density function	255
Process noise	288	State estimate	307
Prosthaphaeresis formula	241	State estimate update law	307
Pseudo measurement distribution	97	State estimation error	307
Q		State evolution	289
Quadratic cost function	63	State penalty matrix	294
Quadratic power form	261	State transition function	157
Quadratic value function	63	State vector	288
R		Stationary particle distribution	153
Random variable	244	Steady-state cost rate	296
Random vector	15, 244	Steady-state error covariance matrix	307
Rare events	159	Steady-state state covariance matrix	291
Recommendation (acquisition func.)	179	Stochastic variational inference	104
Regret	174	Stratified resampling	164
Regret minimization	174	Subset of data approach	127
Regularization	43	Subset of data method	104
Relative entropy	178	Surface area (wing)	34
Resampling	162	Surprise (measurements)	106
Reward function	58	Sylvester equation	223
Riccati equation	301, 307	Sylvester matrix	222
Root bending moment	187	System input	299
Root mean squared error	137	System matrix	288
Round of challenges	152	Systematic resampling	164
Runtime requirement	75	T	
S		Thompson sampling	180
S-N Curve	189	Tower shadow	6
Schur complement	207	Trace operator	204
Science per cognitive load	xiii		
Self-normalized importance sampling	161		

Training step (regression)	75
Trial function values	20
Trial input set	20
Trial points	20
Trigonometric addition formula	241
True value	13
Try-out points	173
Turbulence	6

U

Uncertainty incorporating squared exponential covariance function .	116
Unit vector	301
Unmerging distributions	77, 267
Upper confidence bound acquisition function	176

V

Value	58
Value function	58
Variance constrained LQG	330
Vectorization	206

W

Weight (particles)	159
Weight degeneracy	162
Weight vector	49
Weighted mean reward	58
Wind shear	6
Woodbury matrix identity	209
Writing mindset	xii