University of Cambridge

Gonville & Caius College

Machine Learning for Control:

Incorporating Prior Knowledge

by

Joseph Alexander Hall

Supervisor:
Prof. Jan Maciejowski

Advisor:
Dr. Carl Rasmussen

April 2013

A dissertation submitted for the degree of

Doctor of Philosophy

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

"Aye, I suppose I could stay up that late."

James Clerk Maxwell
after being informed of a compulsory 6 a.m
church service on his arrival at Cambridge University.

Dedicated to Pa.
Thank you for paving the way.

Abstract

We tackle the problem of learning control policies for systems in which knowledge of the underlying dynamics is unavailable or incomplete. This problem is given particular attention by the fields of Machine Learning and Control, in which interaction with the system is required in order to determine such a policy. Specifically we address the problem of how to incorporate useful prior knowledge about the system or task at hand and investigate what effect this may have on the learning process. To do this we build on the probabilistic learning control framework presented by Deisenroth & Rasmussen (2011).

The first contribution of this thesis is to make use of prior structural knowledge regarding the dynamical equations governing the system in the form of known, or approximate, relationships between subsets of the state variables. We achieve this through the use of multiple dynamics models. Common examples of such prior knowledge are position-velocity relationships and tracking a known set, or distribution, of reference signals. Through demonstration of this technique on some simulated control problems we show situations in which forcing these prior beliefs into our model can have beneficial or adverse effects on learning.

The second contribution is a class of priors that can be placed directly over a space of dynamics functions. These priors encode the fact that the discrete-time system we are dealing with is generated through sampling of some underlying continuous-time process, a commonly encountered situation. Often the underlying continuous process will have some useful structure in the form of invariant, linear or additively separable relationships. This structure gets lost when we move to the discrete system. We define a prior which is able to exploit any such underlying structure for improved predictive performance given only sampled data.

The final contribution is to provide a qualitative comparison of the learning framework we have developed with a standard control theoretic approach. The comparison takes the form of a discussion outlining what knowledge is required of the user in order to implement a given method and what the potential pitfalls of each approach are. This discussion takes place in the context of two case study examples: a simulated robotic unicycle and a process control problem.

Acknowledgements

This is an unashamedly extensive (but far from exhaustive) list of acknowledgements to people who have helped and influenced me along the journey which has culminated in this thesis. I will start by saying that were it not for a few individuals I doubt I would have even considered doing a PhD. In particular, thank you Peter Doig and Linda McInnes for inspiring me about Physics and Mathematics during my time at Knox Academy. And Dr. Dave Anderson, at the University of Glasgow, thank you for your enthusiasm, your encouragement and for first sparking the idea of doing research.

I wish to sincerely thank my supervisor, Prof. Jan Maciejowski, for giving me the freedom to pursue whichever avenue of research I felt most excited about and for gently guiding me through the subsequent rough waters. I have also received much input from my advisor, Dr. Carl Rasmussen, thank you for your patience in explaining many confusing concepts to me and taking the time to invest in my research and development. Dr. Ed Hartley, thank you for pouring your time and effort into answering my near continuous stream of questions in the early days of my studies. The beautiful figures in this thesis are a direct product of the wisdom you have passed on. For technical discussions and insightful comments I am greatly indebted to Marc Deisenroth, Andrew McHutcheon, Roger Frigola and Philip Hennig. Thank you for your help.

Rich Pates and Alberto Carignano, the PhD experience can often be depressing, full of uncertainty and confusion, but you have made it fun and light-hearted with your wit, banter and the general making light of each others' research plights (admittedly the time and effort we put into this amusement could have been better invested in actually solving these problems, but where is the fun in that). This good atmosphere has been perpetuated by the rest of the Control Group, so thank you Kris, Dave, Kaoru, Alison, Rohan, Panos, Ellie, Tim, Dariusz, Alex, Carlos, Luisa, Ian, Muyiwa, Amy, Marco, Xiaoke, Ye and Ze for being great colleagues. Anna Langley and Patrick Gosling, thank you for solving all my computer problems, I would still be doing my PhD now were it not for your expertise. And thank you Rachel Fogg, for keeping the whole division from descending into disarray, making sure we all stay sane and that we make wise choices.

Outside of research, I have been privileged to be a part of Gonville & Caius College, who have provided me with more than my fair share of travel funding and free port. But the highlight of my time at Caius was being able to row. In particular I wish to thank the M2 crew of May Bumps 2010 for their camaraderie and fighting spirit. Housemates also make a huge difference in the day-to-day grind. I have been blessed with getting to live with the likes of Luke Smith, Tom Lee and Josh Noblett. Thank you for your friendship, support, laughs and shenanigans. My Cambridge experience would have been a poor one indeed were it not for you.

As well as great housemates I am also grateful to the friends I have made since embarking on this journey. Thank you to Matt & Mickey (and Malachi & Hero) Peacock, Mike & Belinda (and Stjohn) Bendzulla, Simon & Christy Munday, Wei-Jin Goh, Bob & Jacquie (and Lizzie & Johnny) Storey, Faith & Simon (and Adlai) Dwight, Philippe & Christine Lafon, Chris Page, Matthias Weber, Ben & Meri Leggott, Keith MacPherson, Phil & Carol Wilthew, Ben & Faith Rawley, Olly & Katherine Cole, Matt & Sarah Fox, Karen Carr & the members of *Supergroup* (#crankthatbass) and those in my King's Arms and City Church home groups. You have all helped me through the inevitable ups and downs of research, simply by being yourselves. And of course, there are old friends whose influence just goes on and on. So thank you Andrew-John & Elli McAdam, Luke & Mim Emerton, Harry & Esther Droop, Nick & Jan Treadgold, Phil & Lizzie Ford and Andy & Theresa Merrick (now Dad and Mum-in-law!). Your friendship gives me life.

As wonderful as it is having a family away from family, it is also amazing to have the support of your actual family! Mum & Dad, you have invested so much in me. Your continual encouragement and enthusiasm for the work I do has spurred me on again and again. My sisters, Lisa and Sarah, you inspire me with your exploits. Gran, you are so generous in your love towards me, even diligently reading my technical papers to make sure I haven't made any mistakes. I treasure your words and your support. Grandad, thank you for making Engineering (and all that we did together) fun. I miss you. Uncle George & Auntie Lorna, thank you for your prayerful support of me during my time here, I can tell you now that it has made the world of difference. Granny Lee, thank you for the love and kind words that you share so freely. And Pa, to whom this thesis is dedicated, your passion for knowledge and wisdom is an inspiration to me. It is an honour to follow in your footsteps.

Kezia. My words cannot do justice to the support and love you have given me over the past months and years. Thank you for being a wonderful acquaintance, friend, girlfriend, fiancèe and wife, all in the course of a PhD. Thank you for giving me courage and hope when things looked bleak. Thank you for the strange and exotic food you make. Thank you for making sure I went outside. Thank you for pursuing what I think and how I feel. Thank you for letting me be me. Thank you for being the best distraction. I love you.

Finally, thank you Jesus. Thank you for chasing after me. Thank you for giving me a hope and a purpose. Thank you for your overwhelming goodness towards me, even when I was losing hope or was angry and frustrated with you. Thank you for giving me freedom. Thank you for bringing me here to do me good and not to shame or disappoint me. I look forward to the next

adventure . . .

Nomenclature

Sets

$\mathbb{R}^D$	the set of vectors of dimension D with real elements
$\mathbb{Z}_{[a,b]}$	the set of integers in the range $[a, b]$

Operators

$\otimes$	the Kronecker product
$\circ$	the Hadamard product, element-wise matrix multiplication
$\text{diag}\{\cdot\}$	returns a diagonal matrix with elements given by the elements of the input vector, given a matrix the operator will set all off diagonal elements to zero
$\text{vec}(\cdot)$	takes a matrix and returns a vector containing the concatenation of the columns, we often employ the shorthand $\vec{\mathbf{X}} = \text{vec}(\mathbf{X})$

Vectors and Matrices

$x[i]$	the i^{th} element of the vector $\mathbf{x}$
$\mathbf{x}[\mathbf{i}]$	a sub-vector of $\mathbf{x}$ consisting of the elements indexed by the elements of vector $\mathbf{i}$
$X[i, j]$	the $(i, j)^{\text{th}}$ element of the matrix $\mathbf{X}$
$X[:, i]$	the i^{th} column of the matrix $\mathbf{X}$
$\mathbf{X}[\mathbf{i}, \mathbf{j}]$	a sub-matrix of $\mathbf{X}$ consisting of rows and columns indexed by the elements of vectors $\mathbf{i}$ and $\mathbf{j}$ respectively

Random Variables

$p(\cdot)$	probability density function over the vector space containing the input argument
$\mathbb{E}_{\mathbf{x}}[\cdot]$	expectation of the input argument with respect to $\mathbf{x}$, e.g. $\mathbb{E}_{\mathbf{x}}[\mathbf{y}] = \int \mathbf{y}p(\mathbf{x})\mathrm{d}\mathbf{x}$
$\mathrm{cov}_{\mathbf{x}}[\cdot,\cdot]$	covariance between two vectors with respect to $\mathbf{x}$, related to the expectation by $\mathrm{cov}_{\mathbf{x}}[\mathbf{y},\mathbf{z}] = \mathbb{E}_{\mathbf{x}}[\mathbf{y}\mathbf{z}^\top] - \mathbb{E}_{\mathbf{x}}[\mathbf{y}]\mathbb{E}_{\mathbf{x}}[\mathbf{z}]^\top$, we use the shorthand $\mathrm{cov}_{\mathbf{x}}[\mathbf{y}] = \mathrm{cov}_{\mathbf{x}}[\mathbf{y},\mathbf{y}]$ for the variance
$\sim$	a random vector $\mathbf{x}$ sampled from the distribution $p(\mathbf{x})$ is denoted $\mathbf{x} \sim p(\mathbf{x})$
$\perp\!\!\!\perp$	conditional independence, $\mathbf{x} \perp\!\!\!\perp \mathbf{y}\vert\mathbf{z}$ means $\mathbf{x}$ is conditionally independent of $\mathbf{y}$ given $\mathbf{z}$ or $\mathrm{cov}[\mathbf{x},\mathbf{y}\vert\mathbf{z}] = \mathbf{0}$
$\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$	a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, sometimes the random variable $\mathbf{x}$ will be given explicitly as $\mathcal{N}(\mathbf{x}\vert\boldsymbol{\mu},\boldsymbol{\Sigma})$

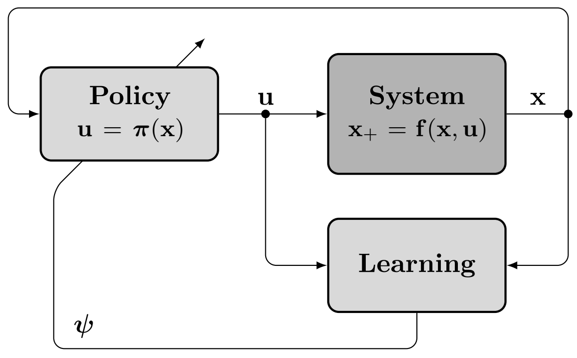
Contents

CHAPTER 1 Introduction

1.1 Motivation

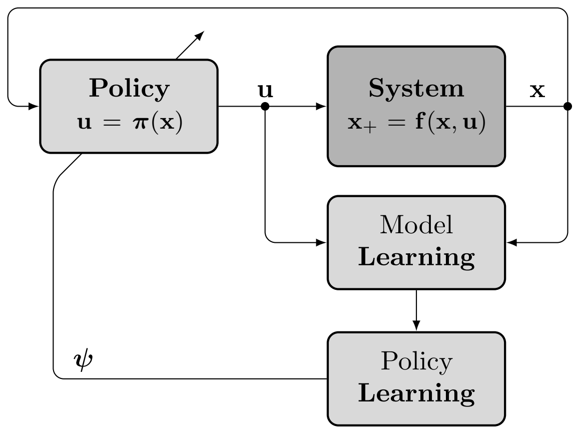
Many real world applications use a feedback control policy to determine appropriate actions to apply to a dynamical system in order to achieve a given task. This policy makes its decision based on a set of measurements from the system. The procedure for designing such a policy is usually based on a mathematical model of the system in which simplifying assumptions may have been employed in the derivation. In many cases the equations of motion governing the system are unknown, or partially unknown, and building up a model based on physical intuition may be difficult. Therefore the question of how to design a control policy based on a combination of data obtained directly from interaction with the system and physical insight is an important one.

There are many techniques available to tackle problems of this kind based on different assumptions and levels of generality. There are also many issues that must be addressed. Do we adopt a model-based or model-free scheme? If a model-based scheme is used, what structure do we place on the dynamics of the system a priori? How do we deal with modelling uncertainty? How do we incorporate useful prior information about the system that we do know? In this thesis we define a probabilistic learning framework which addresses each of these issues, but with particular focus given to the latter. In particular, what are the advantages and potential pitfalls of forcing our prior beliefs into the learning framework.



Policy
$\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$
u
System
$\mathbf{x}_+ = \mathbf{f}(\mathbf{x},\mathbf{u})$
x
Learning
$\boldsymbol{\psi}$

(a) Model-free learning



Policy
$\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$
u
System
$\mathbf{x}_+ = \mathbf{f}(\mathbf{x},\mathbf{u})$
x
Model **Learning**
Policy **Learning**
$\boldsymbol{\psi}$

(b) Model-based learning

Figure 1.1: These figures demonstrate the high level distinction between a model-free and model-based framework for learning control. Both frameworks are used for learning a parameterised discrete-time state-feedback control policy $\boldsymbol{\pi}$, mapping the system state $\mathbf{x}$ to the control action $\mathbf{u}$, to achieve some task or meet some performance criteria. The system has dynamics governed by the unknown function $\mathbf{f}$ which maps the current state and action to the state at the following timestep $\mathbf{x}_+$.

1.2 The Learning Framework

The general framework we consider in this thesis is shown in Fig. 1.1(a). The system dynamics (which we consider to be in discrete-time) are governed by the function $\mathbf{f}$ which we consider to be initially unknown or partially unknown. The goal is then to find a control policy $\boldsymbol{\pi}$, with free parameters $\boldsymbol{\psi}$, to achieve a task. This task may be encoded in the scalar stage-cost c. System states are given by $\mathbf{x}$ while actions are given by $\mathbf{u}$. It is important to be clear at this stage that we have restricted our attention to scenarios in which we have access to measurements which completely determine the current state of a system. Therefore we do not explicitly deal with the problem of state estimation.

We consider now the distinction between *model-free* and *model-based* methods. The view we adopt for this thesis is that in a model-based method we can split the learning block into two parts: model learning and policy learning, as depicted in Fig. 1.1(b). The model provides some kind of representation of the unknown system dynamics function $\mathbf{f}$ which the learning block then makes use of when choosing the policy parameters. The learned model has the important property that it is able to generalise the information it has gained through interaction with the system based on some prior belief about the system dynamics. An example of such prior belief is the notion that states and actions that are close to each other, in some sense, will elicit similar responses from the system. We then take the view that a model free method adopts no such representation of the system dynamics during the design process.

What goes on in this learning process is the subject of much research. The two major fields which address the problem of designing algorithms for the learning block (and modelling block) are adaptive control and reinforcement learning, subfields of control theory and machine learning respectively. Although each field has very different roots, in recent years there has been much overlap in the problems they address. So before outlining our own solution to the control of initially unknown, or partially unknown, systems we shall consider a number of methods from the wide body of available literature across these two fields.

CHAPTER 2 Review of the Literature

2.1 Background

Control theory is a discipline concerned with the behaviour of dynamical systems that can be specified by their inputs, states and outputs. The aim of control is to apply inputs to a system such that a desired output response is achieved. One of the standard assumptions in control theory is that a mathematical model of the system is available that describes its dynamic behaviour to a reasonable degree of accuracy. This model is then used as though it was the real system to design feedback control policies.

Among the most compelling arguments for the use of feedback rather than simply open loop control is that the effect of modelling uncertainty is reduced. Sometimes the discrepancy is so large however that the problem must be given further consideration. This is the subject of adaptive control. Adaptive control combines data obtained from interaction with the system with a control design procedure in order to adapt the policy online. The methods provided by the adaptive control community tend to place quite rigid assumptions on the form of the dynamics of the system.

Policy design for initially unknown systems has also been the subject of reinforcement learning. This is a framework born out of the Artificial Intelligence community and tackles the problem of learning a policy that is optimal with respect to some cost function given little or no prior knowledge about the system or environment. Originally these reinforcement learning methods were designed for systems with discrete state and input spaces, but recent advances have allowed algorithms for continuous systems to be defined.

2.2 Adaptive Control

2.2.1 Introduction

The methods in adaptive control can be split into *direct* and *indirect* algorithms. Direct adaptive control compares the system output response with some desired response and proceeds to adapt the controller parameters directly. Indirect adaptive control seeks to estimate unknown parameters in some system model and then uses this model to perform online controller design according to some design methodology. For a review of the fundamentals of the field, the reader is referred to the book by Åström & Wittenmark (1994) or the more recent text by Ioannou & Fidan (2006).

As a side note, many reinforcement learning algorithms can in fact be viewed as direct adaptive control algorithms, as observed by Sutton *et al.* (1992). While indirect methods are equivalent to model-based algorithms, since it means that an internal model of the system is maintained and used for policy design. We shall now proceed to give an overview of a selection of the methods available in the literature for control of initially unknown or partially unknown systems.

2.2.2 Model Reference Adaptive Control

Traditional Approach

Model Reference Adaptive Control (MRAC) is a mechanism for adjusting policy parameters based on the error between the output of the actual plant and some desired plant $\boldsymbol{\epsilon} = \mathbf{x} - \mathbf{x}_m$. Fig. 2.1 shows the general layout of an MRAC scheme, which generally operates on continuous-time systems. The original approach to MRAC was the celebrated "MIT rule" $\dot{\boldsymbol{\psi}} = -\gamma \boldsymbol{\epsilon} \nabla_{\boldsymbol{\psi}} \boldsymbol{\epsilon}$ where γ is a scalar to control the parameter rate of change and $\nabla_{\boldsymbol{\psi}} \boldsymbol{\epsilon}$ is the so-called sensitivity derivative which must be approximated in some manner. This is equivalent to a continuous-time policy-gradient algorithm which finds the minimum of the cost function $J = \frac{1}{2}\boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}$ (discussed later). The currently favoured methods are that of Lyapunov-based design and the backstepping design of Krstić *et al.* (1995).

As an example, the Lyapunov-based design for MRAC is based on stability theory and in particular, Lyapunov's Direct Method. It consists of the following steps

(1) Define a parameterised policy

(2) Derive the error equation as a function of policy parameters

(3) Find a Lyapunov function from which a parameter updating law can be defined such that the error will go to zero

Now apply this to the linear system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$. We want the system to respond according to the model $\dot{\mathbf{x}}_m = \mathbf{A}_m \mathbf{x}_m + \mathbf{B}_m \mathbf{r}$ using some policy $\mathbf{u} = \boldsymbol{\pi}_1(\mathbf{r}) - \boldsymbol{\pi}_2(\mathbf{x})$ with free parameters

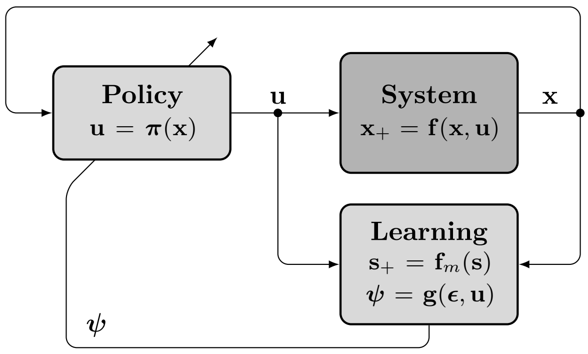

Policy
$\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$
u
System
$\mathbf{x}_+ = \mathbf{f}(\mathbf{x}, \mathbf{u})$
x
Learning
$\mathbf{s}_+ = \mathbf{f}_m(\mathbf{s})$
$\boldsymbol{\psi} = \mathbf{g}(\boldsymbol{\epsilon}, \mathbf{u})$
$\boldsymbol{\psi}$

Figure 2.1: Layout of an MRAC scheme where $\mathbf{f}_m$ describes the ideal closed loop system and $\boldsymbol{\epsilon}$ is the error between the actual $\mathbf{x}$ and ideal response $\mathbf{s}$. The adaptation function $\mathbf{g}$ can be determined through a variety of methods including stability theory.

$\boldsymbol{\psi}$. The closed loop system responds according to $\dot{\mathbf{x}} = \mathbf{A}_c(\boldsymbol{\psi})\mathbf{x} + \mathbf{B}_c(\boldsymbol{\psi})\mathbf{r}$. Therefore the error equation $\boldsymbol{\epsilon} = \mathbf{x} - \mathbf{x}_m$ has time derivative $\dot{\boldsymbol{\epsilon}} = \mathbf{A}_m\boldsymbol{\epsilon} + (\mathbf{A}_c(\boldsymbol{\psi}) - \mathbf{A}_m)\mathbf{x} + (\mathbf{B}_c(\boldsymbol{\psi}) - \mathbf{B}_m)\mathbf{r} = \mathbf{A}_m\boldsymbol{\epsilon} + \Psi(\boldsymbol{\psi} - \boldsymbol{\psi}_0)$ for some matrix Ψ and under the strong assumption that there exists a $\boldsymbol{\psi}_0$ such that $\mathbf{A}_c(\boldsymbol{\psi}_0) = \mathbf{A}_m$ and $\mathbf{B}_c(\boldsymbol{\psi}_0) = \mathbf{B}_m$. Now introduce the Lyapunov function $V = \frac{1}{2}(\gamma\boldsymbol{\epsilon}^\top\mathbf{P}\boldsymbol{\epsilon} + (\boldsymbol{\psi} - \boldsymbol{\psi}_0)^\top(\boldsymbol{\psi} - \boldsymbol{\psi}_0))$ for some positive definite matrix $\mathbf{P}$. The associated time derivative is $\dot{V} = \frac{1}{2}\gamma\boldsymbol{\epsilon}^\top\mathbf{Q}\boldsymbol{\epsilon} + (\boldsymbol{\psi} - \boldsymbol{\psi}_0)^\top\big(\dot{\boldsymbol{\psi}} + \gamma\Psi^\top\mathbf{P}\boldsymbol{\epsilon}\big)$ where $\mathbf{Q}$ is positive definite and satisfies $\mathbf{A}_m^\top\mathbf{P} + \mathbf{P}\mathbf{A}_m = -\mathbf{Q}$ (such a matrix will always exist if $\mathbf{A}_m$ is stable). Therefore, choosing the update law $\dot{\boldsymbol{\psi}} = -\gamma\Psi^\top\mathbf{P}\boldsymbol{\epsilon}$ leads to a negative semidefinite $\dot{V}$ and using Barbalat's lemma, exponential decay to zero can be proven. In practise it is usually impossible to implement the actual update law because one is unlikely to have access to the matrix Ψ and therefore will have to approximate it in some manner.

For MRAC schemes in general, a reasonable degree of expert knowledge regarding the form of the system dynamics must be available in order to define a good adaptive law for the policy parameters.

Iterative Feedback Tuning

Iterative Feedback Tuning (IFT) is a method of tuning the parameters of a policy in order to minimise some cost functional without knowledge of the plant dynamics. It was the contribution of Hjalmarsson *et al.* (1994) to show that an unbiased estimate of the gradient of the cost function with respect to policy parameters can be computed from signals obtained from closed loop experiments with the current policy operating on the actual system. These ideas were further developed and presented in the cornerstone paper of Hjalmarsson *et al.* (1998). On the other hand, in MRAC the true closed loop plant is replaced by the reference model in the gradient computation step. The minimisation of the cost function is then achieved using a Gauss-Newton based gradient descent scheme.

For the tuning of a one-degree of freedom controller, each gradient step consists of two batch experiments. The first is simply the collection of data under normal operating conditions while the second involves feeding back the output of the first experiment as the reference input to be tracked. This method has gained some acceptance in industry for the tuning of PID controllers due to its simplicity. The basic theory can be found in the survey paper of Hjalmarsson (2002).

2.2.3 Self-Tuning Regulators

Traditional Approach

Self-Tuning Regulators (STRs) maintain parameter approximations for a model of the system dynamics (usually a linear transfer function). This model is then used as part of an online control design scheme, for example pole placement or LQG. In general STRs are indirect adaptive methods since system parameters are learned explicitly. This framework is shown in Fig. 2.2 where $\mathbf{w}$ are the free parameters in the model and $\mathcal{D}$ is the observed data set. However if an algebraic mapping straight to the policy parameters can be learned then it can be implemented as a direct method and can in fact be viewed as a form of MRAC.

The system model that is traditionally assumed in the STR framework is a Single Input Single Output (SISO) AutoRegressive with eXogenous inputs (ARX) model. This is defined as

$$y_k = \boldsymbol{\phi}_{k-1}^\top \mathbf{w} + e_k$$

where y_k is the output, $e \sim \mathcal{N}(0, \sigma_e^2)$ is white noise, $\boldsymbol{\phi}_{k-1} = [y_{k-1:k-n}; u_{k-m:k-n}; \hat{e}_{k-1:k-n}]$ is a vector of lagged outputs, inputs and estimated noise signals and $\mathbf{w}$ is the vector of unknown parameters. Since these equations are linear in the unknown parameters $\mathbf{w}$, we can employ standard Kalman filtering techniques to maintain a Gaussian distribution over this vector space $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{w}, \boldsymbol{\Sigma}_\mathbf{w})$. The equations used in this case would be

$$\begin{aligned}\boldsymbol{\mu}_{\mathbf{w},k} &= \boldsymbol{\mu}_{\mathbf{w},k-1} + \mathbf{K}_k\Big(y_k - \boldsymbol{\phi}_{k-1}^\top \boldsymbol{\mu}_{\mathbf{w},k-1}\Big)\\ \boldsymbol{\Sigma}_{\mathbf{w},k} &= \Big(\mathbf{I} - \mathbf{K}_k \boldsymbol{\phi}_{k-1}^\top\Big)\boldsymbol{\Sigma}_{\mathbf{w},k-1}\end{aligned}$$

where $\mathbf{K}_k = \boldsymbol{\Sigma}_{\mathbf{w},k-1}\boldsymbol{\phi}_{k-1}^\top\big(\sigma_e^2\mathbf{I} + \boldsymbol{\phi}_{k-1}^\top\boldsymbol{\Sigma}_{\mathbf{w},k-1}\boldsymbol{\phi}_{k-1}\big)^{-1}$ and assuming that we are given prior measurements $\boldsymbol{\phi}_0$ and an initial distriubtion $\mathbf{w}_0 \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{w},0}, \boldsymbol{\Sigma}_{\mathbf{w},0})$. The noise terms in $\boldsymbol{\phi}_{k-1}$ cannot be measured directly in general, therefore they are replaced with estimates based on the current model.

It is important to note that although an estimate of the full distribution over $\mathbf{w}$ is maintained, in general only the mean prediction is used. This is due to the *certainty equivalence principle* which assumes that the optimal control law is only dependent on the mean prediction of the parameter estimate. Tse & Bar-Shalom (1975) prove that this principle holds for any known linear system with additive noise of known but arbitrary statistics, a measured output that is a nonlinear function of the state and noise (also with known but arbitrary statistics) and a quadratic cost function of state and input. However, it does not hold in the general case. It is

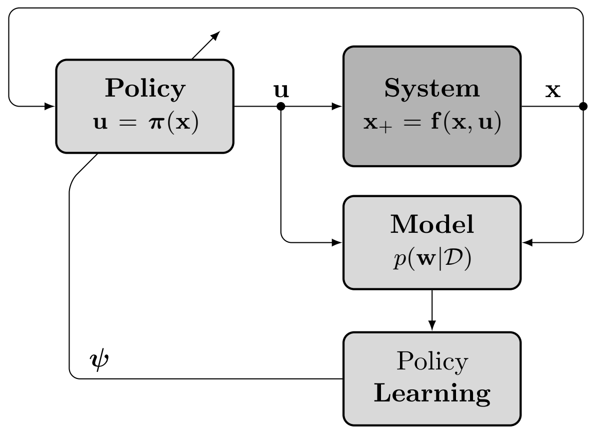

Policy
$\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$
$\mathbf{u}$
System
$\mathbf{x}_+ = \mathbf{f}(\mathbf{x}, \mathbf{u})$
$\mathbf{x}$
Model
$p(\mathbf{w}|\mathcal{D})$
$\boldsymbol{\psi}$
Policy
Learning

Figure 2.2: Layout of an *indirect* self-tuning regulator for a system described by the equations $\mathbf{x}_+ = \mathbf{f}(\mathbf{x}, \mathbf{u})$ parameterised by the unknown or time-varying parameters $\mathbf{w}$. First an estimate of the unknown parameters is made $\mathbb{E}[\mathbf{w}]$, then the estimated system model is used under some control design scheme to determine policy parameters $\boldsymbol{\psi}$.

this issue that is tackled by the framework of dual control which is outlined in Sec. 2.2.3.

The traditional STR scheme is then to perform an online policy design following some standard methodology using the estimated model of the plant as though it was the real plant. Åström & Wittenmark (1994) use either pole-placement or LQG methods for design which recursively solve a Diophantine equation or Riccati equation respectively. This would be an *indirect* STR. A *direct* STR can be obtained by combining the plant parameter estimation stage and the controller design to a problem where the controller parameters can be estimated directly.

Adaptive Model Predictive Control

The framework is of course much broader than these traditional methods. We could in fact consider a Model Predictive Control (MPC) scheme, in which the internal model is obtained through online estimation, to come under this framework. MPC uses an internal model of a system to perform an online optimisation over some finite predictive horizon and the space of possible control actions in order to determine the optimal sequence of actions to apply. It then applies the first (or first few) actions in this sequence to the real system and repeats the procedure at the subsequent timestep. One of the advantages of this method is that constraints on states and inputs can be handled quite naturally. For an excellent introductory resource to this control framework the reader is referred to Maciejowski (2002).

Dual Control

One issue with the STR scheme, and many other adaptive control methods, is how to ensure that the input signal under feedback is rich enough to excite the system in order to obtain good parameter estimates. In other words, is the input "persistently exciting"? The desire to handle this problem in an optimal manner led to the notion of dual control.

Dual control was first introduced in the seminal papers by Fel'dbaum (1960–61) and presented a theoretical framework for the optimal control of unknown, nonlinear stochastic systems. A helpful introduction can be found in Wittenmark (2000). The name dual control was given due to the fact that the resulting optimal control law sought to simultaneously control the system to the desired setpoint while introducing random probing inputs in order to improve parameter estimates. It was shown that the resulting control law was a function of the full distribution over the estimated values of model parameters $p(\mathbf{w})$, not just the mean prediction $\mathbb{E}[\mathbf{w}]$. The information contained in this full distribution is termed the *hyperstate*. As previously discussed, this is in contrast to the vast majority of indirect STR algorithms.

In general the optimal dual solution is intractable and heuristic methods are employed to give a policy *dual characteristics*. A survey of some of these approximation methods can be found in Wittenmark (1995) and Unbehauen (2000). Tse & Bar-Shalom (1972) provide an early example of a dual control heuristic showing significant improvement over the associated certainty equivalent scheme. Another example is that of Fabri & Kadirkamanathan (1998), where an extra term is added in the cost function of a minimum-variance controller to induce dual action. Simpkins *et al.* (2008) use a value function approximation scheme to solve the Hamilton-Jacobi-Bellman (HJB) equation for the dual control problem for a nonlinear system in which only a subset of the state variables were measured. The system dynamics are augmented with the dynamics of a Kalman-Bucy filter to yield a fully observable system. Then the value function is approximated using available training data and the optimal policy can be derived analytically, which is possible only because the considered system is affine in the inputs and quadratic in the input cost.

2.2.4 Extremum Seeking Control

Background

In some situations, the general setup given in Fig. 1.1(a) can be decomposed into the form given in Fig. 2.3. In this case, the entire closed loop system has been reduced to a SISO static map g (a form of cost function which is a function of the policy parameter ψ directly) with no dynamics. In general, extremum seeking is not restricted to this assumption and can handle both dynamic maps and Multiple Input Multiple Output (MIMO) systems, but it allows an intuitive understanding of the method to be obtained. The aim is to find the optimal policy parameter ψ^* to yield the optimal function value g^*. An illustrative example is an anti-lock braking system in which there is a mapping from the desired slip of the wheel to the amount of friction generated. This mapping has some maximum value (in between no slip at all and skidding) which would be beneficial to operate at if the objective was to stop the vehicle as fast as possible. However the position of this maximum varies with changes in road condition, whether it is wet, dry, icy etc. The control methodology employs the use of an excitation or "dither" signal to provide an implicit search for the extremum. A recent real world example by

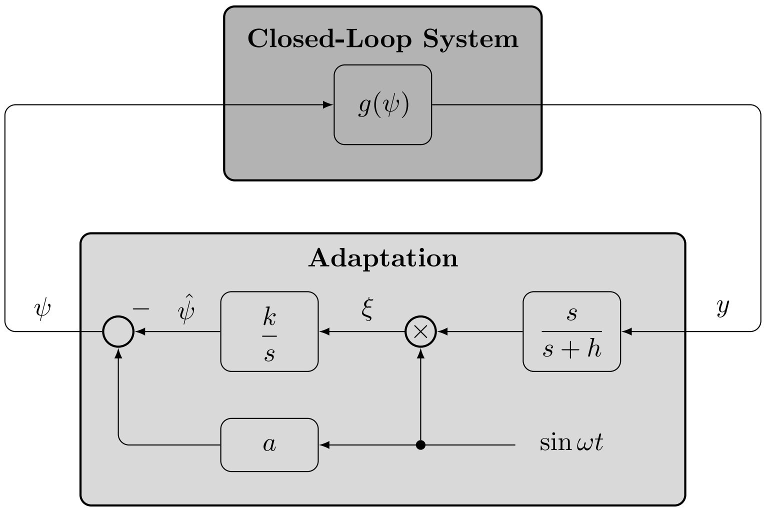


Figure 2.3: Layout of the extremum-seeking scheme with a sinusoidal dither signal of frequency ω for an unknown static map $g(\psi)$ with optimal value g^* and associated setpoint ψ^*.

Bastin *et al.* (2009) is of its application in bioprocesses.

Ariyur & Krstić (2003) claim that extremum-seeking was the original method of adaptive control, appearing as far back as the 1920s, developed mainly in the USSR during the 1940s and was popular in the 1950s and 1960s long before the theoretical breakthroughs in the adaptive control of linear systems in the 1980s. This popularity died down for a while, partly due to the difficulty in implementing optimizing controllers, however developments in computing power have led to a renewed interest, including the semi-plenary lecture given by Nešić (2009). There are two main approaches to extremum-seeking. One is the adaptive control method which makes use of a certain excitation or dither signal (often a sinusoid) to generate the desired suboptimal behaviour required to search for the extremum point. The other way is to use nonlinear programming techniques for numerical optimisation using probing inputs to generate an online approximation of the objective function gradient. By way of example, the adaptive control approach for an unknown static map will now be considered.

Illustrative Example

We will now gain some intuition as to how the scheme given in Fig. 2.3 actually works. This example is outlined in Ariyur & Krstić (2003). Assuming that the mapping g is smooth then when the input ψ is close to ψ^* a locally valid approximation would be given by the quadratic

$$g(\psi) = g^* + \frac{g''}{2}(\psi - \psi^*)^2 \tag{2.1}$$

where $g'' := \mathrm{d}^2 g/\mathrm{d}\psi^2 > 0$ (if $g'' < 0$ simply replace k ($k > 0$) in Fig. 2.3 with $-k$), g^* and ψ^* are unknown constants. The value g^* is a local optimum of the general map. It is the dither signal $a \sin \omega t$ fed into the plant that helps get a measure of gradient information as will be illustrated in the following simple analysis. First observe that $\hat{\psi}$ is the estimate of the unknown optimal input ψ^* and define the estimation error $\tilde{\psi} = \psi^* - \hat{\psi}$. When substituted into Eq. (2.1) the following is obtained

$$\begin{aligned} y &= g^* + \frac{g''}{2}(\tilde{\psi} - a\sin\omega t)^2 \\ &= g^* + \frac{a^2 g''}{4} + \frac{a^2 g''}{2}\tilde{\psi}^2 - a g''\tilde{\psi}\sin\omega t - \frac{a^2 g''}{4}\cos 2\omega t \end{aligned}$$

using basic trigonometric identities. The washout (high pass) filter $s/(s+h)$, with ‘corner frequency’ h, then serves to remove the constant terms. After this the signal is demodulated through multiplication with $\sin\omega t$ as follows

$$\begin{aligned} \xi &\approx \frac{a^2 g''}{2}\tilde{\psi}^2\sin\omega t - a g''\tilde{\psi}\sin^2\omega t - \frac{a^2 g''}{4}\cos 2\omega t\sin\omega t \\ &\approx -\frac{a^2 g''}{2}\tilde{\psi} \quad + \quad \underbrace{\frac{a^2 g''}{2}\tilde{\psi}\cos 2\omega t + \frac{a^2 g''}{8}(\sin\omega t - \sin 3\omega t)}_{\text{attenuated by the integrator } k/s} \quad + \quad \underbrace{\frac{g''}{2}\tilde{\psi}^2\sin\omega t}_{\text{negligible for local analysis}} \end{aligned} \tag{2.2}$$

again using basic trigonometric identities. Due to Lyapunov’s Indirect Method the quadratic term in $\tilde{\psi}$ can be neglected as only local analysis is of interest. Assuming that ω is a relatively high frequency, the sinusoidal terms in Eq. (2.2) would be significantly attenuated by the integrator. This leaves the following simple relationship

$$\hat{\psi} \approx -\frac{k}{s}\left(-\frac{a g''}{2}\tilde{\psi}\right)$$

Finally, since ψ^* is constant then $\dot{\tilde{\psi}} = -\dot{\hat{\psi}}$ so

$$\dot{\tilde{\psi}} \approx -\frac{k a g''}{2}\tilde{\psi}$$

Now since $kg'' > 0$, this is a stable system and it can be concluded that $\tilde{\psi} \to 0$, in other words the estimate $\hat{\psi}$ will converge to the the actual extremum ψ^*.

Discrete-time extremum seeking control is exactly analogous to the continuous-time as shown in Fig. 2.3. We can apply the Tustin transform $s \leftarrow 2(z-1)/\delta_t(z+1)$, where δ_t is the sampling period, to translate the integrator and the high pass filter to discrete equivalents. However, the stability proofs are quite different (see Choi *et al.* (2002)).

2.3 Reinforcement Learning

2.3.1 Introduction

Reinforcement learning is a field that was born, in the form that can be seen today, during the early 1980s. It was formed by the combination of two main threads of independent research. One thread was research carried out in optimal control and solutions involving the use of dynamic programming based methods. Although this did not involve learning as such, the algorithms gradually reach the correct solution through successive approximations so were considered among the other reinforcement learning methods. The other was research in learning by trial and error which started in the psychology of animal learning and led to Monte-Carlo methods for learning. Concepts from each of these threads were then combined to form a what is known as reinforcement learning today, a cornerstone of which were the Temporal Difference (TD) methods. One of the most famous successes of early TD methods came in the development of a world-class computer backgammon player in Tesauro (1992, 1995). Good reviews of the early literature in the field of reinforcement learning and the basic concepts can be found in Sutton & Barto (1998), Kaelbling *et al.* (1996) and Bertsekas & Tsitsiklis (1996) for a more in depth analysis.

2.3.2 Markov Decision Process Framework

Reinforcement learning algorithms assume the framework of a Markov Decision Process (MDP). MDPs are discrete-time dynamical systems where the next state is dependent only on the current state $\mathbf{x}_k \in \mathbb{X} \subset \mathbb{R}^n$ and action $\mathbf{u}_k \in \mathbb{U} \subset \mathbb{R}^m$ and governed by the mapping $\mathbf{f} : \mathbb{X} \times \mathbb{U} \to \mathbb{X}$. This property is known as the Markov property. Further, define $\mathbf{z} := [\mathbf{x}; \mathbf{u}]$ as the combined state-action vector. The system is in some initial state drawn from the distribution $\mathbf{x}_0 \sim p(\mathbf{x}_0)$ and at any state $\mathbf{x}_k$ the agent will draw an action from the stochastic policy $\pi(\mathbf{u}_k|\mathbf{x}_k)$, again parameterised by $\boldsymbol{\psi}$. Note that the policy in this framework is considered as giving a distribution over actions rather than a deterministic mapping. However, usually once the learning phase is complete, the mean of this stochastic policy could be returned as a deterministic control policy.

A trajectory through the state-action space of an MDP is denoted $\boldsymbol{\tau} = \{\mathbf{x}_{0:H}, \mathbf{u}_{0:H}\} \in \mathcal{T}$ where H is the task horizon. The probability of a trajectory being generated by a given policy is $p(\boldsymbol{\tau}|\pi) = \pi(\mathbf{u}_0|\mathbf{x}_0)p_0(\mathbf{x}_0)\prod_{k=1}^{H}\pi(\mathbf{u}_k|\mathbf{x}_k)$ according to the deterministic unknown dynamics function $\mathbf{f}$ (stochastic dynamics may also be considered). The cost $\rho(\boldsymbol{\tau})$ for a given trajectory $\boldsymbol{\tau}$ is defined as

$$\rho(\boldsymbol{\tau}) = \sum_{k=0}^{H} a_k c(\mathbf{z}_k)$$

where a_k is some time dependent weighting factor. Common examples are the *discounted return* $a_k = \gamma^k$ where $\gamma \in (0, 1]$ and the *average return* $a_k = 1/(H + 1)$. The general goal of reinforcement learning is to find a policy such that the expected cost

$$J(\pi) = \mathbb{E}_{\boldsymbol{\tau}}\left[\rho(\boldsymbol{\tau})|\pi\right] = \int_{\mathcal{T}} \rho(\boldsymbol{\tau})P(\boldsymbol{\tau}|\pi)\mathrm{d}\boldsymbol{\tau}$$

is minimised. The policy which yields the minimum expected cost is denoted $\pi^* := \arg\min_\pi J(\pi)$.

We note that significant research is devoted to *minimum regret* reinforcement learning. The notion of regret is to penalise the cost incurred by the algorithm during learning as well as the cost incurred by the final policy. Jointly addressing both of these issues is beyond the scope of this thesis therefore such algorithms will not be pursued further.

Important concepts in reinforcement learning and optimal control are that of the value function $V^\pi(\mathbf{x}, k)$ and action-value function $Q^\pi(\mathbf{z}, k)$. The value function is the expected return if policy π is followed from state $\mathbf{x}_k$. Similarly, the action-value function gives the expected return of taking action $\mathbf{u}_k$ in state $\mathbf{x}_k$ then choosing actions according to the policy π thereafter. In mathematical terms

$$V^\pi(\mathbf{x}, k) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{i=k}^{H} a_i c(\mathbf{z}_i)\middle|\mathbf{x}_k = \mathbf{x}, \pi\right]$$
$$Q^\pi(\mathbf{z}, k) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{i=k}^{H} a_i c(\mathbf{z}_i)\middle|\mathbf{z}_k = \mathbf{z}, \pi\right]$$

If we restrict our attention to problems with time-invariant dynamics then in the infinite horizon case $H \to \infty$, these functions become time-invariant so are denoted $V^\pi(\mathbf{x})$ and $Q^\pi(\mathbf{z})$ respectively. Note that the expected return can be written in terms of the value function by averaging over all possible start states $\mathbf{x}_0$, $J(\pi) = \int_{\mathbb{X}} V^\pi(\mathbf{x}_0)p(\mathbf{x}_0)\mathrm{d}\mathbf{x}_0$. The optimal value function is given by

$$V^*(\mathbf{x}) = \min_\pi[V^\pi(\mathbf{x})],$$

where an optimal policy is the argument to the minimal solution $\pi^*(\mathbf{x}) = \arg\min_\pi V^\pi(\mathbf{x})$. The action-value function and value function are linked through the relationship $V^\pi(\mathbf{x}) = \mathbb{E}_{\mathbf{u}}[Q^\pi(\mathbf{z})|\mathbf{x}, \pi]$. Similarly, the optimal action-value function is related to the optimal value function through the relationship $V^*(\mathbf{x}) = \min_{\mathbf{u}\in\mathbb{U}}[Q^*(\mathbf{z})]$.

2.3.3 Dynamic Programming

The dynamic programming paradigm of Bellman (1957) is the framework that most reinforcement learning algorithms use to find an optimal policy. In order to introduce the concepts we shall restrict our attention to the infinite horizon ($H \to \infty$), discounted return ($a_k = \gamma^k$) case. It can be observed that the value function satisfies the Bellman (fixed-point) equation

$$V^\pi(\mathbf{x}) = \mathbb{E}_{\mathbf{x}_+,\mathbf{u}}\Big[c(\mathbf{z}) + \gamma V^\pi(\mathbf{x}_+)\big|\mathbf{x}, \pi\Big]$$

where $\mathbf{x}_+ = \mathbf{f}(\mathbf{x}, \mathbf{u})$. Note that the uncertainty in $\mathbf{x}_+$ comes from the stochastic policy π. The optimal value function satisfies the Bellman *optimality* equation (otherwise known as the discrete-time Hamilton-Jacobi-Bellman (HJB) equation)

$$V^*(\mathbf{x}) = \min_{\mathbf{u}\in\mathbb{U}} \mathbb{E}_{\mathbf{x}_+}\Big[c(\mathbf{z}) + \gamma V^*(\mathbf{x}_+)\big|\mathbf{z}\Big].$$

Given these equations we can now define two important forward in time algorithms: Policy Iteration and Value Iteration. Each one iterates between *evaluation* of the value function of a given policy and *improvement* of the policy but making it "greedy" with respect to the current value function. These procedures are described in Algorithms 2.1 and 2.2.

Algorithm 2.1 (Policy Iteration). *Initialise procedure with some policy π_0. Iterate the following steps until convergence:*

Evaluation: *Determine the value of policy π_j by finding the value function V_{j+1} that satisfies $V_{j+1}(\mathbf{x}) = \mathbb{E}_{\mathbf{x}_+,\mathbf{u}}\big[c(\mathbf{z}) + \gamma V_{j+1}(\mathbf{x}_+)|\mathbf{x}, \pi_j\big]$.*

Improvement: *Determine an improved policy π_{j+1} by being greedy with respect to the value function V_{j+1} using $\pi_{j+1}(\mathbf{x}) = \arg\min_{\mathbf{u}\in\mathbb{U}} \mathbb{E}_{\mathbf{x}_+}\big[c(\mathbf{z}) + \gamma V_{j+1}(\mathbf{x}_+)|\mathbf{z}\big]$.*

Algorithm 2.2 (Value Iteration). *Replace the evaluation step in Policy Iteration with a truncation by setting $V_{j+1}(\mathbf{x}) = \mathbb{E}_{\mathbf{x}_+,\mathbf{u}}\big[c(\mathbf{z}) + \gamma V_j(\mathbf{x}_+)|\mathbf{x}, \pi_j\big]$ where V_{j+1} has been replaced by V_j on the right hand side.*

Both Policy Iteration and Value Iteration can be viewed under the Generalised Policy Iteration scheme of Sutton & Barto (1998) where evaluation and improvement steps are interleaved. In fact, all value function based schemes can be viewed under this framework.

Now, even if these algorithms were carried out using only data from interaction with the system, some a priori knowledge of the system dynamics is required in the improvement step, which is common to both algorithms. In order to find the minimal solution, the derivative with respect to the input $\mathbf{u}$ is required which leads to the necessity of knowing the derivative $\partial\mathbf{f}(\mathbf{x}, \mathbf{u})/\partial\mathbf{u}$ (the "$\mathbf{B}$" matrix if the system is linear). This problem is alleviated in reinforcement learning algorithms through the use of the action-value function in place of the value function. The action-value function satisfies

$$Q^{\pi}(\mathbf{z}) = \mathbb{E}_{\mathbf{z}_+}\Big[c(\mathbf{z}) + \gamma Q^{\pi}(\mathbf{z}_+)\Big|\mathbf{z}, \pi\Big]. \tag{2.3}$$

The optimal action-value function satisfies

$$Q^{*}(\mathbf{z}) = \mathbb{E}_{\mathbf{x}_+}\Big[c(\mathbf{z}) + \gamma \min_{\mathbf{u}_+\in\mathbb{U}} Q^{*}(\mathbf{z}_+)\Big|\mathbf{z}\Big].$$

We shall now discuss how these can be used as the basis of practical algorithms.

2.3.4 Temporal Differences

Bootstrapping Methods

Bootstrapping methods are based on estimating the Temporal Difference (TD)-error which will be defined below. The term “bootstrapping” pertains to the idea that estimates of parameters are based on estimates of other parameters. These methods are ways of evaluating or estimating the value function of a given policy V^π (not considered further, since knowledge of the system dynamics is required to find a policy), the action-value function Q^π (the foundation of the SARSA algorithm) or the optimal action-value function Q^* directly (the foundation of $\mathcal{Q}$-learning). They have enjoyed much empirical success in the discrete state-space domain, including learning how to play backgammon Tesauro (1992, 1995) and chess Baxter *et al.* (2001). These methods utilise the TD-error, or Bellman error, as part of a stochastic gradient descent algorithm. In most applications it is necessary to maintain a parametric approximation of these functions, denoted by $\hat{V}_{\boldsymbol{\theta}}$ and $\hat{Q}_{\boldsymbol{\theta}}$ where $\boldsymbol{\theta}$ is the vector of free parameters. Borrowing from the survey paper of Geist & Pietquin (2010a) the following cost functions are minimised

$$J_{Q^\pi}(\boldsymbol{\theta}) = \parallel Q^\pi - \hat{Q}_{\boldsymbol{\theta}} \parallel^2 \tag{2.4}$$

$$J_{Q^*}(\boldsymbol{\theta}) = \parallel Q^* - \hat{Q}_{\boldsymbol{\theta}} \parallel^2 \tag{2.5}$$

for SARSA and $\mathcal{Q}$-learning respectively. Finding the optimal setting for the parameters $\boldsymbol{\theta}$ is achieved using a stochastic gradient descent where the parameters are adjusted according to an approximation of the gradient of the cost function. Given an observation of a transition $(\mathbf{x}, \mathbf{u}, c, \mathbf{x}_+, \mathbf{u}_+)$ for SARSA or $(\mathbf{x}, \mathbf{u}, c, \mathbf{x}_+)$ for $\mathcal{Q}$-learning the update law is given by

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1} + \alpha_i \Big(\nabla_{\boldsymbol{\theta}_{i-1}} \hat{Q}_{\boldsymbol{\theta}}(\mathbf{z})\Big)\Delta_i$$

where α_i is a learning rate that satisfies the classical stochastic approximation criterion $\sum_{i=1}^{\infty} \alpha_i = \infty, \sum_{i=1}^{\infty} \alpha_i^2 < \infty$ and the TD-error Δ_i is given by

$$\Delta_i = \Big(c(\mathbf{z}) + \gamma \hat{Q}_{\boldsymbol{\theta}_{i-1}}(\mathbf{z}_+)\Big) - \hat{Q}_{\boldsymbol{\theta}_{i-1}}(\mathbf{z})$$

$$\Delta_i = \Big(c(\mathbf{z}) + \gamma \min_{\mathbf{u}_+ \in \mathbb{U}} \hat{Q}_{\boldsymbol{\theta}_{i-1}}(\mathbf{z}_+)\Big) - \hat{Q}_{\boldsymbol{\theta}_{i-1}}(\mathbf{z})$$

for SARSA and $\mathcal{Q}$-learning respectively. It should be noted that in approximating the action-value function (as in SARSA) the agent must be following the policy π in order to converge. This is known as an *on-policy* method. A convergence proof of SARSA with linear function approximation in finite state-action space can be found in Tsitsiklis & Van Roy (1997). $\mathcal{Q}$-learning was introduced for the discrete space, look-up table formulation, in the thesis by Watkins (1989) with convergence proofs following in Watkins & Dayan (1992). It is an *off-policy* method since it can follow an arbitrary policy (provided it explores sufficiently in the state-action space) but will converge to the optimal action-value function.

An important extension of the standard $\mathcal{Q}$-learning algorithm: fitted $\mathcal{Q}$-iteration, was presented by Ernst *et al.* (2005) and seeks to combine the $\mathcal{Q}$-learning principle with supervised learning methods to form a batch algorithm. A comparison of this approach with MPC can be found in Ernst *et al.* (2009).

Residual Methods

As opposed to minimising the cost functions given by (2.4) and (2.5) one could minimise the error obtained by forcing the function approximation $\hat{Q}_{\boldsymbol{\theta}}$ to satisfy the fixed point equation (2.3)

$$J(\boldsymbol{\theta}) = \| \hat{Q}_{\boldsymbol{\theta}} - (c + \gamma \hat{Q}_{\boldsymbol{\theta}}) \|^2 .$$

This method does not lend itself to the stochastic gradient descent method outlined in the previous section because a penalty term arises which acts to favour smooth functions and hence gives a biased estimate of the optimal solution. However, least squares approaches to this minimisation have worked well. For example Gaussian Process Temporal Differences (GPTD) of Engel *et al.* (2003, 2005) or Kalman Temporal Differences (KTD) of Geist *et al.* (2009) (see Geist & Pietquin (2010b) for a full derivation).

Practical Issues

There are a few notable problems with the algorithms outlined in this section. First, there is the problem of reconstructing a useable or parameterised policy from the action-value function once it has been learned (a notable exception is when the system is affine and the cost is quadratic in the control actions). However, the most important problem is that these methods do not scale to continuous state-action spaces. The reason for this is that the algorithms are not able to generalise what they have learned in one area of the state-action space to another, they have no notion that states that are “close” to each other will respond in a similar way. Neglecting to include any such prior information means that all states must be visited individually in order to build a policy. This is impossible in continuous states.

One class of reinforcement learning algorithms that has been able to overcome these issues is policy-gradient algorithms and are the subject of the following section.

2.3.5 Policy-Gradient Algorithms

Background

The aim of policy-gradient algorithms is to directly adjust policy parameters $\boldsymbol{\psi}$ by utilising an estimate of the gradient of the expected return with respect to the policy parameters $\nabla J(\boldsymbol{\psi})$.

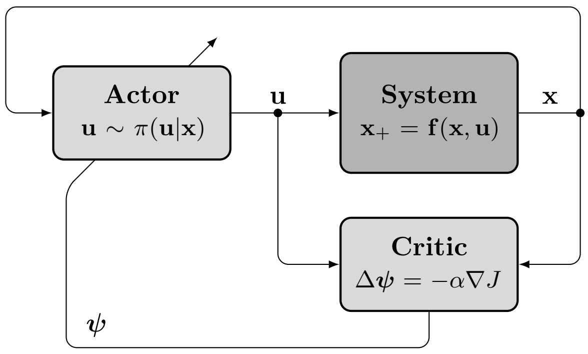

Actor
$\mathbf{u} \sim \pi(\mathbf{u}|\mathbf{x})$
$\mathbf{u}$
System
$\mathbf{x}_+ = \mathbf{f}(\mathbf{x},\mathbf{u})$
$\mathbf{x}$
Critic
$\Delta\boldsymbol{\psi} = -\alpha\nabla J$
$\boldsymbol{\psi}$

Figure 2.4: Layout of the actor-critic policy gradient scheme where the policy is updated according to the rule $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} + \Delta\boldsymbol{\psi}$. Note that the actor applies the policy and the critic updates the parameters.

Note that $\nabla = \nabla_{\boldsymbol{\psi}}$ unless otherwise specified. Such algorithms are often referred to as *Actor-Critic Methods* where the *critic* maintains the estimate of the action-value function which in turn informs the *actor* how to update its parameters. The actor-critic layout is shown in Fig. 2.4. Note that from an adaptive control perspective, this is a *direct* method since it provides an update rule for the policy parameters directly. The policy parameters of the actor are updated in the direction of the negative gradient provided by the critic according to the adjustment law

$$\boldsymbol{\psi}_{i+1} = \boldsymbol{\psi}_i - \alpha_i \nabla J(\boldsymbol{\psi})|_{\boldsymbol{\psi}=\boldsymbol{\psi}_i}$$

where the learning rate α_i satisfies $\sum_{i=0}^{\infty} a_i = \infty$ and $\sum_{i=0}^{\infty} a_i^2 < \infty$. Since the policy π is now fully characterised by its free parameters $\boldsymbol{\psi}$ the expected return $J(\pi)$ can be written $J(\boldsymbol{\psi})$.

The problem here is how do we obtain a good estimate of the gradient? Two prominent approaches can be found in the literature: finite-difference and likelihood ratio methods. For an overview of these methods the reader is directed to the article by Peters & Schaal (2008). Finite-difference methods can be dated back as far as the 1950s, originating in the stochastic simulation community. They involve simply applying the policy to the system a number of times using slightly perturbed parameters $\boldsymbol{\psi} + \Delta\boldsymbol{\psi}$ each time. The gradient estimation problem is then turned into a regression problem. However, determining the size of $\Delta\boldsymbol{\psi}$ requires good knowledge of the system and can be hard to tune without expert guidance. For this reason, these methods shall not be pursued further. Likelihood ratio methods are based on the *likelihood ratio trick.*

The Likelihood Ratio Trick

The gradient of the expected return can be written analytically as

$$\begin{aligned}\nabla J(\boldsymbol{\psi}) &= \int_{\mathcal{T}} \rho(\boldsymbol{\tau}) \nabla P(\boldsymbol{\tau}|\pi) \mathrm{d}\boldsymbol{\tau} \\ &= \int_{\mathcal{T}} \rho(\boldsymbol{\tau}) P(\boldsymbol{\tau}|\pi) \nabla \log P(\boldsymbol{\tau}|\pi) \mathrm{d}\boldsymbol{\tau} \\ &= \mathbb{E}_{\boldsymbol{\tau}}\Big[\big(\rho(\boldsymbol{\tau}) - b\big) \nabla \log P(\boldsymbol{\tau}|\pi)\big|\pi\Big].\end{aligned}$$

where the introduction of the arbitrary constant b is possible since $\int_{\mathcal{T}} b\nabla P(\boldsymbol{\tau}|\pi)\mathrm{d}\boldsymbol{\tau} = 0$, which is clearly true because $\int_{\mathcal{T}} P(\boldsymbol{\tau}|\pi)\mathrm{d}\boldsymbol{\tau} = 1$. This form is useful in two ways. The first is that it lends itself to sample based approximation given a set of trajectories sampled from the MDP using policy parameters $\boldsymbol{\psi}$. Secondly, the term $\nabla \log P(\boldsymbol{\tau}|\pi) = \sum_{k=0}^{H} \nabla \log \pi(\mathbf{u}_k|\mathbf{x}_k)$ requires no knowledge of the system dynamics to calculate. This is the basis of the episodic REINFORCE algorithm of Williams (1992). Although b has no effect on the mean of the gradient estimate it can have a huge effect on the variance of the estimate and can be chosen so as to minimise this variance. The optimal baseline was derived by Peters & Schaal (2008). Improved algorithms were developed by Sutton *et al.* (2000) and Marbach & Tsitsiklis (2001) with the Policy Gradient Theorem and Baxter & Bartlett (2001) with GPOMDP. They make the intuitive observation that future actions are independent of past rewards. Peters & Schaal (2008) show from a trajectory based perspective that these algorithms are in fact equivalent. We shall now look further at the Policy Gradient Theorem.

The Policy Gradient Theorem

Consider the infinite horizon, normalised case, in which $\lim_{H\to\infty} \frac{1}{H}\sum_{k=0}^{H} a_k = 1$. It was noted by Sutton *et al.* (2000) that the expected return can be written in state-space form

$$J(\boldsymbol{\psi}) = \int_{\mathbb{X}} d^{\pi}(\mathbf{x}) \int_{\mathbb{U}} \pi(\mathbf{u}|\mathbf{x}) c(\mathbf{x},\mathbf{u})\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{x}$$

where $d^{\pi}(\mathbf{x}) = \sum_{k=0}^{\infty} a_k p(\mathbf{x}_k = \mathbf{x})$ is the weighted state distribution which is assumed to be independent of $\mathbf{x}_0$ for all policies. Note that the control policy must be chosen carefully in order to make this a reasonable assumption. Setting the problem up in this way led to Theorem 2.1.

Theorem 2.1 (Policy Gradient Theorem). *For any MDP and arbitrary baseline $b^{\pi}(\mathbf{x})$ the following relationship holds*

$$\nabla J(\boldsymbol{\psi}) = \int_{\mathbb{X}} d^{\pi}(\mathbf{x}) \int_{\mathbb{U}} \nabla\pi(\mathbf{x}|\mathbf{u})\Big(Q^{\pi}(\mathbf{x},\mathbf{u}) - b^{\pi}(\mathbf{x})\Big)\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{x}$$

Proof. The interested reader is directed to Theorem 1 in Marbach & Tsitsiklis (2001) or Proposition 1 in Sutton *et al.* (2000) for the original derivations. ∎

It was further shown by Marbach & Tsitsiklis (2001) and Sutton *et al.* (2000) that the term $Q^{\pi}(\mathbf{z}) - b^{\pi}(\mathbf{x})$ can be replaced by a *compatible function approximation* $f^{\pi}_{\boldsymbol{\theta}}(\mathbf{z}) = \nabla \log \pi(\mathbf{u}|\mathbf{x})^{\top}\boldsymbol{\theta}$ which does not affect the unbiasedness of the gradient estimate and is irrespective of the choice of baseline. Substituting this into the gradient equation from Theorem 2.1 yields

$$\nabla J(\boldsymbol{\psi}) = \int_{\mathbb{X}} d^{\pi}(\mathbf{x})\hat{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{x})\mathrm{d}\mathbf{x}\, \boldsymbol{\theta} = \mathbf{G}_{\boldsymbol{\theta}}\boldsymbol{\theta}$$

where the integral $\hat{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{x}) = \int_{\mathbb{U}} \pi(\mathbf{u}|\mathbf{x}) \nabla \log \pi(\mathbf{u}|\mathbf{x}) \nabla \log \pi(\mathbf{u}|\mathbf{x})^{\top} \mathrm{d}\mathbf{u}$ can be evaluated analytically since $\pi(\mathbf{u}|\mathbf{x})$ is specified by the user. However, evaluating $\mathbf{G}_{\boldsymbol{\theta}}$ is still expensive since the stationary distribution $d^{\pi}(\mathbf{x})$ is unknown.

A major issue with the algorithms discussed so far is that they tend to reduce exploration noise in the policy too quickly and therefore converge slowly on the optimal solution. Kakade (2002) showed that this problem is a product of using the standard gradient metric but can be overcome through the use of the *natural policy gradient*.

The Natural Policy Gradient

Rather than following the steepest direction in parameter space the natural gradient follows the steepest descent direction with respect to the Fisher information matrix $\mathbf{F}_{\boldsymbol{\theta}}$

$$\tilde{\nabla} J(\boldsymbol{\psi}) = \mathbf{F}_{\boldsymbol{\theta}}^{-1} \nabla J(\boldsymbol{\psi})$$

The method came out of the supervised learning community where Amari (1998) showed that the natural gradient is superior to the standard gradient method since it exploits the properties of Riemannian manifolds. It was first utilised in a reinforcement learning context by Kakade (2002) who demonstrated improved performance over the standard method. But it was the contribution of Peters *et al.* (2005) to show that the Fisher information matrix $\mathbf{F}_{\boldsymbol{\theta}}$ is in fact identically equal to the matrix $\mathbf{G}_{\boldsymbol{\theta}}$ derived in the previous section. Therefore, utilising natural policy gradients we have the very elegant result that

$$\tilde{\nabla} J(\boldsymbol{\psi}) = \mathbf{F}_{\boldsymbol{\theta}}^{-1} \mathbf{G}_{\boldsymbol{\theta}} \boldsymbol{\theta} = \boldsymbol{\theta}$$

Therefore the only remaining issue is to estimate the parameter vector $\boldsymbol{\theta}$ of the compatible function approximation. It is important and interesting to note that $f_{\boldsymbol{\theta}}^{\pi}(\mathbf{z})$ is mean zero with respect to the action distribution $\left(\int_{\mathbb{U}} \pi(\mathbf{u}|\mathbf{x}) f_{\boldsymbol{\theta}}^{\pi}(\mathbf{z}) \mathrm{d}\mathbf{u} = \boldsymbol{\theta}^{\top} \nabla \int_{\mathbb{U}} \pi(\mathbf{u}|\mathbf{x}) \mathrm{d}\mathbf{u} = 0 \right)$. Therefore it represents an *advantage function* $A^{\pi}(\mathbf{z}) = Q^{\pi}(\mathbf{z}) - V^{\pi}(\mathbf{x})$ implying that the baseline $b^{\pi}(\mathbf{x})$ has implicitly been set to the value function. However, the advantage function cannot be estimated using TD-like bootstrapping methods without also maintaining an estimate of the value function $V^{\pi}(\mathbf{x})$ which can lead to problems if the function approximation scheme is chosen inappropriately.

This problem can be overcome with a couple of mild assumptions resulting in the Episodic Natural Actor-Critic algorithm of Peters *et al.* (2005). A Bellman-like equation in terms of the advantage function can be written as

$$\sum_{k=0}^{H} a_k A^{\pi}(\mathbf{z}_k) = a_{H+1} V^{\pi}(\mathbf{x}_k) + \sum_{k=0}^{H} a_k c(\mathbf{z}_k) - V^{\pi}(\mathbf{x}_0)$$

Therefore, assuming a long task horizon $a_{H+1} \to 0$ (or $c(\mathbf{z}_H)$ is the final cost) and a narrow start state distribution so that $V^{\pi}(\mathbf{x}_0)$ can be approximated by a single scalar J_0, a simple regression problem results

$$\sum_{k=0}^{H} a_k \nabla \log \pi(\mathbf{z}_k)^\top \boldsymbol{\theta} + J_0 = \sum_{k=0}^{H} a_k c(\mathbf{z}_k)$$

with $\dim\boldsymbol{\theta} + 1$ unknowns. So for a deterministic system, only $\dim\boldsymbol{\theta} + 1$ rollouts are required to find the natural gradient using least-squares regression $[\boldsymbol{\theta}; J_0] = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Psi}$ where $\Phi[i,:] = \big[\sum_{k=0}^{H} a_k \nabla \log \pi(\mathbf{u}_{i,k}|\mathbf{x}_{i,k})^\top, 1\big]$ and $\Psi[i] = \sum_{k=0}^{H} a_k c(\mathbf{z}_{i,k})$ with $\mathbf{z}_{i,k}$ representing $\mathbf{z}_k$ taken from the i^{th} rollout.

Model-Based Policy Gradients

Now all the reinforcement learning methods we have discussed so far have been model free in the sense that a model of the actual system is not required or estimated. Furthermore, these approaches tend to require in the thousands to tens of thousands of trials with the system in order to converge to a good policy. Model-based algorithms tend to be much more efficient in terms of the amount of interaction time required to find a good policy. The reason for this is difficult to pin down precisely but is to do with the fact that a model allows the algorithm to generalise its experience in some sense, or to take advantage of some prior knowledge that says the response of the system in states close to each other will be similar.

The main reason for avoiding model-based methods is that most algorithms substitute a deterministic learned model in place of the real system in their calculations and therefore produces a biased result, a phenomenon known as model bias. One way of addressing the model-bias issue is to take account of modelling uncertainty explicitly in the design procedure through the use of a probabilistic model, see Fig. 1.1(b). For example, the PEGASUS algorithm of Ng & Jordan (2000) has been used with outstanding results on learning aerobatic manoeuvres with an autonomous helicopter, see Ng *et al.* (2004b) and Ng *et al.* (2004a). A probablistic model of the system is trained using locally weighted linear regression (see the article by Ting *et al.* (2010) for an overview) then gradient descent is used to find a good policy. A number of random deterministic models are then drawn from this distribution and optimisation takes place with respect to the average over all the responses.

Another gradient-based method proposed in the thesis by Deisenroth (2009) and published in Deisenroth & Rasmussen (2011) seeks to provide a full treatment of model uncertainty by learning a distribution over the dynamics function space using Gaussian processes. The algorithm then uses this probabilistic model to make a prediction of the distribution over trajectories $\boldsymbol{\tau}$ based on the uncertainty in the model and given a set of policy parameters $\boldsymbol{\psi}$. The derivative vector $\nabla J(\boldsymbol{\psi})$ can also be found analytically therefore learning can proceed using gradient descent. Note that we do not have to resort to sampling techniques as with PEGASUS since the distribution over trajectories can be determined analytically. The approach has led to huge savings in terms of learning efficiency, see Deisenroth & Rasmussen (2011) for a comparison with different methods on the cart-pendulum swing-up and balance task.

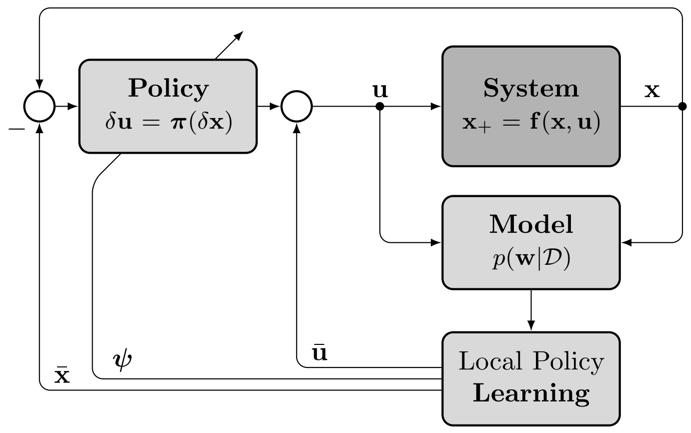

Policy
$\delta\mathbf{u} = \boldsymbol{\pi}(\delta\mathbf{x})$
$-$
$\mathbf{u}$
System
$\mathbf{x}_+ = \mathbf{f}(\mathbf{x},\mathbf{u})$
$\mathbf{x}$
Model
$p(\mathbf{w}|\mathcal{D})$
$\boldsymbol{\psi}$
$\bar{\mathbf{u}}$
$\bar{\mathbf{x}}$
Local Policy
Learning

Figure 2.5: Layout of a local learning scheme with learned dynamics model parameterised by $\mathbb{E}[\mathbf{w}|\mathcal{D}]$. The iLQG or DDP algorithms would be placed in the local learning block in this diagram.

We further note that these algorithms could be likened to the indirect self-tuning regulator shown in Fig. 2.2. Instead of learning the distribution over a set of parameters $\mathbf{w}$ related directly to the model structure, the use of Gaussian processes means we can place a distribution over the space of functions $\mathbf{f}$ directly. How this can be achieved will be discussed at length in the following chapter.

2.3.6 Local Learning

Differential Dynamic Programming and Iterative LQG

An important class of learning methods, which has found particular success in robotics is that of local learning. This class of methods is inspired by the Differential Dynamic Programming (DDP) framework of Jacobson & Mayne (1970). DDP maintains a local second order approximation to the action-value function around some nominal trajectory $\bar{\boldsymbol{\tau}} = \{\bar{\mathbf{x}}_0, \bar{\mathbf{u}}_0 \cdots \bar{\mathbf{x}}_H, \bar{\mathbf{u}}_H\}$ using second order Taylor series approximations of the dynamics and cost function. The nominal trajectory is then iteratively improved to find an optimal state-control sequence with the addition of local linear feedback policies $\delta\mathbf{u}_k = \boldsymbol{\pi}_k(\delta\mathbf{x}_k)$. A recent example of this can be found in Todorov & Tassa (2009). Abbeel *et al.* (2007) use DDP together with a learned dynamics model for learning helicopter aerobatic manœvres. Morimoto *et al.* (2003) use DDP with a minimax cost criterion (in the spirit of $\mathcal{H}_\infty$ robust control) and a learned error dynamics model to learn a robust walking motion for a biped robot.

A newer methodology called iterative LQG (iLQG) introduced by Todorov & Li (2005) is based on a first order Taylor series expansion of the dynamics and solves a local LQG problem at each timestep. It has been utilised for incremental online learning in Mitrovic *et al.* (2010b) where the iLQG framework was combined with a learned dynamics model obtained by the LWPR (Locally Weighted Projection Regression) algorithm of Vijayakumar *et al.* (2005). Fig. 2.5 depicts the general framework of local learning using iLQG. This framework would equally lend itself to the use of DDP in the place of iLQG as the design methodology. Mitrovic *et al.* (2010a) postulate a method for including model uncertainty in the learning process.

Ting *et al.* (2010) state that the difference between adaptive control and learning control is that learning control is allowed to fail during the learning process, resembling how humans and animals learn. Adaptive control emphasises single trial convergence in systems where ‘failure’ is not an option.

Policy Improvement with Path Integrals

A recent approach to reinforcement learning has been adopted by Theodorou *et al.* (2010a) based on the framework of stochastic optimal control with path integrals, Policy Improvement with Path Integrals (PI^2). It has strong links to DDP in that it is a trajectory-based optimisation scheme and can also learn local feedback rules as demonstrated by Buchli *et al.* (2011). However, it does not require a model of the system dynamics or backwards-time integration over a trajectory but requires only sample trajectories from the system. It is an appealing approach because neither matrix inversions nor gradient learning rates are required, in fact the only free parameter in the algorithm is the exploration noise. Significant improvement in learning speed over policy-gradient methods for trajectory planning problems has been reported in Theodorou *et al.* (2010b).

2.4 Summary

We conclude this chapter with the observation that there are a plethora of algorithms available in the literature of both adaptive control and reinforcement learning that tackle the general problem shown in Fig. 1.1(a). These can be split into model-based and model-free algorithms, in which model-based approaches learn an explicit model of the unknown dynamics of the system. It is clear from our review that adaptive control approaches have tended to place tighter assumptions on the form of the unknown dynamics in order to satisfy stability criteria and put bounds on the performance. On the other hand, the model-free reinforcement learning literature has tended to produce algorithms that can guarantee convergence to the optimal policy but in practice require many thousands of trials on the system to find it. The use of a deterministic model can lead to the issue of model-bias. This can be addressed through the use of a probabilistic model which explicitly takes account of the modelling uncertainty. Of the model based solutions from reinforcement learning we intend to pursue that of Deisenroth & Rasmussen (2011) since it makes use of Gaussian process priors, a general and flexible setting for incorporating useful prior knowledge, as we shall demonstrate in the following chapter. We believe that the local learning approach is a promising one in terms of efficient use of interaction data and a setting for incorporating prior information but this is beyond the scope of this thesis.

CHAPTER 3

Probabilistic Learning Control

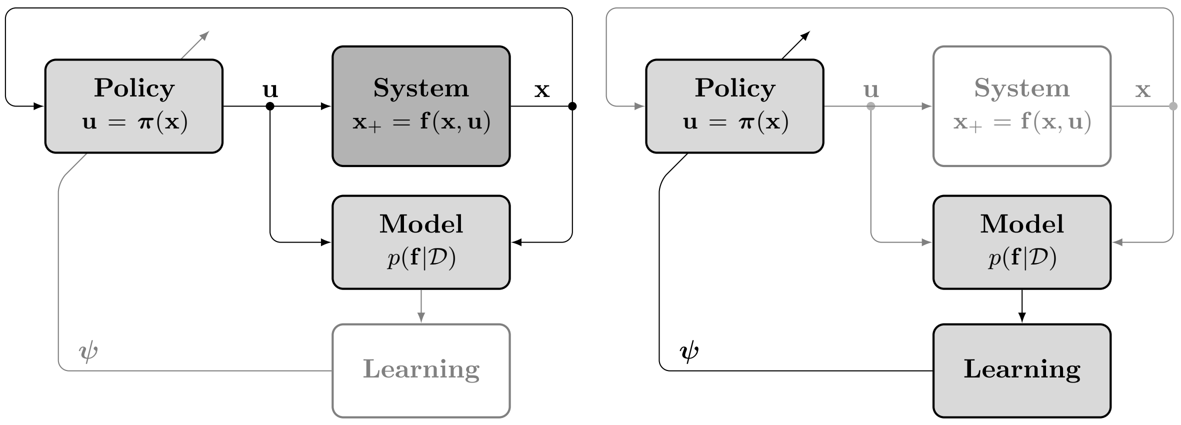
3.1 Overview

3.1.1 Problem Formulation

The problem under consideration in this chapter is how to train a feedback control policy for a given nonlinear system based on a model obtained from interaction data. The model is probabilistic in the sense that it will define a distribution over the function space where the real dynamics lie. This means that if the model is queried at a deterministic point in the state-action space it will return a distribution over the states at the following timestep. The uncertainty in the model will encode predictive uncertainty. The nonlinear system will have the following general form

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \tag{3.1}$$

with states $\mathbf{x} \in \mathbb{R}^E$, actions $\mathbf{u} \in \mathbb{R}^F$ and discrete time instance $k \in \mathbb{Z}_{[1,\infty)}$. Further, define the joint state-action vector $\mathbf{z} = [\mathbf{x}; \mathbf{u}] \in \mathbb{R}^D$ and the sampling interval $\delta_t \in \mathbb{R} > 0$. The control objective is encoded in the stage cost $c : \mathbb{R}^D \to \mathbb{R}^+$ over some predictive horizon of H steps. The task is to determine a fixed state-feedback policy $\boldsymbol{\pi} : \mathbb{R}^E \to \mathbb{R}^F$, parameterised by $\boldsymbol{\psi}$, to



(a) Interaction phase (b) Simulation phase

Figure 3.1: Framework for model-based learning control. The *interaction* phase involves the policy running in closed-loop with the system according to the current best parameter setting $\boldsymbol{\psi}$. During this phase the model updates its training data set $\mathcal{D}$ and then updates its internal model of the system. The *simulation* phase then involves improving the policy based on the updated model. This procedure then iterates until the task is has been achieved.

satisfy the following optimisation problem

$$\text{minimise} \qquad J(\boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{k=0}^{H} c(\mathbf{x}_k, \mathbf{u}_k)\,\middle|\, p(\mathbf{x}_0)\right] \tag{3.2}$$

$$\text{subject to} \qquad \mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \quad \text{where} \quad \mathbf{f} \sim p\big(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta}\big) \tag{3.3}$$

$$\mathbf{u}_k = \boldsymbol{\pi}(\mathbf{x}_k)$$

with cost function $J(\boldsymbol{\psi})$, state-action trajectory $\boldsymbol{\tau} := [\mathbf{x}_0; \mathbf{u}_0 \dots \mathbf{x}_H; \mathbf{u}_H]$ and initial state distribution $p(\mathbf{x}_0)$. The model in this problem is defined as the distribution over function space $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$ given the observed training data set $\mathcal{D}$ and hyperparameters $\boldsymbol{\theta}$. Hyperparameters are so-called because they parameterise a distribution over functions rather than the function itself. The details of how a distribution over functions can actually be defined will be discussed in Sec. 3.4. Note that the expectation operator can be brought inside the summation to give

$$J(\boldsymbol{\psi}) = \sum_{k=0}^{H} \mathbb{E}_{\mathbf{z}_k}\big[c(\mathbf{z}_k)\big] \tag{3.4}$$

where $\mathbf{z} := [\mathbf{x}; \mathbf{u}]$ as in the previous chapter. This formulation aims to reduce the expected, or average, cost over a finite horizon rather than minimise some worst case scenario as in standard robust control. This choice was made because when learning a task, the learner (or *agent*) is often aiming to improve his performance on average based on his belief about the world rather than to avoid some worst case scenario.

3.1.2 Model-Based Approach

The algorithmic framework for tackling this problem follows the model-based approach depicted in Fig. 3.1. This consists of two learning phases: *interaction* and *simulation*. The interaction phase consists of the agent applying control actions to the system based on the current policy and building up an observed data set $\mathcal{D}$ of the dynamic response of the system. Based on this data set, the internal probabilistic model $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$ is updated. Once a new model has been trained, the performance of the current policy $\boldsymbol{\pi}$ is evaluated using the cost function in Eq. (3.2) and the updated model. The policy is then improved such that its parameters belong to the set $\boldsymbol{\psi} \in \arg\min J(\boldsymbol{\psi})$ evaluated with respect to the current model. This is an iterative procedure that is repeated until the policy is deemed to have achieved satisfactory performance. If the cost function in Eq. (3.2) and its derivatives with respect to the policy parameters can be evaluated analytically, the evaluation and improvement steps of the simulation phase can be achieved efficiently using gradient-descent methods. We note that if the derivatives cannot be obtained analytically then there are methods which can provide an estimate, such as finite differences. However, this is generally a less efficient approach, especially when dealing with a large number of policy parameters.

3.1.3 The Gaussian Assumption

Evaluating the cost function given in Eq. (3.2) is in general only possible using Monte-Carlo sampling methods. Despite major advances in the efficiency of these methods, the full optimisation problem will still be highly computationally demanding in many cases of interest. Therefore another solution is sought. The Linear Quadratic Gaussian (LQG) control problem, in which $\mathbf{f}$ is restricted to linear functions, the cost is quadratic and the initial state distribution is Gaussian $\mathbf{x}_0 \sim \mathcal{N}$, yields an analytically tractable optimisation problem in which the optimal policy $\boldsymbol{\pi}$ is linear. One of the main reasons that an analytical solution is available is due to the fact that the distribution $p(\mathbf{x}_k, \mathbf{u}_k)$ remains Gaussian at each timestep and the expectation of the quadratic cost can be evaluated analytically. This will not be true for an arbitrary nonlinear system and non-quadratic cost. Therefore the following assumption will be made in order to find an approximation to the real problem in Eqs. (3.2)–(3.3) that is analytically tractable:

Assumption 3.1 (Moment Matching). *Given $p(\mathbf{z}_{k-1})$ is Gaussian, the resulting distribution of the next state-action $p(\mathbf{z}_k)$ is replaced by the Gaussian with mean and covariance equal to the mean and covariance of the actual distribution: $\mathbb{E}_{\mathbf{z}_{k-1},\mathbf{f}}[\mathbf{z}_k|\boldsymbol{\pi}]$ and $\mathrm{cov}_{\mathbf{z}_{k-1},\mathbf{f}}[\mathbf{z}_k|\boldsymbol{\pi}]$ respectively.*

This is known as moment matching since the real distribution is approximated as a Gaussian with the same first and second moments. For linear dynamics $\mathbf{z}_k = \mathbf{M}\mathbf{z}_{k-1}$ and initial state-action distribution $\mathbf{z}_{k-1} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ the Gaussian assumption is correct and yields the true distribution $\mathbf{z}_k \sim \mathcal{N}\big(\mathbf{M}\boldsymbol{\mu}, \mathbf{M}\boldsymbol{\Sigma}\mathbf{M}^\top\big)$. This assumption will be discussed in greater detail in Sec. 3.3.

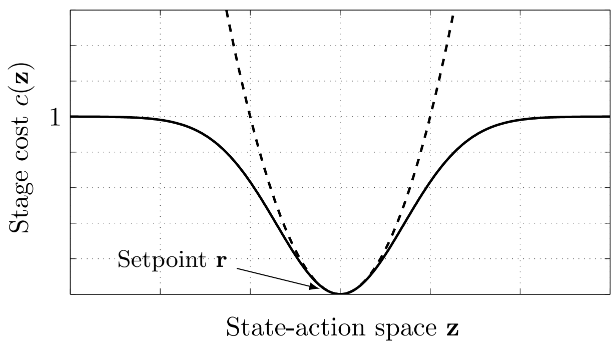


Figure 3.2: Stage-costs for which the expectation in Eq. (3.5) can be evaluated analytically for $\mathbf{z}_k \sim \mathcal{N}$. These functions are useful for penalising deviations of the state-action $\mathbf{z}$ from a given setpoint $\mathbf{r}$. The quadratic is shown by the dashed line and the inverted-Gaussian is shown by the solid.

3.2 Learning Control Policies

3.2.1 Policy Evaluation

Stage-Costs

Due to Assumption 3.1, the approximate distribution over state-action at each timestep is Gaussian $\mathbf{z}_k \sim \mathcal{N}$. In order to maintain an analytic evaluation problem, we require stage costs in which the expectation

$$\mathbb{E}_{\mathbf{z}_k}\big[c(\mathbf{z}_k)\big] = \int c(\mathbf{z}_k)p(\mathbf{z}_k)\mathrm{d}\mathbf{z}_k \tag{3.5}$$

can be evaluated analytically. If we desire to penalise the deviation of the state-action from some given setpoint $\mathbf{r}$ then the standard quadratic and the inverted squared exponential (or "inverted-Gaussian") stage costs may be considered

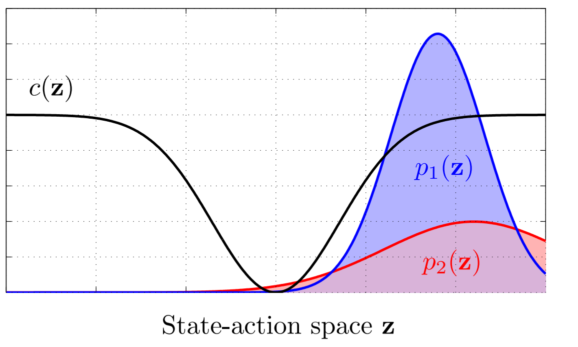
$$c_{\text{quad}}(\mathbf{z}) = \tfrac{1}{2}(\mathbf{z}-\mathbf{r})^\top \mathbf{Q}(\mathbf{z}-\mathbf{r}) \tag{3.6}$$

$$c_{\text{gauss}}(\mathbf{z}) = 1 - \exp\Big(-\tfrac{1}{2}(\mathbf{z}-\mathbf{r})^\top \mathbf{Q}(\mathbf{z}-\mathbf{r})\Big) \tag{3.7}$$

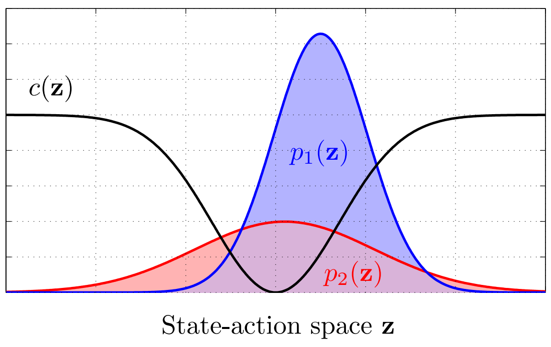
where $\mathbf{Q}$ is a positive semi-definite weighting matrix. These are depicted in Fig. 3.2 which shows that close to $\mathbf{r}$ these functions are equivalent. The factor of a half is included in the definitions so that the $\mathbf{Q}$ matrix can be interpreted as the covariance matrix of an unnormalised Gaussian and its elements tuned such that any state-action more than two standard-deviations away from the setpoint will incur a stage cost of approximately one. The inverted-Gaussian stage cost is locally quadratic and can be viewed as a saturated quadratic. The expectations of these costs are given as

$$\mathbb{E}_{\mathbf{z}}\big[c_{\text{quad}}(\mathbf{z})\big] = \tfrac{1}{2}(\boldsymbol{\mu}-\mathbf{r})^\top \mathbf{Q}(\boldsymbol{\mu}-\mathbf{r}) + \tfrac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}\mathbf{Q}) \tag{3.8}$$

$$\mathbb{E}_{\mathbf{z}}\big[c_{\text{gauss}}(\mathbf{z})\big] = 1 - \big|\mathbf{I}+\boldsymbol{\Sigma}\mathbf{Q}\big|^{-1/2} \exp\Big(-\tfrac{1}{2}(\boldsymbol{\mu}-\mathbf{r})^\top \overbrace{\mathbf{Q}^{1/2}(\mathbf{I}+\mathbf{Q}^{1/2}\boldsymbol{\Sigma}\mathbf{Q}^{1/2})^{-1}\mathbf{Q}^{1/2}}^{(\mathbf{Q}^{-1}+\boldsymbol{\Sigma})^{-1}}(\boldsymbol{\mu}-\mathbf{r})\Big) \tag{3.9}$$



(a) Distributions with means far away from the setpoint



(b) Distributions with means close to the setpoint

Figure 3.3: Illustration of how a "natural" exploration of the state-action space can arise through use of an inverted-Gaussian stage cost. The stage cost is shown as a black line and two distributions over state-action space are shown by the blue and red curves, with the means shown explicitly.

for $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The use of the inverted-Gaussian stage cost leads to a useful behaviour in the context of probabilistic learning control.

Natural Exploration-Exploitation

The motivation for preferring the inverted-Gaussian stage cost Eq. (3.7) over the conventional quadratic form in the context of learning control will now be outlined. Consider the optimal control problem posed in Eq. (3.4) in which the initial state is known exactly $\boldsymbol{\Sigma}_0 = \mathbf{0}$ and the only source of uncertainty comes from the probabilistic model which is used to capture modelling uncertainty. As part of the offline simulation phase of the learning framework posed in Sec. 3.1.2 now consider that the policy has to make a choice between two state-action densities $p_1(\mathbf{z})$ and $p_2(\mathbf{z})$ as shown in Fig. 3.3(a). If modelling uncertainty was ignored then the policy would favour the state distribution with mean closest to the setpoint $\mathbf{r}$ regardless of the uncertainty associated with the prediction. However, if uncertainty is taken into account, the policy will favour the more uncertain state with mean further from the setpoint since this has more density in the low cost region therefore yields a lower expected cost.

The converse situation is depicted in Fig. 3.3(b) where the probabilistic framework would favour policies that chose the more certain state despite the fact that its mean is further from the setpoint. This example illustrates that using a probabilistic model will lead to policies which drive the system into uncertain areas of its state-action space in high cost regions but favour cautious behaviour when close to the setpoint. This "natural" exploration will aid the model learning process.

The standard quadratic cost would not display this behaviour since the covariance of the state-action is always explicitly penalised by the additive term shown in Eq. (3.8). Therefore the quadratic cost leads only to behaviour which seeks to exploit its current knowledge and not explore. It should be noted that this is not the standard exploration-exploitation tradeoff that appears in the reinforcement learning literature since this behaviour arises from being greedy with respect to a probabilistic model rather than explicitly applying random inputs.

3.2.2 Policy Improvement

Since an analytic approximation to the cost function in Eq. (3.2) is being considered, the gradient of this cost with respect to the policy parameters $\mathrm{d}J(\boldsymbol{\psi})/\mathrm{d}\boldsymbol{\psi}$ can also be obtained in closed form. This means that we can make efficient use of gradient descent optimisation in order to carry out the improvement step of the simulation phase. The derivatives have the following structure

$$\frac{\mathrm{d}J(\boldsymbol{\psi})}{\mathrm{d}\boldsymbol{\psi}} = \sum_{k=0}^{H} \frac{\mathrm{d}\mathbb{E}_{\mathbf{z}_k}\big[c(\mathbf{z}_k)\big]}{\mathrm{d}\boldsymbol{\psi}} \tag{3.10}$$

Due to the Gaussian assumption given in Assumption 3.1, the expectation of the stage cost $\mathbb{E}_{\mathbf{z}_k}\big[c(\mathbf{z}_k)\big]$ is only dependant on the mean and variance of this distribution. Therefore the full derivative can be split up into partial derivatives

$$\frac{\mathrm{d}\mathbb{E}_{\mathbf{z}_k}\big[c(\mathbf{z}_k)\big]}{\mathrm{d}\boldsymbol{\psi}} = \frac{\partial\mathbb{E}_{\mathbf{z}_k}\big[c(\mathbf{z}_k)\big]}{\partial\boldsymbol{\mu}_k}\frac{\mathrm{d}\boldsymbol{\mu}_k}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\mathbb{E}_{\mathbf{z}_k}\big[c(\mathbf{z}_k)\big]}{\partial\boldsymbol{\Sigma}_k}\frac{\mathrm{d}\boldsymbol{\Sigma}_k}{\mathrm{d}\boldsymbol{\psi}} \tag{3.11}$$

where $\boldsymbol{\mu}_k = \mathbb{E}_{\mathbf{z}_{k-1},\mathbf{f}}[\mathbf{z}_k|\boldsymbol{\pi}]$ and $\boldsymbol{\Sigma}_k = \mathrm{cov}_{\mathbf{z}_{k-1},\mathbf{f}}[\mathbf{z}_k|\boldsymbol{\pi}]$. The dimensions of these derivatives are defined according to the matrix calculus convention outlined in Appendix A.3. The derivatives of the mean and covariance at each step can be calculated in the following recursive manner

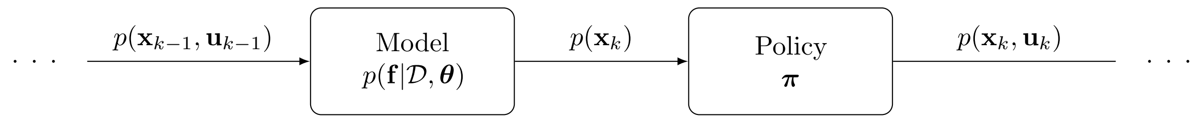
$$\frac{\mathrm{d}\boldsymbol{\mu}_k}{\mathrm{d}\boldsymbol{\psi}} = \frac{\partial\boldsymbol{\mu}_k}{\partial\boldsymbol{\mu}_{k-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{k-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\mu}_k}{\partial\boldsymbol{\Sigma}_{k-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{k-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\mu}_k}{\partial\boldsymbol{\psi}} \tag{3.12}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_k}{\mathrm{d}\boldsymbol{\psi}} = \frac{\partial\boldsymbol{\Sigma}_k}{\partial\boldsymbol{\mu}_{k-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{k-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\Sigma}_k}{\partial\boldsymbol{\Sigma}_{k-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{k-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\Sigma}_k}{\partial\boldsymbol{\psi}} \tag{3.13}$$

for $k \geq 1$ where each partial derivative is dependent on the type of dynamics model and policy structure used. The mean and covariance at $k = 0$ can be partitioned as follows

$$\boldsymbol{\mu}_0 = \begin{bmatrix}\boldsymbol{\mu}_{\mathbf{x}_0}\\ \boldsymbol{\mu}_{\mathbf{u}_0}\end{bmatrix}, \qquad \boldsymbol{\Sigma}_0 = \begin{bmatrix}\boldsymbol{\Sigma}_{\mathbf{x}_0} & \boldsymbol{\Sigma}_{\mathbf{x}\mathbf{u}_0}\\ \boldsymbol{\Sigma}_{\mathbf{x}\mathbf{u}_0}^\top & \boldsymbol{\Sigma}_{\mathbf{u}_0}\end{bmatrix}$$

where $\mathrm{d}\boldsymbol{\mu}_{\mathbf{x}_0}/\mathrm{d}\boldsymbol{\psi} = \mathbf{0}$ and $\mathrm{d}\boldsymbol{\Sigma}_{\mathbf{x}_0}/\mathrm{d}\boldsymbol{\psi} = \mathbf{0}$ since the initial state distribution is fixed. The exact form of these derivatives depend on the form of the probabilistic model and the policy used. These calculations are implemented in an iterative fashion using MATLAB and can be found in Appendix C.1 implemented by the function `value.m`.



. . .
$p(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$
Model
$p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$
$p(\mathbf{x}_k)$
Policy
$\boldsymbol{\pi}$
$p(\mathbf{x}_k, \mathbf{u}_k)$
. . .

Figure 3.4: Propagation of uncertainty from a given state-action pair $(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$ to the pair at the following timestep $(\mathbf{x}_k, \mathbf{u}_k)$ given a distribution over dynamics functions $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$ and the control policy $\boldsymbol{\pi}$.

3.3 State-Action Prediction

3.3.1 Propagating Uncertainty

Attention is now turned to the problem of inferring the assumed Gaussian distribution over state-action space $p(\mathbf{z}_k)$ given the distribution at the previous timestep $p(\mathbf{z}_{k-1}) \sim \mathcal{N}$. This can then be cascaded in order to build up a distribution over the full trajectory $p(\boldsymbol{\tau})$. It would be useful to treat this problem in two stages as shown in Fig. 3.4. First, find the distribution $p(\mathbf{x}_k)$ using the probabilistic dynamics model $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$. Then find the joint distribution $p(\mathbf{x}_k, \mathbf{u}_k)$ using the policy $\boldsymbol{\pi}$. In order to do this, a tighter assumption is made:

Assumption 3.2. *Given $p(\mathbf{z}_{k-1})$ is Gaussian, the resulting distribution of the next state-action $p(\mathbf{z}_k)$ is replaced by the Gaussian distribution*

$$\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbb{E}_{\mathbf{z}_{k-1},\mathbf{f}}[\mathbf{x}_k] \\ \mathbb{E}_{\boldsymbol{\chi}_k}[\mathbf{u}_k|\boldsymbol{\pi}] \end{bmatrix}, \begin{bmatrix} \mathrm{cov}_{\mathbf{z}_{k-1},\mathbf{f}}[\mathbf{x}_k] & \mathrm{cov}_{\boldsymbol{\chi}_k}[\boldsymbol{\chi}_k, \mathbf{u}_k|\boldsymbol{\pi}] \\ \mathrm{cov}_{\boldsymbol{\chi}_k}[\mathbf{u}_k, \boldsymbol{\chi}_k|\boldsymbol{\pi}] & \mathrm{cov}_{\boldsymbol{\chi}_k}[\mathbf{u}_k|\boldsymbol{\pi}] \end{bmatrix} \right)$$

where $p(\boldsymbol{\chi}_k)$ is the moment matched approximation of the real distribution $p(\mathbf{x}_k)$.

Note that Assumption 3.1 uses the real distribution $p(\mathbf{x}_k)$ as the input to the control policy whereas Assumption 3.2 uses the moment matched approximation $p(\boldsymbol{\chi}_k)$. From now on, no distinction will be made between the real distribution $p(\mathbf{x})$ and its approximation $p(\boldsymbol{\chi})$.

3.3.2 Gaussian Approximation Schemes

The use of the moment matching scheme given in Assumptions 3.1 and 3.2 will now be motivated. For now, consider the propagation of uncertainty from the input $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ through a distribution over functions $p(\mathbf{f})$ to the next state $\mathbf{f}(\mathbf{z})$. In this case we assume that the probabilistic model returns a Gaussian distribution when queried at a deterministic input. This is the first step shown in Fig. 3.4. There are a number of methods available for approximating the real distribution $p\big(\mathbf{f}(\mathbf{z})\big)$ with a Gaussian $\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$. We shall briefly introduce a number of these methods now

1. **Taylor Series Expansions.** Approximate the dynamics $p(\mathbf{f})$ with a finite Taylor series expansion of the real nonlinear equations about the input mean $\boldsymbol{\mu}$. Then for a first order

expansion for example we have

$$\boldsymbol{\mu}_* = \mathbb{E}_{\mathbf{f}}[\mathbf{f}(\boldsymbol{\mu})] \qquad\qquad \boldsymbol{\Sigma}_* = \big(\partial\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\boldsymbol{\mu})]/\partial\boldsymbol{\mu}\big)^2 + \mathrm{cov}_{\mathbf{f}}[\mathbf{f}(\boldsymbol{\mu})]$$

This is equivalent to taking linearisations, which is the method that is used in the prediction step of the Extended Kalman Filter (EKF). See Chapter 8 of Anderson & Moore (1979) for more information on the EKF.

2. **Unscented Transformation.** Instead of making assumptions about the structural nature of the dynamics model or policy, the unscented transformation takes a finite number of (weighted) "sigma-points" $\mathcal{S} = \{\hat{\mathbf{z}}_1 \dots \hat{\mathbf{z}}_S\}$ with the same second-order statistics as $p(\mathbf{z})$. Mathematically $\mathbb{E}_{\mathbf{z}}[\mathbf{z}] = \sum_{s=1}^S W_s\hat{\mathbf{z}}_s$ and $\mathrm{cov}_{\mathbf{z}}[\mathbf{z}] = \sum_{s=1}^S W_s(\hat{\mathbf{z}}_s - \mathbb{E}_{\mathbf{z}}[\mathbf{z}])(\hat{\mathbf{z}}_s - \mathbb{E}_{\mathbf{z}}[\mathbf{z}])^\top$ where $\sum_{s=1}^S W_s = 1$, $W_s > 0$. It then maps them through the mean of the probabilistic dynamics model to obtain the transformed set $\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\mathcal{S})] = \{\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\hat{\mathbf{z}}_1)] \dots \mathbb{E}_{\mathbf{f}}[\mathbf{f}(\hat{\mathbf{z}}_S)]\}$. The mean and covariance are then set to that of the (weighted) statistics of the transformed data set

$$\boldsymbol{\mu}_* = \sum_{s=1}^S W_s\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\hat{\mathbf{z}}_s)] \quad \boldsymbol{\Sigma}_* = \sum_{s=1}^S W_s\big(\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\hat{\mathbf{z}}_s)] - \boldsymbol{\mu}_*\big)\big(\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\hat{\mathbf{z}}_s)] - \boldsymbol{\mu}_*\big)^\top + \mathrm{cov}_{\mathbf{f}}[\mathbf{f}(\boldsymbol{\mu})]$$

respectively to give an unbiased prediction. This method was proposed as the prediction step of the Unscented Kalman Filter (UKF) of Julier & Uhlmann (1997).

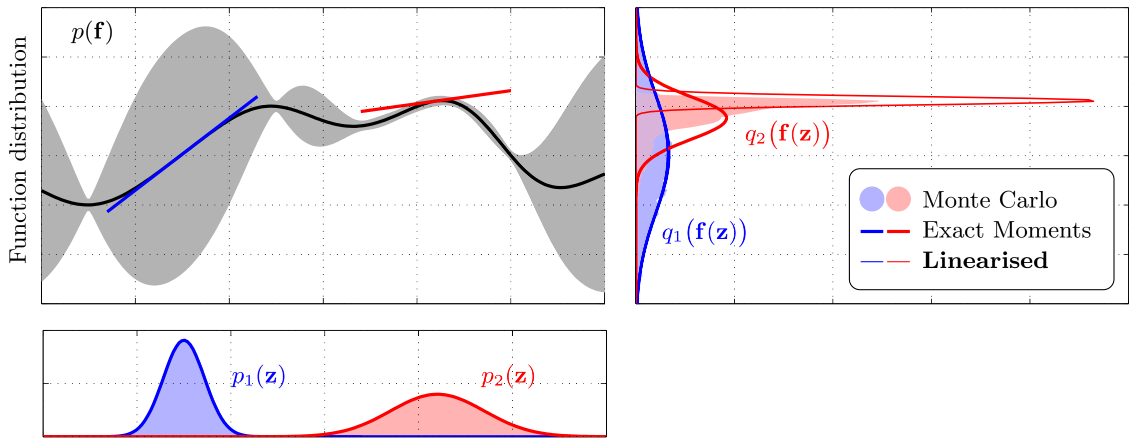
3. **Exact Moment Matching.** In situations where they are possible to obtain, the moment matching approximation finds the exact mean and covariance

$$\boldsymbol{\mu}_* = \mathbb{E}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})] \qquad\qquad \boldsymbol{\Sigma}_* = \mathrm{cov}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})]$$

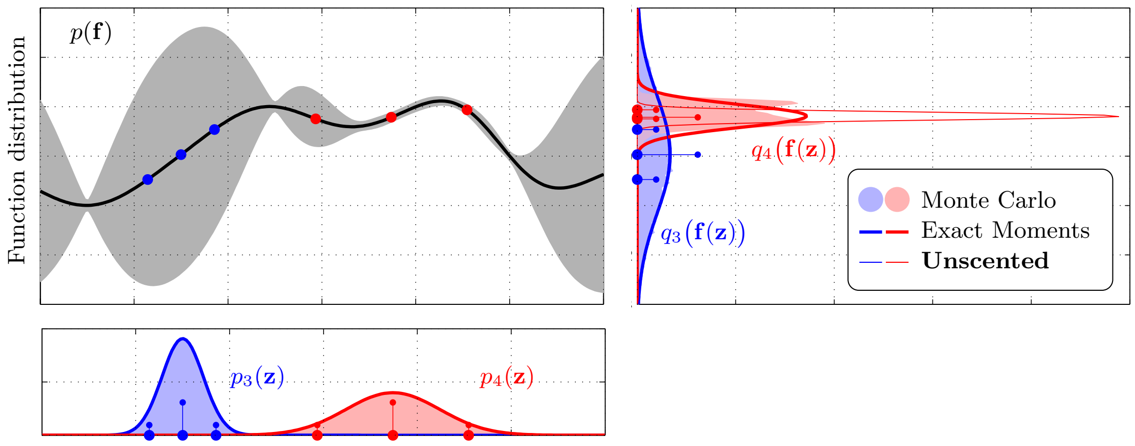
of the actual distribution $p\big(\mathbf{f}(\mathbf{z})\big)$. To complete the filtering parallel of the previous two examples, the moment matching technique is used in the prediction step of an Assumed Density Filter (ADF). See Chapter 12 of Maybeck (1982) for more information on the ADF.

Both the Taylor series expansion and the unscented transformation methods can potentially suffer from degeneracy as demonstrated in Fig. 3.5. In the cases depicted by the blue distributions the function $\mathbf{f}$ is close to linear in the region where the distribution has high density and therefore all methods make good approximations to the actual distribution. An example of the degeneracy of the linear approximation is shown by the red plot in Fig. 3.5(a) where the gradient approaches zero and the output distribution approaches the delta distribution and ignores much of the mass of the real output distribution. Similarly, the potential degeneracy of the unscented transformation is demonstrated in the red case shown in Fig. 3.5(b). A further example is given in Deisenroth *et al.* (2012).

The moment matching approach does not suffer from degeneracy for obvious reasons. We further note that the moment matching approach is, in a sense, a conservative approximation. In order



(a) Linear approximation. Linearisations of the function about the state-action mean $\mathbb{E}[\mathbf{z}]$ are shown.



(b) Unscented transformation. Sigma points $\mathcal{S}$ along with their projections $\mathbf{f}(\mathcal{S})$ and weights W are shown.

Figure 3.5: Propagation of uncertainty through a deterministic mapping $\mathbf{f}$ using linearisation, the unscented transformation and exact moment matching. The lower plots show two distributions over $\mathbf{z}$. The central plot shows the stochastic function $\mathbf{f}$ with mean shown by the solid line and the 2σ uncertainty region in gray. The right hand plot shows the true output distributions (obtained by sampling) and the various approximation schemes. Fig. (a) is an example of the degeneracy of the linearised approximation and Fig. (b) shows the degeneracy of the unscented transform.

to appreciate this, consider the Kullback-Leibler (KL) divergence $\mathrm{KL}(q||p)$ of the underlying distribution p with respect to some Gaussian distribution q. We can prove that the Gaussian which minimises this divergence is the moment matched solution. For details of this proof see Appendix A.2. The nature of the KL divergence ensures that the approximate distribution q is non-negligible where the real distribution p is non-negligible, but not the other way around. In this sense the predictions will be conservative.

The advantage of using a Taylor series or unscented transformation based method is that they can be applied to a broader class of systems whereas the moment matching assumption is more restrictive. However, since a very useful set of models is amenable to the moment matching approach this was the method that has been adopted in this thesis.

3.3.3 Modelling Uncertainty

In a learning context it is important to treat modelling uncertainty in a rigorous manner. This is readily handled in a Bayesian framework. Given a parametric representation of the dynamics consisting of a linear sum of appropriate basis functions the Bayesian framework provides a way of making predictions based on a prior distribution over parameters and an observed data set.

3.4 Bayesian Modelling

3.4.1 Problem Outline

This section considers the problem of finding a probabilistic model $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$ defining a distribution over the function space, given some training data set $\mathcal{D}$ and hyperparameters $\boldsymbol{\theta}$ and using this model to make predictions. This problem can be addressed by a Bayesian method known as Gaussian Processes (GPs). A Gaussian process is defined by Rasmussen & Williams (2006) in Definition 3.1.

Definition 3.1 (Gaussian process). *A collection of random variables, any finite number of which have a joint Gaussian distribution.*

Informally speaking, GPs are a means of placing a prior over a space of functions $p(\mathbf{f}|\boldsymbol{\theta})$ from which samples can be drawn, an extension of the idea of multivariate Gaussian distributions to infinitely long vectors, or functions. Once training data has been observed, this can be updated to a posterior distribution $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$ which is the probabilistic model required by the learning framework outlined in Sec. 3.1. Given a deterministic input $\mathbf{z}$, GPs provide a predictive Gaussian distribution over possible next states with mean $\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\mathbf{z})|\mathbf{z}, \mathcal{D}]$ and covariance $\mathrm{cov}_{\mathbf{f}}[\mathbf{f}(\mathbf{z})|\mathbf{z}, \mathcal{D}]$. Explicit dependence on the hyperparameters $\boldsymbol{\theta}$ will be dropped from now on for clarity. However, from Sec. 3.3, making predictions with respect to uncertain $\mathbf{z} \sim \mathcal{N}$ is required for the moment matching framework. In other words, making predictions of the mean $\mathbb{E}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})|\mathcal{D}]$ and covariance $\mathrm{cov}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})|\mathcal{D}]$.

Now the exact form of the training data set $\mathcal{D}$ will be discussed. Data will be given in the form of n state-actions and next-state pairs

$$\mathbf{Z} = \begin{bmatrix} \mathbf{x}_{k_1}^\top & \mathbf{u}_{k_1}^\top \\ \vdots & \vdots \\ \mathbf{x}_{k_n}^\top & \mathbf{u}_{k_n}^\top \end{bmatrix} \in \mathbb{R}^{n\times D} \qquad \text{and} \qquad \mathbf{Y} = \begin{bmatrix} \mathbf{x}_{k_1+1}^\top \\ \vdots \\ \mathbf{x}_{k_n+1}^\top \end{bmatrix} \in \mathbb{R}^{n\times E} \tag{3.14}$$

for $-\infty < k_1 < \cdots < k_n < 0$ not necessarily sampled consecutively. For the purposes of the following discussion, state-action pairs are referred to as "inputs" and the next-states are referred to as "outputs". Now, an important issue arises here. In reality all of these measurements will be corrupted by noise, normally assumed to be additive Gaussian noise. However, the GP framework assumes that only the outputs $\mathbf{Y}$ are corrupted by additive noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}})$ with diagonal covariance $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$ and uncorrelated between timesteps but the inputs are noise free. The GP approach to a fully noisy data set will be discussed later, but for now this assumption will be made. Therefore, the noisy output measurements will be referred to by $\mathbf{Y}$, while the noise-free set is referred to by the matrix $\mathbf{F}$.

It is worthy of note that the following presentation of Gaussian process theory is novel in the sense that we deal with the multivariate case from the very outset. Most textbooks and presentations of the fundamental theory talk about the multivariate case as an afterthought.

3.4.2 Parametric Approach

Prior

The linear parametric approach assumes that the unknown function $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^E$ belongs to the set of functions parameterised by

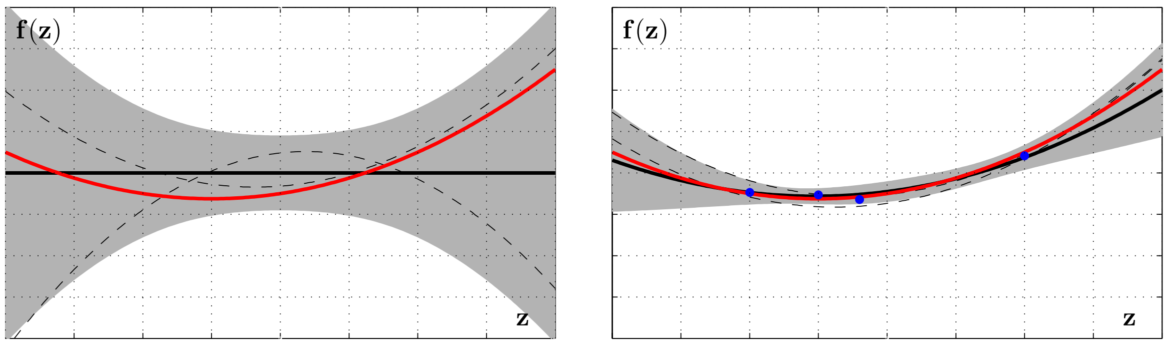
$$\mathbf{f}(\mathbf{z}) = \boldsymbol{\Phi}(\mathbf{z})^\top \mathbf{w} \tag{3.15}$$

where $\mathbf{w} \in \mathbb{R}^P$ is a vector of weights and the feature matrix $\boldsymbol{\Phi}(\mathbf{z}) \in \mathbb{R}^{P\times E}$ is made up of a number of basis functions. Note that the feature matrix is often of a form such that there are no cross correlations between output dimensions through common weights. In other words $\boldsymbol{\Phi}(\mathbf{z}) = \mathbf{I} \otimes \boldsymbol{\phi}(\mathbf{z})$. The feature matrix is defined to handle matrix inputs by defining the following convention

$$\boldsymbol{\Phi}(\mathbf{Z}) := \Big[\Phi[:,1](\mathbf{Z}) \quad \ldots \quad \Phi[:,E](\mathbf{Z})\Big] \in \mathbb{R}^{P\times En} \tag{3.16}$$

$$\Phi[:,i](\mathbf{Z}) := \Big[\Phi[:,i](\mathbf{z}_{k_1}) \quad \ldots \quad \Phi[:,i](\mathbf{z}_{k_n})\Big] \in \mathbb{R}^{P\times n} \tag{3.17}$$

where $\Phi[:,i](\mathbf{z})$ is the i^{th} column of $\boldsymbol{\Phi}(\mathbf{z})$. Note that the standard linear dynamics model $\mathbf{f}(\mathbf{z}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ falls under this model class by setting $\boldsymbol{\Phi}(\mathbf{z}) = \mathbf{I} \otimes \mathbf{z}$ and $\mathbf{w} = \text{vec}\big([\mathbf{A}\ \mathbf{B}]^\top\big)$. In a Bayesian framework a prior distribution is placed over the unknown parameter space



(a) Samples from the prior $p(\mathbf{f}|\boldsymbol{\theta})$. (b) Samples from the posterior $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$.

Figure 3.6: Samples from a prior and posterior distribution over quadratic functions $\mathbf{f}(\mathbf{z}) = \boldsymbol{\Phi}(\mathbf{z})^\top \mathbf{w}$ where $\boldsymbol{\phi}(\mathbf{z}) = \left[1; \mathbf{z}; \text{vec}(\mathbf{z}\mathbf{z}^\top)\right]$ with prior Gaussian distribution over the weights $p(\mathbf{w}|\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega})$. The red, black and dashed lines show the underlying function $\mathbf{f}$, mean of the distribution over functions (prior $\mathbf{m}$ and posterior $\mathbf{m}_+$) and sampled functions respectively. The grey regions denote two standard deviations, or 95% confidence region, of the distribution. Training data is shown as blue dots on the posterior plot.

$p(\mathbf{w}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\eta}, \boldsymbol{\Omega})$ which is defined in terms of the hyperparameters $\boldsymbol{\theta} = \{\boldsymbol{\eta}, \boldsymbol{\Omega}\}$ and represents any prior knowledge regarding their relationship.

This prior over parameters leads to a prior over the space of functions $p(\mathbf{f}|\boldsymbol{\theta})$ which satisfies the definition of a Gaussian process. The GP is parameterised by a *mean function* and *covariance function* (or *kernel*) defined to be

$$\mathbf{m}(\mathbf{z}) := \mathbb{E}_{\mathbf{w}}\left[\mathbf{f}(\mathbf{z})\right] \qquad = \boldsymbol{\Phi}(\mathbf{z})^\top \boldsymbol{\eta} \tag{3.18}$$

$$\mathbf{K}(\mathbf{z}, \tilde{\mathbf{z}}) := \text{cov}_{\mathbf{w}}\left[\mathbf{f}(\mathbf{z}), \mathbf{f}(\tilde{\mathbf{z}})\right] = \boldsymbol{\Phi}(\mathbf{z})^\top \boldsymbol{\Omega} \boldsymbol{\Phi}(\tilde{\mathbf{z}}) \tag{3.19}$$

respectively. These functions are also defined to take matrix inputs with dimensions determined through the way in which matrix inputs are handled by $\boldsymbol{\Phi}$. For example, if $\boldsymbol{\Phi}(\mathbf{z}) = \mathbf{I} \otimes \boldsymbol{\phi}(\mathbf{z})$ the covariance matrix $\mathbf{K}(\mathbf{Z}, \mathbf{Z})$ would be block diagonal, with one block per output dimension. For notational convenience we shall now define $\mathbf{K}_{nn} := \mathbf{K}(\mathbf{Z}, \mathbf{Z})$. The covariance function can also accept a single input by letting $\mathbf{K}(\mathbf{z}) := \mathbf{K}(\mathbf{z}, \mathbf{z})$. Zero mean priors $\boldsymbol{\eta} = \mathbf{0}$ shall be considered for the remainder of this discussion in order to keep the notation uncluttered, however extension to $\boldsymbol{\eta} \neq \mathbf{0}$ is trivial.

Posterior

The posterior over parameters given the observed data set $\mathcal{D}$ is computed using Bayes' rule

$$p(\mathbf{w}|\mathcal{D}, \boldsymbol{\theta}) = \frac{p(\mathbf{Y}|\mathbf{w}, \mathbf{Z}, \boldsymbol{\theta})}{p(\mathbf{Y}|\mathbf{Z}, \boldsymbol{\theta})} p(\mathbf{w}|\boldsymbol{\theta}) \tag{3.20}$$

where $p(\mathbf{Y}|\mathbf{w}, \mathbf{Z}, \boldsymbol{\theta})$ is the *likelihood* of the data given the parameters and $p(\mathbf{Y}|\mathbf{Z}, \boldsymbol{\theta})$ is the *marginal likelihood* (or *evidence*) since it is the likelihood of the data marginalised over the parameters

$p(\mathbf{Y}|\mathbf{Z},\boldsymbol{\theta}) = \int p(\mathbf{Y}|\mathbf{w},\mathbf{Z},\boldsymbol{\theta})p(\mathbf{w}|\boldsymbol{\theta})\mathrm{d}\mathbf{w}$. Note that the prior over parameters is independent of the training input set therefore $p(\mathbf{w}|\boldsymbol{\theta}) = p(\mathbf{w}|\mathbf{Z},\boldsymbol{\theta})$. Given that the likelihood and the prior are simply multivariate Gaussians, their product can be calculated analytically as

$$p(\mathbf{w}|\mathcal{D},\boldsymbol{\theta}) \propto p(\mathbf{Y}|\mathbf{w},\mathbf{Z},\boldsymbol{\theta})p(\mathbf{w}|\boldsymbol{\theta}) = \mathcal{N}\Big(\boldsymbol{\Phi}(\mathbf{Z})^\top\mathbf{w}, \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\otimes\mathbf{I}\Big)\mathcal{N}(\boldsymbol{\eta},\boldsymbol{\Omega}) \tag{3.21}$$

where the product is an unnormalised Gaussian which can be found using the identity given in Appendix A.1. It is assumed that the additive noise covariance $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$ is also a hyperparameter to be inferred. By normalising this product, the posterior over parameter space is given in closed form as $p(\mathbf{w}|\mathcal{D},\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\eta}_+,\boldsymbol{\Omega}_+)$ with mean and covariance

$$\boldsymbol{\eta}_+ = \Big(\boldsymbol{\Omega}^{-1} + \boldsymbol{\Phi}(\mathbf{Z})(\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\otimes\mathbf{I})^{-1}\boldsymbol{\Phi}(\mathbf{Z})^\top\Big)^{-1}\boldsymbol{\Phi}(\mathbf{Z})(\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\otimes\mathbf{I})^{-1}\vec{\mathbf{Y}} \tag{3.22}$$

$$\boldsymbol{\Omega}_+ = \Big(\boldsymbol{\Omega}^{-1} + \boldsymbol{\Phi}(\mathbf{Z})(\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\otimes\mathbf{I})^{-1}\boldsymbol{\Phi}(\mathbf{Z})^\top\Big)^{-1} \tag{3.23}$$

Since the distribution is Gaussian, the mean $\boldsymbol{\eta}_+ = \mathbb{E}[\mathbf{w}|\mathcal{D},\boldsymbol{\theta}]$ is the Maximum A Posteriori (MAP) estimate of the parameters $\mathbf{w}$. It follows that the posterior distribution over function space $\mathbf{f}|\mathcal{D} \sim \mathcal{GP}\big(\mathbf{m}_+,\mathbf{K}_+\big)$ is also a Gaussian process with mean function and covariance function

$$\mathbf{m}_+(\mathbf{z}) = \boldsymbol{\Phi}(\mathbf{z})^\top\boldsymbol{\eta}_+ \tag{3.24}$$

$$\mathbf{K}_+(\mathbf{z},\tilde{\mathbf{z}}) = \boldsymbol{\Phi}(\mathbf{z})^\top\boldsymbol{\Omega}_+\boldsymbol{\Phi}(\tilde{\mathbf{z}}) \tag{3.25}$$

An example of sampling from the prior and posterior of a single-input single-output quadratic function is shown in Fig.3.6. Prediction is then achieved simply as $p\big(\mathbf{f}(\mathbf{z})|\mathbf{z},\mathcal{D},\boldsymbol{\theta}\big) = \mathcal{N}\big(\mathbf{m}_+(\mathbf{z}),\mathbf{K}_+(\mathbf{z})\big)$.

It is instructive to note that as the prior becomes "flat", $\boldsymbol{\Omega} = \omega\mathbf{I}$ as $\omega \to \infty$, the posterior mean prediction tends to $\boldsymbol{\eta}_+ \to \big(\boldsymbol{\Phi}(\mathbf{Z})\boldsymbol{\Phi}(\mathbf{Z})^\top\big)^{-1}\boldsymbol{\Phi}(\mathbf{Z})\vec{\mathbf{Y}}$. This is the solution of the linear least squares problem $\boldsymbol{\Phi}(\mathbf{Z})^\top\mathbf{p} = \vec{\mathbf{Y}}$. A flat prior implies that the MAP estimate is the same as the Maximum Likelihood estimate since $p(\mathbf{w}|\mathcal{D},\boldsymbol{\theta}) \propto p(\mathbf{Y}|\mathbf{w},\mathbf{Z},\boldsymbol{\theta})$.

Kernel Trick

The expressions for the posterior mean and covariance in Eqs. (3.24)–(3.25) can be manipulated into an equivalent form given in Theorem 3.1. This is a well known result in the Machine Learning community and is based on *Mercer's Theorem*, see Mercer (1909). As will be shown in the following section, these are actually the predictive equations for a Gaussian process.

Theorem 3.1 (Kernel Trick). *The expressions for the posterior mean and covariance function in Eqs. (3.24)–(3.25) are equivalent to*

$$\mathbf{m}_+(\mathbf{z}) = \mathbf{K}(\mathbf{z},\mathbf{Z})\Big(\mathbf{K}_{nn} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\otimes\mathbf{I}\Big)^{-1}\vec{\mathbf{Y}} \tag{3.26}$$

$$\mathbf{K}_+(\mathbf{z},\tilde{\mathbf{z}}) = \mathbf{K}(\mathbf{z},\tilde{\mathbf{z}}) - \mathbf{K}(\mathbf{z},\mathbf{Z})\Big(\mathbf{K}_{nn} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\otimes\mathbf{I}\Big)^{-1}\mathbf{K}(\mathbf{Z},\tilde{\mathbf{z}}) \tag{3.27}$$

where the kernel matrices $\mathbf{K}$ *consist of inner products of the feature matrix* $\boldsymbol{\Phi}$ *with respect to the prior covariance* $\boldsymbol{\Omega}$ *as defined in Eq. (3.19).*

Proof. Let $\mathbf{A} = \boldsymbol{\Omega}$, $\mathbf{B} = \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}$ and $\mathbf{C} = \boldsymbol{\Phi}(\mathbf{Z})$. Then the parameter mean $\boldsymbol{\eta}_+$ from Eq. (3.22) can be written as

$$\begin{aligned}
\boldsymbol{\eta}_+ &= \big(\mathbf{A}^{-1} + \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^\top\big)^{-1}\mathbf{C}\mathbf{B}^{-1}\vec{\mathbf{Y}} \\
&= \mathbf{A}\mathbf{C}\big(\mathbf{C}^\top\mathbf{A}\mathbf{C} + \mathbf{B}\big)^{-1}\vec{\mathbf{Y}} \\
&= \boldsymbol{\Omega}\boldsymbol{\Phi}(\mathbf{Z})\Big(\boldsymbol{\Phi}(\mathbf{Z})^\top\boldsymbol{\Omega}\boldsymbol{\Phi}(\mathbf{Z}) + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}\Big)^{-1}\vec{\mathbf{Y}}
\end{aligned}$$

which follows from a variant of the matrix inversion lemma[1]. Therefore, recognising that $\mathbf{K}(\mathbf{z},\mathbf{Z}) = \boldsymbol{\Phi}(\mathbf{z})^\top\boldsymbol{\Omega}\boldsymbol{\Phi}(\mathbf{Z})$ and $\mathbf{K}(\mathbf{Z},\mathbf{Z}) = \boldsymbol{\Phi}(\mathbf{Z})^\top\boldsymbol{\Omega}\boldsymbol{\Phi}(\mathbf{Z})$, Eq. (3.26) is equivalent to Eq. (3.24). Using the matrix inversion lemma the parameter covariance from Eq. (3.23) can be written as

$$\begin{aligned}
\boldsymbol{\Omega}_+ &= \big(\mathbf{A}^{-1} + \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^\top\big)^{-1} \\
&= \mathbf{A} - \mathbf{A}\mathbf{C}\big(\mathbf{C}^\top\mathbf{A}\mathbf{C} + \mathbf{B}\big)^{-1}\mathbf{C}^\top\mathbf{A} \\
&= \boldsymbol{\Omega} - \boldsymbol{\Omega}\boldsymbol{\Phi}(\mathbf{Z})\Big(\boldsymbol{\Phi}(\mathbf{Z})^\top\boldsymbol{\Omega}\boldsymbol{\Phi}(\mathbf{Z}) + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}\Big)^{-1}\boldsymbol{\Phi}(\mathbf{Z})^\top\boldsymbol{\Omega}
\end{aligned}$$

and it is clear that Eq. (3.27) is equal to Eq. (3.25). ■

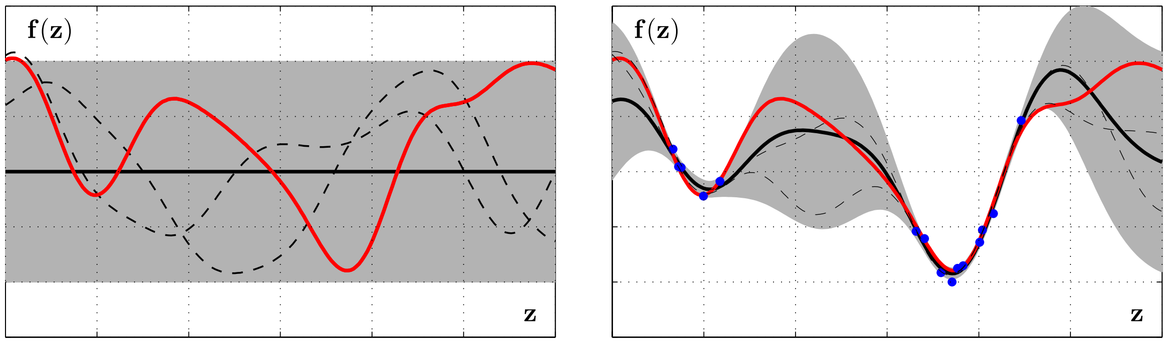
Note that the covariance function is defined in terms of an inner product with respect to the prior covariance $\boldsymbol{\Omega}$. An important result follows. As pointed out by Rasmussen & Williams (2006), if an algorithm is defined solely in terms of inner products in the input space then it can be lifted into the feature space through direct calculation of the covariance function. In other words, instead of defining a feature matrix $\boldsymbol{\Phi}(\mathbf{z})$ and determining the kernel $\mathbf{K}$ indirectly from this, the kernel could be defined directly. For a small set of features $P \ll n$ this is unnecessary and will increase computation since inversion of an n by n rather than a P by P matrix is required. However, if the feature space is large $P \gg n$ or even infinite, this formulation yields a tractable problem with the provision that the equivalent covariance function can be computed. This is known as the *kernel trick*. An equivalent way of viewing this reformulation can be found by considering placing a prior over functions directly instead of indirectly via the parameters.

3.4.3 Nonparametric Approach

Prior

In the context of the problem outlined in Sec. 3.4.1, the assumption is that $\mathbf{f}$ has been drawn from a Gaussian process prior $\mathbf{f}|\boldsymbol{\theta} \sim \mathcal{GP}\big(\mathbf{m},\mathbf{K}\big)$ with mean and covariance function defined

[1] $(\mathbf{P}^{-1} + \mathbf{R}\mathbf{Q}^{-1}\mathbf{R}^\top)^{-1}\mathbf{R}\mathbf{Q}^{-1} = \mathbf{P}\mathbf{R}^\top(\mathbf{R}^\top\mathbf{P}\mathbf{R} + \mathbf{Q})^{-1}$ for $\mathbf{P}, \mathbf{Q}$ positive definite.



(a) Samples from the GP prior $p(\mathbf{f}|\boldsymbol{\theta})$. (b) Samples from the GP posterior $p(\mathbf{f}|\mathcal{D},\boldsymbol{\theta})$.

Figure 3.7: Samples from a Gaussian process prior and posterior after data from the underlying function has been observed. The red, black and dashed lines show the underlying function $\mathbf{f}$, mean of the Gaussian process (prior $\mathbf{m}$ and posterior $\mathbf{m}_+$) and sampled functions respectively. The grey regions denote two standard deviations, or 95% confidence region, of the GP distribution. Training data is shown as blue dots on the posterior plot.

explicitly as

$$\mathbf{m}(\mathbf{z}) = \mathbb{E}_{\mathbf{f}}\big[\mathbf{f}(\mathbf{z})\big] \tag{3.28}$$

$$\mathbf{K}(\mathbf{z},\tilde{\mathbf{z}}) = \mathrm{cov}_{\mathbf{f}}\big[\mathbf{f}(\mathbf{z}),\mathbf{f}(\tilde{\mathbf{z}})\big] \tag{3.29}$$

where the expectation is now taken with respect to the random function $\mathbf{f}$ rather than the random parameters $\mathbf{w}$. These functions have some given structure, denoted by $\mathcal{H}$, which is parameterised by a set of hyperparameters $\boldsymbol{\theta}$. Again, for ease of notation and with little loss of generality, a zero mean prior $\mathbf{m}(\mathbf{z}) = \mathbf{0}$ shall be considered for the remainder of this section.

It can be proven through *Mercer's Theorem* that the positive semi-definite covariance function $\mathbf{K} : \mathbb{R}^E \times \mathbb{R}^E \to \mathbb{R}^{E\times E}$ will always have an (infinite) inner product representation in terms of eigenfunctions and associated eigenvalues. Therefore, from the previous section, any positive definite function $\mathbf{K}$ will represent a distribution over a function space spanned by an (infinite) sum of basis functions.

Posterior

Given the observed training data set $\mathcal{D}$ it is desirable to update this prior and form a posterior distribution over the space of functions. This posterior distribution is related to the prior and the training data through Bayes' rule

$$p(\mathbf{f}|\mathcal{D},\boldsymbol{\theta}) = \frac{p(\mathbf{Y}|\mathbf{f},\mathbf{Z},\boldsymbol{\theta})}{p(\mathbf{Y}|\mathbf{Z},\boldsymbol{\theta})}p(\mathbf{f}|\boldsymbol{\theta}) \tag{3.30}$$

with likelihood $p(\mathbf{Y}|\mathbf{f},\mathbf{Z},\boldsymbol{\theta})$ and prior $p(\mathbf{f}|\boldsymbol{\theta}) = p(\mathbf{f}|\mathbf{Z},\boldsymbol{\theta})$ which is independent of the training inputs. The marginal likelihood term is given by the following equivalent integrals

$$p(\mathbf{Y}|\mathbf{Z},\boldsymbol{\theta}) = \int p(\mathbf{Y}|\mathbf{f},\mathbf{Z},\boldsymbol{\theta})p(\mathbf{f}|\boldsymbol{\theta})\mathrm{d}\mathbf{f} = \int p(\mathbf{Y}|\mathbf{F},\boldsymbol{\theta})p(\mathbf{F}|\mathbf{Z},\boldsymbol{\theta})\mathrm{d}\vec{\mathbf{F}} \tag{3.31}$$

These expressions are equivalent because the observed data set $\mathbf{Y}$ is related to the underlying function $\mathbf{f}$ only through the underlying values $\mathbf{F}$ since the likelihood has no notion of the prior beliefs. We shall discuss the importance of this term for training the hyperparameters of a Gaussian process in the following section. Note that the dependence on the structure $\mathcal{H}$ is assumed implicitly through the hyperparameters. It turns out that this posterior distribution over functions is also a Gaussian process and can be calculated analytically as demonstrated in Theorem 3.2. This is again a standard result from the Machine Learning community.

Theorem 3.2 (Gaussian Process Posterior). *Assume that the function* $\mathbf{f}$ *has been drawn from some Gaussian process prior* $\mathbf{f}|\boldsymbol{\theta} \sim \mathcal{GP}(\mathbf{0},\mathbf{K})$ *with hyperparameters* $\boldsymbol{\theta}$ *and that the data set* $\mathcal{D}$ *given by Eq. (3.14) has been observed. The posterior distribution is also a Gaussian process* $\mathbf{f}|\mathcal{D},\boldsymbol{\theta} \sim \mathcal{GP}(\mathbf{m}_+,\mathbf{K}_+)$ *with mean and covariance function*

$$\mathbf{m}_+(\mathbf{z}) = \mathbf{K}(\mathbf{z},\mathbf{Z})\big(\mathbf{K}_{nn} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}\big)^{-1}\vec{\mathbf{Y}} \tag{3.32}$$

$$\mathbf{K}_+(\mathbf{z},\tilde{\mathbf{z}}) = \mathbf{K}(\mathbf{z},\tilde{\mathbf{z}}) - \mathbf{K}(\mathbf{z},\mathbf{Z})\big(\mathbf{K}_{nn} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}\big)^{-1}\mathbf{K}(\mathbf{Z},\tilde{\mathbf{z}}) \tag{3.33}$$

Proof. Consider the joint distribution of the noisy observed states $\mathbf{Y}$ and the function values $\mathbf{f}(\mathbf{z})$ and $\mathbf{f}(\tilde{\mathbf{z}})$ to be inferred

$$\left.\begin{bmatrix}\vec{\mathbf{Y}}\\ \mathbf{f}(\mathbf{z})\\ \mathbf{f}(\tilde{\mathbf{z}})\end{bmatrix}\right|\mathbf{z},\tilde{\mathbf{z}},\mathbf{Z},\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0},\begin{bmatrix}\mathbf{K}_{nn} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I} & \mathbf{K}(\mathbf{Z},\mathbf{z}) & \mathbf{K}(\mathbf{Z},\tilde{\mathbf{z}})\\ \mathbf{K}(\mathbf{z},\mathbf{Z}) & \mathbf{K}(\mathbf{z},\mathbf{z}) & \mathbf{K}(\mathbf{z},\tilde{\mathbf{z}})\\ \mathbf{K}(\tilde{\mathbf{z}},\mathbf{Z}) & \mathbf{K}(\tilde{\mathbf{z}},\mathbf{z}) & \mathbf{K}(\tilde{\mathbf{z}},\tilde{\mathbf{z}})\end{bmatrix}\right) \tag{3.34}$$

By conditioning on the observed outputs $\mathbf{Y}$ using the identity for a Gaussian conditional distribution, from Appendix A.1, the result follows. ■

It is clear that this predictive form is exactly the same as was obtained in Eqs. (3.26)–(3.27) by following a parametric approach and extending to possibly infinite dimensional feature spaces using the kernel trick. However in this case the covariance function has been defined directly and there is no need to appeal to a parametric interpretation in terms of a feature vector or a finite set of basis functions. Note that if the covariance function is in the form of a radial basis function, the predictive mean is in the form of a linear sum radial basis functions each centred on a training data point $\mathbf{m}_+(\mathbf{z}) = \mathbf{K}(\mathbf{z},\mathbf{Z})\boldsymbol{\beta}$ where $\boldsymbol{\beta} = \big(\mathbf{K}_{nn} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}\big)^{-1}\vec{\mathbf{Y}}$.

The process of choosing a prior distribution over function space and forming the posterior based on training data is shown in Fig. 3.7. The posterior shown in Fig. 3.7(b) clearly shows that in regions where there is a lot of training data the uncertainty in functional predictions reduces

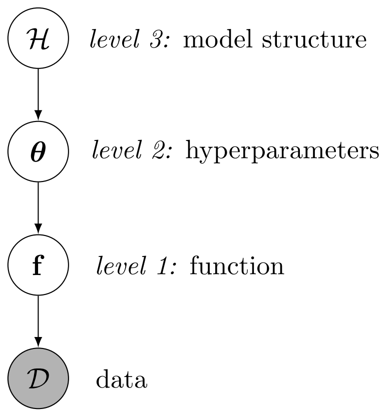


Figure 3.8: Graphical model depicting the model selection hierarchy from possible model structures $\mathcal{H}$ to the associated hyperparameters $\boldsymbol{\theta}$ which in turn control the distribution over the actual underlying function $\mathbf{f}$ which determines the observed data set $\mathcal{D}$. The clear nodes depict hidden variables and the gray nodes depict observed variables.

significantly whereas in regions with little training data the distribution collapses back to the prior. Therefore, given an arbitrary state-action pair $\mathbf{z}_k$, training data set $\mathcal{D}$ and Gaussian process prior over function space, the predictive distribution over the next state $p(\mathbf{x}_{k+1}|\mathbf{z}_k, \mathcal{D})$ can be obtained analytically.

3.4.4 Model Selection and Training

The preceding section laid out the framework for prediction using a Gaussian process prior with known structural form $\mathcal{H}$ and hyperparameters $\boldsymbol{\theta}$. In practice, the form and parameters governing the prior are unknown, and are in fact a construct that has been created in order to view the inference problem from a Bayesian perspective. A fully Bayesian approach to addressing the model selection problem, as outlined in Rasmussen & Williams (2006) Chapter 5, can be viewed through the hierarchy depicted by the graphical model in Fig. 3.8. In this framework a distribution is maintained over hyperparameters and possible model structures as well as the function space itself.

In this thesis, we shall not adopt the fully Bayesian treatment of model selection due to the analytic intractability of integrating over hyperparameter distributions and computational complexity of approximating it using sampling methods. We will assume that the model structure $\mathcal{H}$ has been chosen beforehand based on intuition about the problem at hand and that a point estimate of the hyperparameters $\hat{\boldsymbol{\theta}}$ is chosen based on an evidence maximisation scheme which will now be motivated.

Level 1 inference was discussed in the previous section and is concerned with determining the posterior over function space $p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta})$ once a training data set had been observed and given some known hyperparameters. This was achieved using Bayes' rule in Eq. (3.30) and it turned out that this could be found analytically in the form of Eqs. (3.32)–(3.33).

Level 2 inference considers the distribution over hyperparameters $\boldsymbol{\theta}$ given a specific model structure $\mathcal{H}$. The posterior after observing a training data set is again given by Bayes' rule as

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathbf{Y}|\mathbf{Z},\boldsymbol{\theta})}{p(\mathbf{Y}|\mathbf{Z})} p(\boldsymbol{\theta}) \tag{3.35}$$

Again, dependence of each term on the model structure is taken implicitly. Now a sensible point estimate for the hyperparameters would be the Maximum A Posteriori (MAP) estimate $\hat{\boldsymbol{\theta}} \in \operatorname{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D})$. Taking a flat prior over hyperparameters means that $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathbf{Y}|\mathbf{Z},\boldsymbol{\theta})$. Therefore maximising the marginal likelihood (or log-marginal likelihood) from Eq. (3.30) is equivalent to the MAP estimate of the hyperparameters. This can be written analytically as

$$-\log p(\mathbf{Y}|\mathbf{Z},\boldsymbol{\theta}) = \underbrace{\tfrac{1}{2}\big(\vec{\mathbf{Y}} - \mathbf{m}(\mathbf{Z})\big)^\top \mathbf{K}_{\boldsymbol{\epsilon}}^{-1}\big(\vec{\mathbf{Y}} - \mathbf{m}(\mathbf{Z})\big)}_{\text{data fit}} + \underbrace{\tfrac{1}{2}\log\big|\mathbf{K}_{\boldsymbol{\epsilon}}\big|}_{\text{complexity}} + \tfrac{En}{2}\log 2\pi \tag{3.36}$$

where $\mathbf{z} \in \mathbb{R}^D$, $\mathbf{K}_{\boldsymbol{\epsilon}} = \mathbf{K}_{nn} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}$ since the likelihood $p(\mathbf{Y}|\mathbf{F},\mathbf{Z},\boldsymbol{\theta}) = \mathcal{N}(\vec{\mathbf{F}}, \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I})$ and prior $p(\mathbf{F}|\mathbf{Z},\boldsymbol{\theta}) = \mathcal{N}\big(\mathbf{m}(\mathbf{Z}), \mathbf{K}_{nn}\big)$ are simply multivariate Gaussians.

As highlighted in Chapter 5.4 of Rasmussen & Williams (2006), maximising the marginal likelihood leads to some intuitive properties in terms of the trade off between data fit and model complexity. In non-rigorous terms we consider the complexity of a given model to be how quickly the function varies as we move around the $\mathbf{z}$ space. If the function changes very rapidly then data observed in one region of the $\mathbf{z}$ space will only be useful for making predictions very close to that region (an exception to this would be if we considered periodic covariance functions). Conversely, if the variations are slow then data observed in one region of the space may be generalised to make predictions in the surrounding regions.

We shall now provide a crude argument for why the log likelihood in Eq. (3.36) trades off data fit with complexity. Consider a zero mean prior with squared exponential covariance term. As the length scales are increased, the terms in $\mathbf{K}_{\boldsymbol{\epsilon}}$ become more correlated and therefore the complexity term $\big|\mathbf{K}_{\boldsymbol{\epsilon}}\big|$ decreases. However at the same time the "size" of the inverse $\mathbf{K}_{\boldsymbol{\epsilon}}^{-1}$ increases and therefore deviations from the prior mean are penalised more heavily. If we maximise the marginal likelihood, we are looking for *the least complex model that explains the data.*

3.4.5 Sparse Approximations

One of the greatest problems in employing a full Gaussian process model is the computational requirements required as the data set increases. The source of this burden lies mainly in the inversion of the kernel matrix $\mathbf{K}_{nn}$ and its multiplication with vectors. This issue can be addressed by finding low-rank approximations to this matrix and most of these methods can be viewed under the unifying framework of Quiñonerno Candela & Rasmussen (2005). In this thesis we shall consider the Fully Independent Training Conditional (FITC) approximation (or sparse Gaussian processes using pseudo-inputs) defined by Snelson & Gharamani (2006). We find this to be a more elegant approach than simply throwing away a subset of the data points.

The FITC approximation assumes that a fictitious (or pseudo) data set $\mathcal{D}_\mathrm{p} = \{\mathbf{Z}_\mathrm{p}, \mathbf{F}_\mathrm{p}\}$ of $m < n$ data points is available alongside the actual data $\mathcal{D}$. Note that it would be meaningless to introduce noise onto the pseudo-targets $\mathbf{F}_\mathrm{p}$ and use $\mathbf{Y}_\mathrm{p}$ instead. It turns out that we do not actually need to define values for $\mathbf{F}_\mathrm{p}$ since their effect can be integrated out analytically

$$\begin{aligned} p(\mathbf{Y}|\mathbf{Z}, \mathbf{Z}_\mathrm{p}) &= \int p(\mathbf{Y}|\mathbf{Z}, \mathbf{Z}_\mathrm{p}, \mathbf{F}_\mathrm{p}) p(\mathbf{F}_\mathrm{p}|\mathbf{Z}_\mathrm{p}) \mathrm{d}\vec{\mathbf{F}}_\mathrm{p} \\ &= \int \mathcal{N}\big(\vec{\mathbf{Y}}|\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\vec{\mathbf{F}}_\mathrm{p}, \boldsymbol{\Gamma}\big)\mathcal{N}\big(\vec{\mathbf{F}}_\mathrm{p}|\mathbf{0}, \mathbf{K}_{mm}\big)\mathrm{d}\vec{\mathbf{F}}_\mathrm{p} \\ &= \mathcal{N}\big(\vec{\mathbf{Y}}|\mathbf{0}, \mathbf{Q}_{nn} + \boldsymbol{\Gamma}\big) \end{aligned}$$

where $\mathbf{K}_{nm} := \mathbf{K}(\mathbf{Z}, \mathbf{Z}_\mathrm{p})$, $\mathbf{K}_{mm} := \mathbf{K}(\mathbf{Z}_\mathrm{p}, \mathbf{Z}_\mathrm{p})$ and $\mathbf{Q}_{nn} := \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn}$ is clearly a low rank approximation of $\mathbf{K}_{nn}$. The matrix $\boldsymbol{\Gamma}$ is specific to the FITC approximation and is defined as

$$\boldsymbol{\Gamma} := \mathrm{diag}\{\mathbf{K}_{nn} - \mathbf{Q}_{nn}\} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}$$

which is distinct from many of the other sparse approximate methods which simply use $\boldsymbol{\Gamma} = \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I}$. When training, the locations of these pseudo-inputs $\mathbf{Z}_\mathrm{p}$ are optimised along with the hyperparameters $\boldsymbol{\theta}$.

Now to determine the Gaussian process posterior prediction under this approximation. The posterior mean and covariance are now defined as

$$\mathbf{m}_+(\mathbf{z}) = \mathbf{K}(\mathbf{z}, \mathbf{Z}_\mathrm{p})\mathbf{B}^{-1}\mathbf{K}_{mn}\boldsymbol{\Gamma}^{-1}\vec{\mathbf{Y}} \tag{3.37}$$

$$\mathbf{K}_+(\mathbf{z}, \tilde{\mathbf{z}}) = \mathbf{K}(\mathbf{z}, \tilde{\mathbf{z}}) - \mathbf{K}(\mathbf{z}, \mathbf{Z}_\mathrm{p})\big(\mathbf{K}_{mm}^{-1} - \mathbf{B}^{-1}\big)\mathbf{K}(\mathbf{Z}_\mathrm{p}, \tilde{\mathbf{z}}) \tag{3.38}$$

where $\mathbf{B} := \mathbf{K}_{mm} + \mathbf{K}_{mn}\boldsymbol{\Gamma}^{-1}\mathbf{K}_{nm}$. It is important to note that these equations are in exactly the same form as Eqs. (3.32)–(3.33) with respect to the test inputs $\mathbf{z}$ and $\tilde{\mathbf{z}}$. This means that employing the FITC sparse approximation will have no effect on our discussion of multiple-step ahead predictions in Sec. 3.5.

3.4.6 Priors for General Nonlinear Functions

Squared Exponential Kernel

Attention shall now be given to a particular choice of kernel $\mathbf{K}$, the Squared Exponential (SE). To motivate the use of this kernel, consider the single state case where $E = 1$. The analysis in the section closely follows that given by Deisenroth (2009) in Section 2.2. The standard form of the SE kernel is

$$k_{\mathrm{SE}}(\mathbf{z}, \tilde{\mathbf{z}}) = \alpha^2 \exp\Big(- \tfrac{1}{2}(\mathbf{z} - \tilde{\mathbf{z}})^\top \boldsymbol{\Lambda}^{-1}(\mathbf{z} - \tilde{\mathbf{z}})\Big) \tag{3.39}$$

with hyperparameters $\boldsymbol{\theta} \in \mathbb{R}^{D+1}$ consisting of an output variance α^2 and a diagonal matrix of input length-scales $\boldsymbol{\Lambda} = \mathrm{diag}\big\{[\lambda_1^2 \ldots \lambda_D^2]\big\}$. Theorem 3.3 shows that this kernel defines a prior distribution over the space of all functions in the class of a given universal function approximator.

Theorem 3.3 (Squared Exponential Kernel). *Consider the function $f : \mathbb{R}^D \to \mathbb{R}$ parameterised by a universal function approximator of the form*

$$f(\mathbf{z}) = \int_{\mathbb{R}^D} \phi(\mathbf{z}, \mathbf{s})\gamma(\mathbf{s})\mathrm{d}\mathbf{s} = \int_{\mathbb{R}^D} \exp\Big(-(\mathbf{z}-\mathbf{s})^\top \mathbf{\Lambda}^{-1}(\mathbf{z}-\mathbf{s})\Big)\gamma(\mathbf{s})\mathrm{d}\mathbf{s} \tag{3.40}$$

where $\gamma(\mathbf{s})$ is a weighting function. Assuming a normal Gaussian prior over the weights $\gamma(\mathbf{s}) \sim \mathcal{N}(0,1)$ is equivalent to assuming $f \sim \mathcal{GP}(0, k_{SE})$ has been drawn from a Gaussian process with zero mean and squared exponential covariance function given in Eq. (3.39).

Proof. The equivalent Gaussian process representation of this problem will have a mean function given by $\mathbb{E}_\gamma[f(\mathbf{z})]$ and covariance function given by $\mathrm{cov}_\gamma[f(\mathbf{z}), f(\tilde{\mathbf{z}})]$. The mean is calculated as

$$\mathbb{E}_\gamma[f(\mathbf{z})] = \int f(\mathbf{z})p\big(\gamma(\mathbf{s})\big)\mathrm{d}\gamma(\mathbf{s}) = \int \phi(\mathbf{z},\mathbf{s})\bigg(\int \gamma(\mathbf{s})p\big(\gamma(\mathbf{s})\big)\mathrm{d}\gamma(\mathbf{s})\bigg)\mathrm{d}\mathbf{s} = 0$$

since $\mathbb{E}_\gamma[\gamma(\mathbf{s})] = 0$. Now, because the prior mean function equals zero the prior covariance is

$$\begin{aligned}\mathrm{cov}_\gamma[f(\mathbf{z}), f(\tilde{\mathbf{z}})] &= \mathbb{E}_\gamma[f(\mathbf{z})f(\tilde{\mathbf{z}})] \\ &= \int f(\mathbf{z})f(\tilde{\mathbf{z}})p\big(\gamma(\mathbf{s})\big)\mathrm{d}\gamma(\mathbf{s}) \\ &= \int \phi(\mathbf{z},\mathbf{s})\phi(\tilde{\mathbf{z}},\mathbf{s})\bigg(\int \gamma(\mathbf{s})^2 p\big(\gamma(\mathbf{s})\big)\mathrm{d}\gamma(\mathbf{s})\bigg)\mathrm{d}\mathbf{s} \\ &= \int \phi(\mathbf{z},\mathbf{s})\phi(\tilde{\mathbf{z}},\mathbf{s})\mathrm{d}\mathbf{s}\end{aligned}$$

since $\mathbb{E}_\gamma[\gamma(\mathbf{s})^2] = 1$. Now because ϕ produces an unnormalised Gaussian this integral is tractable and from Eq. (3.40)

$$\mathrm{cov}_\gamma[f(\mathbf{z}), f(\tilde{\mathbf{z}})] = \alpha^2 \exp\Big(-\tfrac{1}{2}(\mathbf{z}-\tilde{\mathbf{z}})^\top \mathbf{\Lambda}^{-1}(\mathbf{z}-\tilde{\mathbf{z}})\Big)$$

for a suitable α. This is exactly the form of the SE kernel in Eq. (3.39). ■

Informally, the SE covariance function provides a means of placing a prior over the space of all smooth, or infinitely differentiable functions, with intrinsic input length scales.

Additive Squared Exponential Kernel

A related class of kernel functions is the additive squared exponential family. These are useful in the situation where we wish to parameterise a distribution over a space of additively separable functions. The family of first-order additive functions are as follows

$$\mathbf{f}(\mathbf{z}) = \sum_{i=1}^{D} \mathbf{f}_i\big(z[i]\big)$$

We may also wish to consider higher order interaction terms. We shall consider a general additive form in which the orders of interaction are stored in the set $\mathcal{I}$ and can be written mathematically as

$$\mathbf{f}(\mathbf{z}) = \sum_{\mathbf{i}\in\mathcal{I}} \mathbf{f}_{\mathbf{i}}\big(\mathbf{z}[\mathbf{i}]\big) \tag{3.41}$$

For example if we know our function is an additive combination of nonlinear functions with respect to the first, third and a combination of the second and third components of the state-action space then $\mathcal{I} = \{1, 3, [2; 3]\}$. Considering D^{th} order interactions $\mathcal{I} = \{[1 \ldots D]\}$ will obviously brings us back to the completely general function $\mathbf{f}(\mathbf{z})$. Now a kernel that parameterises a prior distribution over such a space is the additive squared exponential which we define as follows

$$k_{\text{aSE}}(\mathbf{z}, \tilde{\mathbf{z}}) = \sum_{\mathbf{i}\in\mathcal{I}} \alpha_{\mathbf{i}}^2 \exp\Big(-\tfrac{1}{2}\big(\mathbf{z}[\mathbf{i}] - \tilde{\mathbf{z}}[\mathbf{i}]\big)^\top \boldsymbol{\Lambda}_{\mathbf{i}}^{-1}\big(\mathbf{z}[\mathbf{i}] - \tilde{\mathbf{z}}[\mathbf{i}]\big)\Big) \tag{3.42}$$

where it is clear that setting $\mathcal{I} = \{[1 \ldots D]^\top\}$ recovers the standard squared exponential kernel.

In the literature the first order additive structure appears under Generalized Additive Models (GAMs) as introduced by Hastie & Tibshirani (1990). The general kernel involving all possible interactions was recently introduced by Duvenaud *et al.* (2011). The authors attack the main drawback this general kernel, which is the computational effort required to calculate all the terms, which scales as $\mathcal{O}(2^n)$. They present a neat iterative method of evaluating all the additive kernels in $\mathcal{O}(D^2)$ making it a useful and computationally tractable model to work with. This method requires that the $\alpha_{\mathbf{i}}$ and $\boldsymbol{\Lambda}_{\mathbf{i}}$ parameters are common for each order of interaction, which is not too restrictive. It is based on the Newton-Girard formulae, which assumes that the function can be decomposed as a product

$$k_{\text{aSE}}(\mathbf{z}, \tilde{\mathbf{z}}) = \sum_{\mathbf{i}\in\mathcal{I}} \alpha_{\mathbf{i}}^2 \prod_{i\in\mathbf{i}} \exp\Big(-\tfrac{1}{2}\big(z[i] - \tilde{z}[i]\big)^2/\lambda_i^2\Big)$$

As we shall see in Sec. 3.5 this decomposable structure is lost when we are considering prediction at an uncertain state-input and therefore this iterative method cannot be used. Therefore for most applications we will only be considering separating first order, and maybe second order, interactions and leaving higher order ones to be explained by a standard SE kernel.

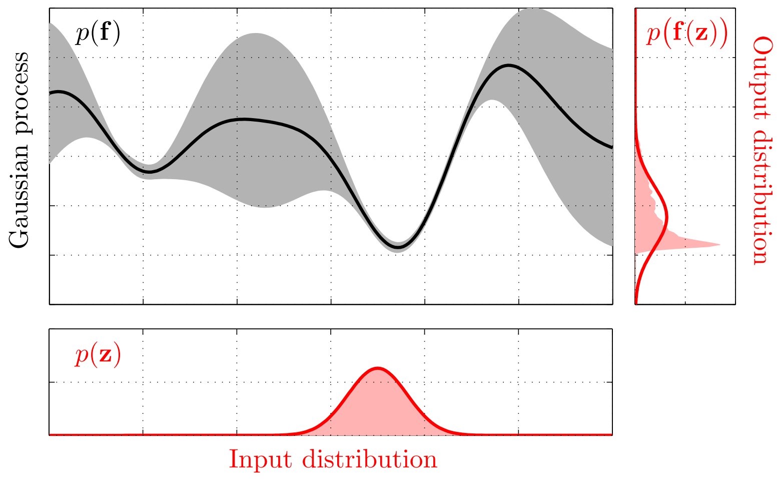


Figure 3.9: The moment matching approximation for a Gaussian process evaluated at a distribution over state-actions $p(\mathbf{z})$. The red lines depict the Gaussian approximations to the true distributions given by the red shaded areas.

3.5 Multiple-Step Predictions

3.5.1 Overview

A framework for maintaining a Gaussian distribution over whole trajectories $\boldsymbol{\tau}$ based on a moment matching approximation was outlined in Sec. 3.3. In order for a Gaussian process model to fit into this framework the mean $\mathbf{m}$ and covariance function $\mathbf{K}$ must be of a form such that the expectation $\boldsymbol{\mu}_* = \mathbb{E}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})]$ and covariance $\boldsymbol{\Sigma}_* = \mathrm{cov}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})]$ are analytically tractable given $\mathbf{f} \sim \mathcal{GP}(\mathbf{m}_+, \mathbf{K}_+)$ and $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The posterior mean $\mathbf{m}_+$ and covariance $\mathbf{K}_+$ are related to the input mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ through the GP predictive equations given in Eqs. (3.32)–(3.33). Through separation of the integrals with respect to the random function $\mathbf{f}$ and the input $\mathbf{z}$ the expectation, covariance and output-input covariance can be broken down as follows

$$\boldsymbol{\mu}_* = \mathbb{E}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})] \quad = \mathbb{E}_{\mathbf{z}}[\mathbf{m}_+(\mathbf{z})] \tag{3.43}$$

$$\boldsymbol{\Sigma}_* = \mathrm{cov}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})] = \mathbb{E}_{\mathbf{z}}\left[\mathbf{K}_+(\mathbf{z}) + \mathbf{m}_+(\mathbf{z})\mathbf{m}_+(\mathbf{z})^\top\right] - \boldsymbol{\mu}_*\boldsymbol{\mu}_*^\top \tag{3.44}$$

An example of the moment matching approximation for the posterior GP shown in Fig. 3.7(b) is given in Fig. 3.9. The exact expressions for these equations shall now be derived for a linear mean function along with a linear, squared exponential or additive squared exponential covariance function.

3.5.2 Parametric Form

General Basis Functions

We shall first consider the predictive equations $\mathbf{m}_+(\mathbf{z})$ and $\mathbf{K}_+(\mathbf{z})$ for a parametric model consisting of a linear sum of known basis functions as given in Eqs. (3.24)–(3.25). Remember that this approach can be viewed from a function space perspective as a Gaussian process for which there exists a finite inner product representation of the covariance function $\mathbf{K}(\mathbf{z}, \tilde{\mathbf{z}}) = \mathbf{\Phi}(\mathbf{z})^\top \mathbf{\Omega} \mathbf{\Phi}(\tilde{\mathbf{z}})$. The elements of the terms in Eqs. (3.43)–(3.44) are given by

$$\begin{aligned} \mu_*[i] &= \mathbb{E}_{\mathbf{z}}\big[\boldsymbol{\phi}_i\big]^\top \boldsymbol{\eta}_+ \\ \Sigma_*[i,j] &= \mathrm{tr}\Big(\mathbb{E}_{\mathbf{z}}\big[\boldsymbol{\phi}_i \boldsymbol{\phi}_j^\top\big]\big(\mathbf{\Omega}_+ + \boldsymbol{\eta}_+ \boldsymbol{\eta}_+^\top\big)\Big) - \mu_*[i]\mu_*[j] \end{aligned}$$

for $i, j \in \mathbb{Z}_{[1,E]}$ where $\boldsymbol{\phi}_i = \Phi[:, i](\mathbf{z})$. With the use of the Kronecker and Hadammard products we can actually define an expression for the full mean $\boldsymbol{\mu}_*$ and covariance $\mathbf{\Sigma}_*$ as follows

$$\boldsymbol{\mu}_* = (\mathbf{I}_E \otimes \mathbf{1}_P^\top)\Big(\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{\Phi}}\big] \circ \big(\mathbf{1}_E \otimes \boldsymbol{\eta}\big)\Big) \tag{3.45}$$

$$\mathbf{\Sigma}_* = (\mathbf{I}_E \otimes \mathbf{1}_P^\top)\Big(\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{\Phi}}\vec{\mathbf{\Phi}}^\top\big] \circ \big(\mathbf{1}_{E\times E} \otimes (\mathbf{\Omega} + \boldsymbol{\eta}\boldsymbol{\eta}^\top)\big)\Big)(\mathbf{I}_E \otimes \mathbf{1}_P) - \boldsymbol{\mu}_*\boldsymbol{\mu}_*^\top \tag{3.46}$$

where $\vec{\mathbf{\Phi}} = \mathrm{vec}\big(\mathbf{\Phi}(\mathbf{z})\big)$. Note that the $(\mathbf{I}_E \otimes \mathbf{1}_P)$ terms act as a kind of multivariable trace operator in the expression for $\mathbf{\Sigma}_*$. We could not find this form anywhere in the literature and therefore the derivation of it is our own. This expression means that the only requirement on the feature matrix is that the expectations $\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{\Phi}}\big]$ and $\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{\Phi}}\vec{\mathbf{\Phi}}^\top\big]$ can be evaluated analytically.

Linear Model

For the standard linear model the feature matrix is given by $\mathbf{\Phi}(\mathbf{z}) = \mathbf{I} \otimes \mathbf{z}$, Eqs. (3.45)–(3.46) take the following simplified form

$$\boldsymbol{\mu}_* = (\mathbf{I}_E \otimes \mathbf{1}_D^\top)\Big(\big(\mathbf{1}_E \otimes \mathbb{E}_{\mathbf{z}}[\mathbf{z}]\big) \circ \boldsymbol{\eta}\Big) \tag{3.47}$$

$$\mathbf{\Sigma}_* = (\mathbf{I}_E \otimes \mathbf{1}_D^\top)\Big(\big(\mathbf{1}_{E\times E} \otimes \mathbb{E}_{\mathbf{z}}\big[\mathbf{z}\mathbf{z}^\top\big]\big) \circ \big(\mathbf{\Omega} + \boldsymbol{\eta}\boldsymbol{\eta}^\top\big)\Big)(\mathbf{I}_E \otimes \mathbf{1}_D) - \boldsymbol{\mu}_*\boldsymbol{\mu}_*^\top \tag{3.48}$$

3.5.3 Nonparametric Form

General Covariance Function

We now turn our attention back to the more general nonparametric form of these equations. Consider the standard Gaussian process predictive equations given in Eqs. (3.32)–(3.33). The elements of the predictive equations in Eqs. (3.43)–(3.44) are given by

$$\begin{aligned}\mu_*[i] &= \mathbb{E}_{\mathbf{z}}\big[\mathbf{k}_i\big]^\top \boldsymbol{\beta} \\ \Sigma_*[i,j] &= \mathbb{E}_{\mathbf{z}}\big[k_{ij}\big] + \operatorname{tr}\Big(\mathbb{E}_{\mathbf{z}}\big[\mathbf{k}_i\mathbf{k}_j^\top\big]\big(\mathbf{K}_{\boldsymbol{\epsilon}}^{-1} + \boldsymbol{\beta}\boldsymbol{\beta}^\top\big)\Big) - \mu_*^{(i)}\mu_*^{(j)}\end{aligned}$$

for $i, j \in \mathbb{Z}_{[1,E]}$ where $\mathbf{k}_i = K[:,i](\mathbf{Z},\mathbf{z})$ is the i^{th} column of $\mathbf{K}(\mathbf{Z},\mathbf{z})$ and $k_{ij} = K[i,j](\mathbf{z})$ is the $(i,j)^{\text{th}}$ element of $\mathbf{K}(\mathbf{z})$. Note that if we are employing a sparse GP approximation then we simply replace the vector $\boldsymbol{\beta}$ and matrix $\mathbf{K}_{\boldsymbol{\epsilon}}$ with the appropriate terms and the rest of the analysis remains the same. Now, as with the parametric form, expressions may be written for the full mean and covariance as follows

$$\boldsymbol{\mu}_* = (\mathbf{I}_E \otimes \mathbf{1}_n^\top)\Big(\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{K}}\big] \circ \big(\mathbf{1}_E \otimes \boldsymbol{\beta}\big)\Big) \tag{3.49}$$

$$\boldsymbol{\Sigma}_* = \mathbb{E}_{\mathbf{z}}[\mathbf{K}(\mathbf{z})] + (\mathbf{I}_E \otimes \mathbf{1}_n^\top)\Big(\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{K}}\vec{\mathbf{K}}^\top\big] \circ \big(\mathbf{1}_{E\times E} \otimes (\mathbf{K}_{\boldsymbol{\epsilon}}^{-1} + \boldsymbol{\beta}\boldsymbol{\beta}^\top)\big)\Big)(\mathbf{I}_E \otimes \mathbf{1}_P) - \boldsymbol{\mu}_*\boldsymbol{\mu}_*^\top \tag{3.50}$$

where $\vec{\mathbf{K}} = \operatorname{vec}\big(\mathbf{K}(\mathbf{Z},\mathbf{z})\big)$. Again, this is a novel parameterisation that has not been found in the literature. Similarly to the parametric form, the only requirements on the covariance function is that the expectations $\mathbb{E}_{\mathbf{z}}\big[\mathbf{K}(\mathbf{z})\big]$, $\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{K}}\big]$ and $\mathbb{E}_{\mathbf{z}}\big[\vec{\mathbf{K}}\vec{\mathbf{K}}^\top\big]$ are analytically tractable. In most cases, stationary covariance functions will be considered i.e. $\mathbf{K}(\mathbf{z},\tilde{\mathbf{z}}) = \mathbf{K}(\mathbf{z}-\tilde{\mathbf{z}})$ therefore $\mathbf{K}(\mathbf{z})$ will be a constant matrix. The remaining expectations will be made up of the building blocks

$$\mathbb{E}_{\mathbf{z}}\big[k_a(\mathbf{a},\mathbf{z})\big] \quad \text{and} \quad \mathbb{E}_{\mathbf{z}}\big[k_a(\mathbf{a},\mathbf{z})k_b(\mathbf{b},\mathbf{z})\big] \tag{3.51}$$

where $k_a(\mathbf{a},\mathbf{z})$ and $k_b(\mathbf{b},\mathbf{z})$ are elements of $\mathbf{K}(\mathbf{a},\mathbf{z})$ and $\mathbf{K}(\mathbf{b},\mathbf{z})$ respectively and $\mathbf{a},\mathbf{b} \in \mathbf{Z}$. We will now evaluate these expectations for the squared exponential kernel.

Squared-Exponential Kernel

The following derivations can be found in Deisenroth (2009), Section 2.3 or Girard *et al.* (2003) for the univariate case. Consider the squared exponential kernel defined in Eq. (3.39). The first expectation in Eq. (3.51) can then be expanded as

$$\begin{aligned}\mathbb{E}_{\mathbf{z}}\big[k_a(\mathbf{a},\mathbf{z})\big] &= \int k_a(\mathbf{a},\mathbf{z})\mathcal{N}(\mathbf{z}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\mathbf{z} \\ &= \alpha_a^2(2\pi)^{D/2}|\boldsymbol{\Lambda}_a|^{1/2}\int \mathcal{N}(\mathbf{z}|\mathbf{a},\boldsymbol{\Lambda}_a)\mathcal{N}(\mathbf{z}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\mathbf{z} \\ &= \alpha_a^2(2\pi)^{D/2}|\boldsymbol{\Lambda}_a|^{1/2}\mathcal{N}\big(\boldsymbol{\mu}|\mathbf{a},\boldsymbol{\Lambda}_a+\boldsymbol{\Sigma}\big)\overbrace{\int \mathcal{N}\big(\mathbf{z}|\boldsymbol{\mu}_{\mathbf{a}},\boldsymbol{\Sigma}_{\mathbf{a}}\big)\mathrm{d}\mathbf{z}}^{1} \\ &= \alpha_a^2\big|\boldsymbol{\Sigma}\boldsymbol{\Lambda}_a^{-1}+\mathbf{I}\big|^{-1/2}\exp\Big(-\tfrac{1}{2}(\mathbf{a}-\boldsymbol{\mu})^\top(\boldsymbol{\Lambda}_a+\boldsymbol{\Sigma})^{-1}(\mathbf{a}-\boldsymbol{\mu})\Big)\end{aligned} \tag{3.52}$$

The first step is made by pulling out the normalising constant for the squared exponential kernel to make it a Gaussian and the second step is due to the identity for the multiplication of two Gaussian densities given in Appendix A.1. Note that setting $\boldsymbol{\Sigma} = \sigma\mathbf{I}$ and letting $\sigma \to 0$, the standard expression for the SE kernel is recovered as expected. The mean $\boldsymbol{\mu}_{\mathbf{a}}$ and covariance $\boldsymbol{\Sigma}_{\mathbf{a}}$ are given by

$$\boldsymbol{\mu}_{\mathbf{a}} := (\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Sigma}^{-1})^{-1}\big(\boldsymbol{\Lambda}_a^{-1}\mathbf{a} + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\big) = (\boldsymbol{\Sigma}\boldsymbol{\Lambda}_a^{-1} + \mathbf{I})^{-1}\big(\boldsymbol{\Sigma}\boldsymbol{\Lambda}_a^{-1}\mathbf{a} + \boldsymbol{\mu}\big) \tag{3.53}$$

$$\boldsymbol{\Sigma}_{\mathbf{a}} := (\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Sigma}^{-1})^{-1} = (\boldsymbol{\Sigma}\boldsymbol{\Lambda}_a^{-1} + \mathbf{I})^{-1}\boldsymbol{\Sigma} \tag{3.54}$$

which can be written in a form in which there is no need to invert the potentially singular matrix $\boldsymbol{\Sigma}$ explicitly. Now turning to the second expectation in Eq. (3.51) and following a similar procedure gives

$$\begin{aligned}
&\mathbb{E}_{\mathbf{z}}\big[k_a(\mathbf{a},\mathbf{z})k_b(\mathbf{b},\mathbf{z})\big] \\
&\quad= \int k_a(\mathbf{a},\mathbf{z})k_b(\mathbf{b},\mathbf{z})\mathcal{N}(\mathbf{z}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\mathbf{z} \\
&\quad= \alpha_a^2\alpha_b^2(2\pi)^D|\boldsymbol{\Lambda}_a|^{1/2}|\boldsymbol{\Lambda}_b|^{1/2}\int \mathcal{N}(\mathbf{z}|\mathbf{a},\boldsymbol{\Lambda}_a)\mathcal{N}(\mathbf{z}|\mathbf{b},\boldsymbol{\Lambda}_b)\mathcal{N}(\mathbf{z}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\mathbf{z} \\
&\quad= \alpha_a^2\alpha_b^2(2\pi)^D|\boldsymbol{\Lambda}_a|^{1/2}|\boldsymbol{\Lambda}_b|^{1/2}\mathcal{N}\big(\mathbf{a}|\mathbf{b},\boldsymbol{\Lambda}_a+\boldsymbol{\Lambda}_b\big)\mathcal{N}\big(\boldsymbol{\mu}|\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\Sigma}}\big)\overbrace{\int \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\mathbf{ab}},\boldsymbol{\Sigma}_{\mathbf{ab}})\mathrm{d}\mathbf{z}}^{1} \\
&\quad= \alpha_a^2\alpha_b^2\big|\boldsymbol{\Sigma}(\boldsymbol{\Lambda}_a^{-1}+\boldsymbol{\Lambda}_b^{-1})+\mathbf{I}\big|^{-1/2}\exp\Big(-\tfrac{1}{2}(\mathbf{a}-\mathbf{b})^\top(\boldsymbol{\Lambda}_a+\boldsymbol{\Lambda}_b)^{-1}(\mathbf{a}-\mathbf{b})\Big) \\
&\qquad\qquad \exp\Big(-\tfrac{1}{2}(\hat{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top(\hat{\boldsymbol{\Sigma}}+\boldsymbol{\Sigma})^{-1}(\hat{\boldsymbol{\mu}}-\boldsymbol{\mu})\Big)
\end{aligned}$$

where $\hat{\boldsymbol{\mu}} := \hat{\boldsymbol{\Sigma}}(\boldsymbol{\Lambda}_a^{-1}\mathbf{a}+\boldsymbol{\Lambda}_b^{-1}\mathbf{b})$ and $\hat{\boldsymbol{\Sigma}} := (\boldsymbol{\Lambda}_a^{-1}+\boldsymbol{\Lambda}_b^{-1})^{-1}$. This can be written in a more intuitive manner by expanding and collecting terms in the exponents

$$\mathbb{E}_{\mathbf{z}}\big[k_a(\mathbf{a},\mathbf{z})k_b(\mathbf{b},\mathbf{z})\big] = k_a(\mathbf{a},\boldsymbol{\mu})k_b(\mathbf{b},\boldsymbol{\mu})\big|\mathbf{R}\big|^{-1/2}\exp\Big(\tfrac{1}{2}\bar{\boldsymbol{\mu}}^\top\mathbf{R}^{-1}\boldsymbol{\Sigma}\bar{\boldsymbol{\mu}}\Big) \tag{3.55}$$

where $\mathbf{R} := \boldsymbol{\Sigma}(\boldsymbol{\Lambda}_a^{-1}+\boldsymbol{\Lambda}_b^{-1})+\mathbf{I}$ and $\bar{\boldsymbol{\mu}} := \boldsymbol{\Lambda}_a^{-1}(\mathbf{a}-\boldsymbol{\mu})+\boldsymbol{\Lambda}_b^{-1}(\mathbf{b}-\boldsymbol{\mu})$. Note that $\mathbf{R}^{-1}\boldsymbol{\Sigma} = (\boldsymbol{\Lambda}_a^{-1}+\boldsymbol{\Lambda}_b^{-1}+\boldsymbol{\Sigma}^{-1})^{-1}$ is a symmetric matrix and can again be computed without explicit inversion of $\boldsymbol{\Sigma}$. For completeness, note that the mean $\boldsymbol{\mu}_{\mathbf{ab}}$ and covariance $\boldsymbol{\Sigma}_{\mathbf{ab}}$ are given by

$$\boldsymbol{\mu}_{\mathbf{ab}} := (\boldsymbol{\Lambda}_a^{-1}+\boldsymbol{\Lambda}_b^{-1}+\boldsymbol{\Sigma}^{-1})^{-1}\big(\boldsymbol{\Lambda}_a^{-1}\mathbf{a}+\boldsymbol{\Lambda}_b^{-1}\mathbf{b}+\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\big) = \mathbf{R}^{-1}\big(\boldsymbol{\Sigma}(\boldsymbol{\Lambda}_a^{-1}\mathbf{a}+\boldsymbol{\Lambda}_b^{-1}\mathbf{b})+\boldsymbol{\mu}\big) \tag{3.56}$$

$$\boldsymbol{\Sigma}_{\mathbf{ab}} := (\boldsymbol{\Lambda}_a^{-1}+\boldsymbol{\Lambda}_b^{-1}+\boldsymbol{\Sigma}^{-1})^{-1} = \mathbf{R}^{-1}\boldsymbol{\Sigma} \tag{3.57}$$

To conclude, the expectations required for moment matching in Eq. (3.51) are given by Eqs. (3.52)–(3.55) for the squared exponential kernel defined in Eq. (3.39).

Additive Squared-Exponential Kernel

The following derivations are new to the field and a contribution of our work. The key to deriving these equations comes from the observation that an infinite length scale associated with a variable is equivalent to ignoring it. This means that we can re-write the additive kernel defined in Eq. (3.42) as

$$k(\mathbf{z},\tilde{\mathbf{z}}) = \sum_{\mathbf{i}\in\mathcal{I}} \alpha_{\mathbf{i}}^2 \exp\Big(-\tfrac{1}{2}\big(\mathbf{z}-\tilde{\mathbf{z}}\big)^\top \boldsymbol{\Xi}_{\mathbf{i}}^{-1}\big(\mathbf{z}-\tilde{\mathbf{z}}\big)\Big) \tag{3.58}$$

where $\boldsymbol{\Xi}_{\mathbf{i}}^{-1} \in \mathbb{R}^{D\times D}$ is a diagonal matrix with nonzero elements given by the diagonal of $\boldsymbol{\Lambda}_{\mathbf{i}}^{-1}$ on the appropriate dimension. For example if $D = 3$, $\mathbf{i} = [1;3]$ and $\boldsymbol{\Lambda}_{\mathbf{i}}^{-1} = \operatorname{diag}\{[\lambda_1^{-2},\lambda_3^{-2}]\}$ then $\boldsymbol{\Xi}_{\mathbf{i}}^{-1} = \operatorname{diag}\{[\lambda_1^{-2},0,\lambda_3^{-2}]\}$. Acknowledging this means that we can derive the appropriate equations directly from the SE equations in Eqs. (3.52)–(3.55). The first expectation is

$$\begin{aligned}
&\mathbb{E}_{\mathbf{z}}\big[k_a(\mathbf{a},\mathbf{z})\big]\\
&\quad= \sum_{\mathbf{i}\in\mathcal{I}} \alpha_{\mathbf{i}a}^2 \big|\boldsymbol{\Sigma}\boldsymbol{\Xi}_{\mathbf{i}a}^{-1}+\mathbf{I}\big|^{-1/2} \exp\Big(-\tfrac{1}{2}(\mathbf{a}-\boldsymbol{\mu})^\top(\boldsymbol{\Xi}_{\mathbf{i}a}+\boldsymbol{\Sigma})^{-1}(\mathbf{a}-\boldsymbol{\mu})\Big)\\
&\quad= \sum_{\mathbf{i}\in\mathcal{I}} \alpha_{\mathbf{i}a}^2 \big|\boldsymbol{\Sigma}[\mathbf{i},\mathbf{i}]\boldsymbol{\Lambda}_{\mathbf{i}a}^{-1}+\mathbf{I}\big|^{-1/2} \exp\Big(-\tfrac{1}{2}\big(\mathbf{a}[\mathbf{i}]-\boldsymbol{\mu}[\mathbf{i}]\big)^\top\big(\boldsymbol{\Lambda}_{\mathbf{i}a}+\boldsymbol{\Sigma}[\mathbf{i},\mathbf{i}]\big)^{-1}\big(\mathbf{a}[\mathbf{i}]-\boldsymbol{\mu}[\mathbf{i}]\big)\Big)
\end{aligned} \tag{3.59}$$

while the second is

$$\mathbb{E}_{\mathbf{z}}[k_a(\mathbf{a},\mathbf{z})k_b(\mathbf{b},\mathbf{z})] = \sum_{\mathbf{i}\in\mathcal{I}}\sum_{\mathbf{j}\in\mathcal{I}} k_a(\mathbf{a},\boldsymbol{\mu})k_b(\mathbf{b},\boldsymbol{\mu})\big|\mathbf{R}_{\mathbf{ij}}\big|^{-1/2}\exp\Big(\tfrac{1}{2}\bar{\boldsymbol{\mu}}^\top\mathbf{R}_{\mathbf{ij}}^{-1}\boldsymbol{\Sigma}\bar{\boldsymbol{\mu}}\Big) \tag{3.60}$$

where $\mathbf{R}_{\mathbf{ij}} := \boldsymbol{\Sigma}(\boldsymbol{\Xi}_{\mathbf{i}a}^{-1}+\boldsymbol{\Xi}_{\mathbf{j}b}^{-1})+\mathbf{I}$ and $\bar{\boldsymbol{\mu}} := \boldsymbol{\Xi}_{\mathbf{i}a}^{-1}(\mathbf{a}-\boldsymbol{\mu})+\boldsymbol{\Xi}_{\mathbf{j}b}^{-1}(\mathbf{b}-\boldsymbol{\mu})$. Observe that all $\boldsymbol{\Xi}$ variables appear in their inverted form while $\boldsymbol{\Sigma}$ appears only as itself which means that calculation of these terms can be done in a numerically stable manner.

3.6 Constraints

3.6.1 State Constraints

It should be noted that the following general manner in which to address state and action constraints is a further contribution of this thesis. First, observe that the optimisation problem posed in Eqs. (3.2)–(3.3) is unconstrained. However, state and action constraints can still be addressed. Consider the case where the state is constrained to lie within some convex polytope $\mathbb{X} = \big\{\mathbf{x}\in\mathbb{R}^E \big| \mathbf{G}\mathbf{x}+\mathbf{g}\in[-\mathbf{1},\mathbf{1}]\big\}$ parameterised by $\mathbf{G}\in\mathbb{R}^{G\times E}$ and $\mathbf{g}\in\mathbb{R}^G$. This can be addressed in the probabilistic framework by soft constraints implemented using barrier functions. Two examples of functions for which the expectation with respect to $\mathbf{x}\sim\mathcal{N}$ can be evaluated

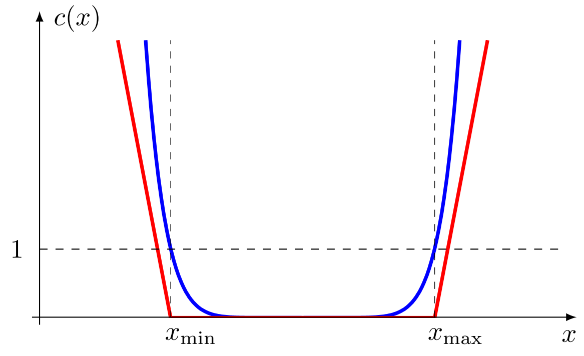


Figure 3.10: Barrier functions for which the expectation in Eq. (3.5) can be evaluated analytically for $x \sim \mathcal{N}$. These functions in are useful for penalising deviations outside of the set defined by x_{min} and x_{max}. The polynomial is shown in blue and the affine is shown in red.

analytically are polynomial and affine barrier functions

$$c_{\text{poly}}(\mathbf{x}) = \sum_{i=1}^{G} \Big(g_i(\mathbf{x})\Big)^p \tag{3.61}$$

$$c_{\text{aff}}(\mathbf{x}) = \sum_{i=1}^{G} \begin{cases} -m\Big(g_i(\mathbf{x})+1\Big) & \text{if } g_i(\mathbf{x}) \leq -1 \\ 0 & \text{if } -1 < g_i(\mathbf{x}) \leq 1 \\ m\Big(g_i(\mathbf{x})-1\Big) & \text{if } g_i(\mathbf{x}) > 1 \end{cases} \tag{3.62}$$

where $g_i(\mathbf{x}) = \mathbf{G}^{(i,:)}\mathbf{x} + g^{(i)}$, p is the order of the polynomial and m is the gradient of the slope. These are depicted in Fig. 3.10. Note that the traditional log-barrier function is inappropriate in this context since the integral with respect to a Gaussian would be infinite.

3.6.2 Action Constraints

Constraints on the action space, again in the form of a convex polytope $\mathbb{U} = \big\{\mathbf{u} \in \mathbb{R}^F \big| \mathbf{H}\mathbf{u} + \mathbf{h} \in [-\mathbf{1}, \mathbf{1}]\big\}$ parameterised by matrix $\mathbf{H} \in \mathbb{R}^{H \times F}$, can be addressed directly through the way in which the policy $\boldsymbol{\pi}$ is structured. Note that in this section H shall denote the number of action constraints rather than the prediction horizon. A policy $\boldsymbol{\pi} : \mathbb{R}^E \to \mathbb{U}$ can be constructed in the following general form for $H \geq F$ linearly independent constraints

$$\boldsymbol{\pi}(\mathbf{x}) = \mathbf{H}^\dagger \mathbf{sat}\big(\mathbf{H}\tilde{\boldsymbol{\pi}}(\mathbf{x}) + \mathbf{h}\big) - \mathbf{H}^\dagger \mathbf{h} \tag{3.63}$$

where $\mathbf{H}^\dagger = (\mathbf{H}^\top\mathbf{H})^{-1}\mathbf{H}^\top$ is the right pseudo-inverse of $\mathbf{H}$ and $\tilde{\boldsymbol{\pi}} : \mathbb{R}^E \to \mathbb{R}^F$ is a general unconstrained policy structure. The saturation function $\mathbf{sat} : \mathbb{R}^H \to \mathbb{R}^H$ has elements

$$\text{sat}^{(i)}(\mathbf{u}) = \begin{cases} -1 & \text{if } u^{(i)} \leq -1 \\ u^{(i)} & \text{if } -1 < u^{(i)} \leq 1 \\ 1 & \text{if } u^{(i)} > 1 \end{cases} \tag{3.64}$$

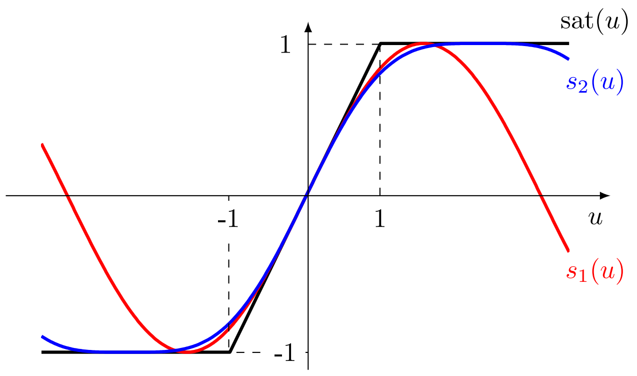


Figure 3.11: Approximations to the sat function for which the expectation and covariance with respect to a Gaussian distributed input can be evaluated analytically. Specifically $s_1(u) = \sin(u)$ and $s_2(u) = \frac{9}{8}\sin\left(\frac{2}{3}u\right) + \frac{1}{8}\sin(2u)$. Both satisfy the conditions $s_i(u) \in [-1, 1]$ and $s_i'(0) = 1$.

for $i \in \mathbb{Z}_{[1,H]}$. However, this will not fit into the moment matching framework since $\text{cov}_{\mathbf{u}}[\textbf{sat}(\mathbf{u})]$ for $\mathbf{u} \sim \mathcal{N}$ cannot be evaluated in closed form. Therefore this function is approximated using sinusoids, for which moment matching is possible. Two candidate approximations are

$$s_1(u) = \sin(u) \tag{3.65}$$

$$s_2(u) = \tfrac{9}{8}\sin\left(\tfrac{2}{3}u\right) + \tfrac{1}{8}\sin(2u) \tag{3.66}$$

which are depicted in Fig. 3.11. Note that a standard Fourier series expansion would not be appropriate since the approximation must lie in the closed region $s_i(u) \in [-1, 1]$ and have unit gradient at the origin $s_i'(0) = 1$ such that $\boldsymbol{\pi}(\mathbf{x}) \approx \tilde{\boldsymbol{\pi}}(\mathbf{x})$ for $\mathbf{H}\tilde{\boldsymbol{\pi}}(\mathbf{x}) + \mathbf{h} \approx \mathbf{0}$. Again, defining the action constraints in this general way is also a contribution of this thesis.

3.7 Summary

In this chapter we have defined the learning control framework, based on the algorithm of Deisenroth & Rasmussen (2011), that we shall use in the remainder of this thesis. We have outlined how Gaussian processes can be used as a probabilistic modelling tool for dynamical systems and how they can be used to obtain distributions over full trajectories in the state-action space. Kernels that can be used for defining distributions over linear systems, general additive systems and nonlinear systems have also been introduced. Finally, methods for handling state and action constraints have been given.

CHAPTER 4

Multiple Dynamics Models

4.1 Introduction

We now begin to address the question of how to incorporate useful prior knowledge regarding the dynamics of a system into the probabilistic learning framework. In this chapter we deal with the problem of how to include known, or approximate, relationships between state variables. An important example of such a relationship is state variables that are related as time derivatives. In other words, position-velocity relationships. Another example would be higher order Markov systems where the system "state" actually depends on a number of delayed "states". Further, the task of tracking a known reference signal, or distribution of reference signals, could be contrived as a known relationship between state variables. With the probabilistic learning framework we have derived thus far, it is not clear how such information could be incorporated. We provide a novel method that tackles this problem.

Once we have formulated a method to achieve the goal of incorporating known state relationships we would like to investigate what effect their inclusion will have on learning. In particular we would like to investigate whether learning can proceed faster, both computationally and in terms of finding a good policy, and what is the propensity for finding good (or bad) local minima in completing the task.

Our method is formulated by considering that the system we are dealing with can be decomposed as multiple dynamics models all acting on different parts, or linear transformations, of the state-action vector. We shall use the terms *multiple dynamics* and *augmented dynamics* interchangeably when referring to this framework. In order to fit into the probabilistic framework, the use of multiple dynamics must satisfy the propagation of uncertainty criteria outlined in the previous chapter. It is this problem that we tackle now.

4.2 Augmented Dynamics

4.2.1 Framework

Consider the case in which the system dynamics function $\mathbf{f}$ consists of the concatenation of M distinct functions

$$\mathbf{f}(\mathbf{z}) = \mathbf{f}_{1:M}(\mathbf{z}) := \begin{bmatrix} \mathbf{f}_1(\mathbf{z}) \\ \vdots \\ \mathbf{f}_M(\mathbf{z}) \end{bmatrix} \tag{4.1}$$

Each sub-function (or sub-dynamics) $\mathbf{f}_m$ can explicitly depend on any of the previous functional outputs $\mathbf{f}_{1:m-1}$ as well as the state-action $\mathbf{z}$. In mathematical terms $\mathbf{f}_m(\mathbf{z}) = \mathbf{f}_m\big(\mathbf{z}, \mathbf{f}_{1:m-1}(\mathbf{z})\big)$ for $m \in \mathbb{Z}_{[2,M]}$. We define the concatenation of the state-action and the $m-1$ previous functions as

$$\mathbf{p}_m(\mathbf{z}) = \begin{bmatrix} \mathbf{z} \\ \mathbf{f}_{1:m-1}(\mathbf{z}) \end{bmatrix}$$

where $\mathbf{p}_1(\mathbf{z}) = \mathbf{z}$ as this will be helpful later on. To see how this setup is beneficial for encoding position-velocity relationships or incorporating reference signals we show how they can be phrased in this framework using the following examples.

Example 4.1 (Position-Velocity). *Consider a system with time derivative related states* $\mathbf{x} = [\mathbf{a}; \dot{\mathbf{a}}]$ *where the positions* $\mathbf{a}$ *evolve according to the dynamics* $\mathbf{a}_+ = \mathbf{f}_1(\mathbf{z})$*. The velocities* $\dot{\mathbf{a}}$ *could then be reconstructed using some approximate relationship* $\dot{\mathbf{a}}_+ = \mathbf{f}_2\big(\mathbf{z}, \mathbf{f}_1(\mathbf{z})\big)$*, for example a linear relationship* $\dot{\mathbf{a}}_+ = \mathbf{M}\big[\mathbf{z}; \mathbf{f}_1(\mathbf{z})\big]$*. One setting for such a matrix could be* $\mathbf{M} = [-\mathbf{I}, \mathbf{0}, \mathbf{I}]/\delta_t$ *which would encode the relationship* $\dot{\mathbf{a}}_+ = (\mathbf{a}_+ - \mathbf{a})/\delta_t$ *which is reasonable for small* δ_t*. This example will be pursued further in Sec. 4.3.2.*

Example 4.2 (Reference Generator). *Consider a system with dynamics* $\mathbf{x}_+ = \mathbf{f}_1(\mathbf{z})$ *which has been tasked to track a some reference signal* $\mathbf{r}$*. Now assume that this reference comes from some underlying dynamical system* $\mathbf{x}^{\mathbf{r}}_+ = \mathbf{f}_2(\mathbf{x}^{\mathbf{r}})$ *where* $\mathbf{r} = \mathbf{C}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}$*. The dynamics of the system itself* $\mathbf{x}$ *can then be augmented with the state of the reference system* $\mathbf{x}^{\mathbf{r}}$ *such that the reference state simply becomes part of an augmented state space. Note that the function* $\mathbf{f}^{\mathbf{r}}$ *could be provided by the user or inferred from data along with the system dynamics. How this setup can be used for learning reference tracking will be explored in Sec. 4.4.*

In order to fit into the probabilistic framework of the previous chapter it is necessary for us to build up a moment-matched Gaussian approximation $\mathbf{f}(\mathbf{z}) \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ given $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Assumption 3.2 states that $\boldsymbol{\mu}_* = \mathbb{E}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})]$ and $\boldsymbol{\Sigma}_* = \mathrm{cov}_{\mathbf{z},\mathbf{f}}[\mathbf{f}(\mathbf{z})]$. However, further approximations will have to be made in the case of multiple dynamics models since these moments cannot be calculated for general sub-functions even if the moments of each can be evaluated individually.

Assumption 4.1. *Given $p(\mathbf{z})$ is Gaussian, the resulting distribution of the next state $p\big(\mathbf{f}(\mathbf{z})\big)$ is replaced by the Gaussian distribution*

$$\mathbf{f}(\mathbf{z}) \sim \mathcal{N}\left(\begin{bmatrix} \mathbb{E}_{\mathbf{z},\mathbf{f}_1}[\mathbf{f}_1(\mathbf{z})] \\ \vdots \\ \mathbb{E}_{\boldsymbol{\rho}_M,\mathbf{f}_M}[\mathbf{f}_M(\boldsymbol{\rho}_M)] \end{bmatrix}, \begin{bmatrix} \mathrm{cov}_{\mathbf{z},\mathbf{f}_1}[\mathbf{f}_1(\mathbf{z})] & \dots & \mathrm{cov}_{\boldsymbol{\rho}_M,\mathbf{f}_M}[\mathbf{f}_1(\mathbf{z}), \mathbf{f}_M(\boldsymbol{\rho}_M)] \\ \vdots & \ddots & \vdots \\ \mathrm{cov}_{\boldsymbol{\rho}_M,\mathbf{f}_M}[\mathbf{f}_M(\boldsymbol{\rho}_M), \mathbf{f}_1(\mathbf{z})] & \dots & \mathrm{cov}_{\boldsymbol{\rho}_M,\mathbf{f}_M}[\mathbf{f}_M(\boldsymbol{\rho}_M)] \end{bmatrix}\right)$$

where $p(\boldsymbol{\rho}_m)$ is the moment matched approximation of the real distribution $p\big(\mathbf{p}_m(\mathbf{z})\big)$.

Note that this is an iterative extension of Assumption 3.2. As before, we shall no longer make a distinction between the real distribution $p\big(\mathbf{p}_m(\mathbf{z})\big)$ and its approximation $p(\boldsymbol{\rho}_m)$. Propagation of uncertainty is now reduced to the iterative procedure of evaluating the mean $\mathbb{E}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big)\big]$ and covariance $\mathrm{cov}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big)\big]$ for each dynamics model, given the assumed joint Gaussian over the previous values $\mathbf{p}_m(\mathbf{z}) \sim \mathcal{N}$. This is possible provided that these moments can be evaluated for each sub-dynamics.

Now in order to fill out the full covariance matrix, given Assumption 4.1, the cross terms $\mathrm{cov}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big), \mathbf{p}_m(\mathbf{z})\big]$ are required. It turns out that if the mean $\mathbb{E}_{\mathbf{p}_m,\mathbf{f}_m}[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big)]$ is differentiable with respect to the input mean $\boldsymbol{\mu} = \mathbb{E}_{\mathbf{p}_m}[\mathbf{p}_m(\mathbf{z})]$ then this term can always be evaluated as

$$\mathrm{cov}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big), \mathbf{p}_m(\mathbf{z})\big] = \Big(\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}}\mathbb{E}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big)\big]\Big)\boldsymbol{\Sigma} \tag{4.2}$$

where $\boldsymbol{\Sigma} = \mathrm{cov}_{\mathbf{p}_m}[\mathbf{p}_m(\mathbf{z})]$ is the input covariance. This expression holds due to Theorem 4.1. This theorem lies at the heart of the multi-model framework. While it is a straightforward derivation, this result was unknown to us from the literature, therefore the derivation is our own.

Theorem 4.1 (Output-Input Covariance). *Consider two random vectors* $\mathbf{a}$ *and* $\mathbf{b}$ *where* $\mathbf{b}$ *is functionally dependent on* $\mathbf{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$*. Then the following statement is true*

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}}\mathbb{E}_{\mathbf{a},\mathbf{b}}[\mathbf{b}] = \mathrm{cov}_{\mathbf{a},\mathbf{b}}[\mathbf{b}, \mathbf{a}]\boldsymbol{\Sigma}^{-1}$$

Proof. The proof follows directly from the definition of expectation and covariance

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}}\mathbb{E}_{\mathbf{a},\mathbf{b}}[\mathbf{b}] &= \int \mathbb{E}_{\mathbf{b}}[\mathbf{b}]\Big(\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}}\mathcal{N}(\mathbf{a}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\Big)\mathrm{d}\mathbf{a} \\ &= \int \mathbb{E}_{\mathbf{b}}[\mathbf{b}]\Big((\mathbf{a} - \boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}\mathcal{N}(\mathbf{a}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\Big)\mathrm{d}\mathbf{a}\end{aligned}$$

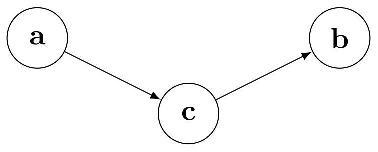


Figure 4.1: Graphical model for which the conditional independence relationship $\mathbf{a} \perp\!\!\!\perp \mathbf{b}|\mathbf{c}$ holds. See Bishop (2006) Chapter 8.2 for a more detailed outline.

Now expanding out the brackets yields the expression

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}}\mathbb{E}_{\mathbf{a},\mathbf{b}}[\mathbf{b}] &= \Big(\mathbb{E}_{\mathbf{a},\mathbf{b}}[\mathbf{b}\mathbf{a}^\top] - \mathbb{E}_{\mathbf{a},\mathbf{b}}[\mathbf{b}]\boldsymbol{\mu}^\top\Big)\boldsymbol{\Sigma}^{-1} \\ &= \mathrm{cov}_{\mathbf{a},\mathbf{b}}[\mathbf{b},\mathbf{a}]\boldsymbol{\Sigma}^{-1} \qquad \blacksquare\end{aligned}$$

Note that the assumption of joint Gaussianity is equivalent to assuming linear relationships between dynamics models. To see this, observe that Eq. (4.2) is in the form of a Jacobian matrix (or linearised dynamics) multiplied by the input covariance.

4.2.2 Subset of Inputs

Often the case will be that each dynamics model is only functionally dependent on a subset, or linear transformation, $\mathbf{s}_m(\mathbf{z})$ of the state-actions and outputs of previous functions $\mathbf{p}_m(\mathbf{z})$. How then is the expectation and covariance with respect to $\mathbf{p}_m(\mathbf{z})$ to be filled in? The answer can be found by considering the general linear transformation $\mathbf{s}_m(\mathbf{z}) = \mathbf{P}_m\mathbf{p}_m(\mathbf{z})$. The matrix $\mathbf{P}_m$ could either pick off appropriate elements of $\mathbf{p}_m$ or be viewed as an arbitrary linear combination. Since this mapping is linear, a Gaussian distribution $\mathbf{p}_m(\mathbf{z}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ leads to another Gaussian distribution $\mathbf{s}_m(\mathbf{z}) \sim \mathcal{N}\big(\boldsymbol{\mu}_\mathbf{s}, \boldsymbol{\Sigma}_\mathbf{s}\big)$ with mean $\boldsymbol{\mu}_\mathbf{s} = \mathbf{P}_m\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}_\mathbf{s} = \mathbf{P}_m\boldsymbol{\Sigma}\mathbf{P}_m^\top$. Therefore the predictive mean and covariance are simply

$$\begin{aligned}\mathbb{E}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big)\big] &= \mathbb{E}_{\mathbf{s}_m,\mathbf{f}_m}\big[\mathbf{f}_m(\mathbf{s}_m(\mathbf{z}))\big] \\ \mathrm{cov}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big)\big] &= \mathrm{cov}_{\mathbf{s}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{s}_m(\mathbf{z})\big)\big]\end{aligned}$$

In order to obtain the cross covariance term $\mathrm{cov}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big),\mathbf{p}_m(\mathbf{z})\big] = \mathrm{cov}_{\mathbf{a},\mathbf{b}}[\mathbf{b},\mathbf{a}]$ first define $\mathbf{c} = \mathbf{s}_m(\mathbf{z})$. Then note that, since $\mathbf{b}$ is only affected by $\mathbf{a}$ through $\mathbf{c}$, the conditional independence relationship $\mathbf{a} \perp\!\!\!\perp \mathbf{b}|\mathbf{c}$ holds. This relationship can be represented by the graphical model shown in Fig. 4.1. Due to the joint Gaussian assumption, Theorem 4.2 can be appealed to. This theorem is well known in the Machine Learning community.

Theorem 4.2 (Conditional Independence). *Take three random vectors* $\mathbf{a}, \mathbf{b}$ *and* $\mathbf{c}$ *which are jointly Gaussian distributed. Given* $\mathbf{a} \perp\!\!\!\perp \mathbf{b}|\mathbf{c}$ *the following statement is true*

$$\mathrm{cov}[\mathbf{b},\mathbf{a}] = \mathrm{cov}[\mathbf{b},\mathbf{c}]\mathrm{cov}[\mathbf{c}]^{-1}\mathrm{cov}[\mathbf{c},\mathbf{a}]$$

Proof. Consider the joint distribution $p(\mathbf{a}, \mathbf{b}, \mathbf{c})$ partitioned as follows

$$\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\\ \mathbf{c}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_\mathbf{a}\\ \boldsymbol{\mu}_\mathbf{b}\\ \boldsymbol{\mu}_\mathbf{c}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_\mathbf{a} & \boldsymbol{\Sigma}_\mathbf{ab} & \boldsymbol{\Sigma}_\mathbf{ac}\\ \boldsymbol{\Sigma}_\mathbf{ba} & \boldsymbol{\Sigma}_\mathbf{b} & \boldsymbol{\Sigma}_\mathbf{bc}\\ \boldsymbol{\Sigma}_\mathbf{ca} & \boldsymbol{\Sigma}_\mathbf{cb} & \boldsymbol{\Sigma}_\mathbf{c}\end{bmatrix}\right)$$

Conditioning $\mathbf{a}$ and $\mathbf{b}$ on $\mathbf{c}$ using the identity for a Gaussian conditional distribution yields the cross covariance term

$$\mathrm{cov}[\mathbf{b}, \mathbf{a}|\mathbf{c}] = \boldsymbol{\Sigma}_\mathbf{ba} - \boldsymbol{\Sigma}_\mathbf{bc}\boldsymbol{\Sigma}_\mathbf{c}^{-1}\boldsymbol{\Sigma}_\mathbf{ca}$$

This expression is equal to zero since the conditional independence relationship $\mathbf{a} \perp\!\!\!\perp \mathbf{b}|\mathbf{c}$ can be stated equivalently as $\mathrm{cov}[\mathbf{a}, \mathbf{b}|\mathbf{c}] = \mathbf{0}$. The result follows from a rearrangement of terms. ■

However, computing the inverse of the potentially singular matrix $\mathrm{cov}[\mathbf{c}]$ is unsatisfactory. Fortunately, explicit calculation of this inversion is unnecessary since $\mathrm{cov}[\mathbf{b}, \mathbf{c}]\boldsymbol{\Sigma}_\mathbf{s}^{-1} = \mathrm{d}\mathbb{E}[\mathbf{b}]/\mathrm{d}\boldsymbol{\mu}_\mathbf{s}$ due to Theorem 4.1. Therefore the cross covariances can be expressed as

$$\mathrm{cov}_{\mathbf{p}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{p}_m(\mathbf{z})\big), \mathbf{p}_m(\mathbf{z})\big] = \left(\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}_\mathbf{s}}\mathbb{E}_{\mathbf{s}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{s}_m(\mathbf{z})\big)\big]\right)\mathbf{P}_m\boldsymbol{\Sigma}$$

where $\mathbf{P}_m\boldsymbol{\Sigma} = \mathrm{cov}[\mathbf{c}, \mathbf{a}]$. The full framework for propagating uncertainty using multiple dynamics models was implemented in MATLAB and can be found in Appendix C.1 implemented by the function `propagated.m`. This function has been developed by multiple contributors where the author's main contributions to it were the multiple dynamics framework and the derivative calculations using the vectorisation convention of Appendix A.3.

This concludes the definition of the framework for propagation of Gaussian uncertainty given multiple dynamics models with the form in Eq. (4.1). Note that the only condition on each dynamics model is that the mean $\mathbb{E}_{\mathbf{s}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{s}_m(\mathbf{z})\big)\big]$ and covariance $\mathrm{cov}_{\mathbf{s}_m,\mathbf{f}_m}\big[\mathbf{f}_m\big(\mathbf{s}_m(\mathbf{z})\big)\big]$ are analytically tractable for $\mathbf{s}_m \sim \mathcal{N}$, $\mathbf{f}_m \sim \mathcal{GP}$ and that the mean is differentiable with respect to the mean of the input distribution $\boldsymbol{\mu}_\mathbf{s}$. Now some specific applications are discussed.

4.2.3 Higher Order Markov Systems

One use of this framework that is immediately transparent is the case in which the system in question has a Markov order greater than one in its relationship to previous states. In other words, with multiple dynamics models, Markov system of order N and of the form

$$\mathbf{x}_{k+1} = \mathbf{f}\big(\mathbf{x}_k, \mathbf{x}_{k-1}\ldots, \mathbf{x}_{k-N+1}, \mathbf{u}_k\big)$$

can be considered by defining the augmented state-space system

$$\mathbf{x}_{k+1}^{\mathrm{aug}} = \begin{bmatrix}\mathbf{f}\big(\mathbf{x}_k^{\mathrm{aug}}, \mathbf{u}_k\big)\\ \begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{x}_k^{\mathrm{aug}}\end{bmatrix}$$

with augmented state vector $\mathbf{x}_k^{\text{aug}} = [\mathbf{x}_k; \mathbf{x}_{k-1} \dots \mathbf{x}_{k-N+1}] \in \mathbb{R}^{NE}$. Obviously a similar procedure could be applied if the system also depends on delayed actions. This is a useful property that we will exploit in the subsequent sections.

4.3 Known State Relationships

4.3.1 General Mappings

The framework of multiple dynamics models outlined in the previous section finds one of its most useful applications in the incorporation of known relationships between states, a form of partial model information. This application is presented in Hall *et al.* (2012). Specifically, given a system composed of M distinct functions as given in Eq. (4.1), then often some of the sub-dynamics may be fixed or known beforehand. In this case, the unknown dynamics can be inferred from data using a parametric or nonparametric method outlined in Sec. 3.4 while the known parts can be included directly. A very clear and useful example of this for dynamical systems is the case where some states are known to be time derivatives of other states. In other words, position-velocity relationships.

4.3.2 Position-Velocity

Consider the common scenario in which the state $\mathbf{x}$ contains *position* states $\mathbf{x}^{\text{p}}$ and *velocity* states $\mathbf{x}^{\text{v}}$ which are related through

$$\mathbf{x}_k^{\text{p}} = \int_{-\infty}^{t} \mathbf{x}^{\text{v}}(\tau)\mathrm{d}\tau \quad \text{or} \quad \mathbf{x}_k^{\text{p}} = \mathbf{x}_{k-1}^{\text{p}} + \int_{t-\delta_t}^{t} \mathbf{x}^{\text{v}}(\tau)\mathrm{d}\tau \tag{4.3}$$

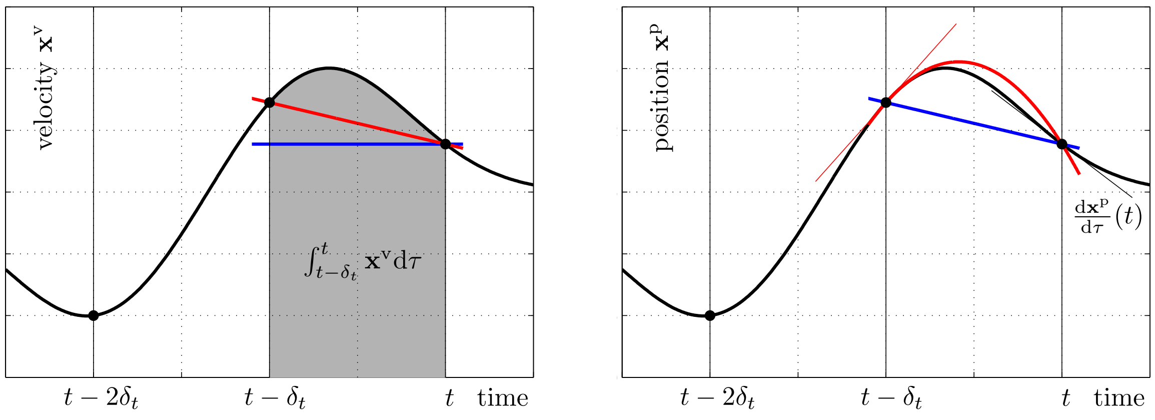
$$\mathbf{x}_k^{\text{v}} = \left.\frac{\mathrm{d}\mathbf{x}^{\text{p}}(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} \tag{4.4}$$

where $k \in \mathbb{Z}$ and $t = k\delta_t$. In order to incorporate this information into the learning framework it is important to first determine whether a predictive model for the position states is to be inferred and the velocity states reconstructed from this prediction (numerical differentiation) or vice versa (numerical integration). Approximate solutions for both cases will be discussed.

First consider the reconstruction of the position states $\mathbf{x}_k^{\text{p}}$ given the previous position $\mathbf{x}_{k-1}^{\text{p}}$ and the set of inferred velocities $\mathbf{x}_k^{\text{v}}$ and $\mathbf{x}_{k-1}^{\text{v}}$ from the probabilistic model. This corresponds to approximating the integral in Eq. (4.3) which is depicted graphically in Fig. 4.2(a). Assuming the velocity follows a constant or linear variation (equivalently the acceleration is zero or constant across the interval) corresponds to the following approximations

$$\mathbf{x}_k^{\text{p}} \approx \mathbf{x}_{k-1}^{\text{p}} + \delta_t \mathbf{x}_k^{\text{v}} \tag{4.5}$$

$$\mathbf{x}_k^{\text{p}} \approx \mathbf{x}_{k-1}^{\text{p}} + \tfrac{1}{2}\delta_t\Big(\mathbf{x}_k^{\text{v}} + \mathbf{x}_{k-1}^{\text{v}}\Big) \tag{4.6}$$



(a) Integral approximations. (b) Derivative approximations.

Figure 4.2: Approximate reconstruction of the current position/velocity states given previous observations of the velocity/position states up to time $t-\delta_t$. The thick blue and red lines in each plot make approximations of zero and constant acceleration $\dot{\mathbf{x}}^{\text{v}}$ respectively. The thin red line in plot (b) shows actual gradient at time $t-\delta_t$, which the constant acceleration approximation utilises.

respectively. These are roughly equivalent to the Euler and Heun methods for numerical integration. Now consider reconstruction of the velocity states $\mathbf{x}_k^{\text{v}}$ given the inferred positions $\mathbf{x}_k^{\text{p}}$ and $\mathbf{x}_{k-1}^{\text{p}}$. In this case assuming the position follows a linear or quadratic variation (equivalently the acceleration is zero or constant across the interval) leads to

$$\mathbf{x}_k^{\text{v}} \approx \delta_t^{-1}\left(\mathbf{x}_k^{\text{p}} - \mathbf{x}_{k-1}^{\text{p}}\right) \tag{4.7}$$

$$\mathbf{x}_k^{\text{v}} \approx 2\delta_t^{-1}\left(\mathbf{x}_k^{\text{p}} - \mathbf{x}_{k-1}^{\text{p}}\right) - \mathbf{x}_{k-1}^{\text{v}} \tag{4.8}$$

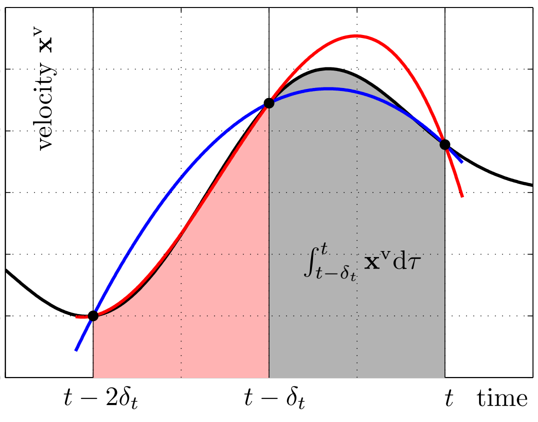
respectively. This process is depicted in Fig. 4.2(b). As expected, these relationships are simply rearrangements of Eqs. (4.5)–(4.6). All these approximate relationships are linear and can therefore be easily incorporated into the multiple dynamics framework.

As discussed in Sec. 4.2.3 it is not a problem to form dependencies on delayed states $\mathbf{x}_{k-i}$ for $i > 1$. Therefore, more elaborate approximations can be made by using information from previous timesteps. For example, fitting a quadratic curve to the points $\mathbf{x}_k^{\text{v}}, \mathbf{x}_{k-1}^{\text{v}}, \mathbf{x}_{k-2}^{\text{v}}$ or fitting a cubic curve to these points and given the area defined by $\mathbf{x}_{k-1}^{\text{p}} - \mathbf{x}_{k-2}^{\text{p}}$ yields the following approximations

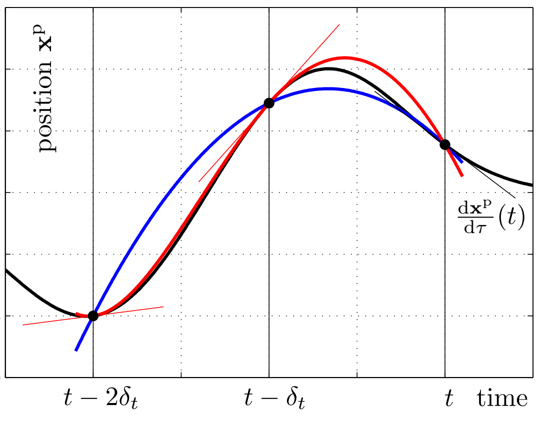
$$\mathbf{x}_k^{\text{p}} \approx \mathbf{x}_{k-1}^{\text{p}} + \tfrac{1}{12}\delta_t\left(5\mathbf{x}_k^{\text{v}} + 8\mathbf{x}_{k-1}^{\text{v}} - \mathbf{x}_{k-2}^{\text{v}}\right) \tag{4.9}$$

$$\mathbf{x}_k^{\text{p}} \approx \mathbf{x}_{k-2}^{\text{p}} + \tfrac{1}{3}\delta_t\left(\mathbf{x}_k^{\text{v}} + 4\mathbf{x}_{k-1}^{\text{v}} + \mathbf{x}_{k-2}^{\text{v}}\right) \tag{4.10}$$

for position reconstruction. These approximations are shown in Fig. 4.3(a) by the blue and red curves respectively. Note that since there are four constraints that must be satisfied then a cubic function can do this uniquely since it has four degrees of freedom. Lower order polynomials could also be considered which would yield a unique analytic solution but would not guarantee



(a) Integral approximations.



(b) Derivative approximations.

Figure 4.3: Approximate reconstruction of the current position/velocity states given previous observations of the velocity/position states up to time $t - 2\delta_t$. The blue lines in plot (a) and (b) use only the three observed points on the graph to make a quadratic approximation of the actual curve. The red line in plot (a) additionally uses the red shaded area $\int \mathbf{x}^{\mathrm{v}} \mathrm{d}\tau$ over $[t - 2\delta_t, t - \delta_t]$ to form a cubic approximation. This is equivalent to the red line in plot (b) which uses the derivatives $\frac{\mathrm{d}\mathbf{x}^{\mathrm{p}}}{\mathrm{d}\tau}(t - \delta_t)$ and $\frac{\mathrm{d}\mathbf{x}^{\mathrm{p}}}{\mathrm{d}\tau}(t - 2\delta_t)$ shown by thin red lines to make a quartic approximation.

satisfaction of the constraints.

Now, in terms of velocity reconstruction, a quadratic fit to the points $\mathbf{x}^{\mathrm{p}}_k, \mathbf{x}^{\mathrm{p}}_{k-1}, \mathbf{x}^{\mathrm{p}}_{k-2}$ and a quartic fit to these points plus the derivatives defined by $\mathbf{x}^{\mathrm{v}}_{k-1}$ and $\mathbf{x}^{\mathrm{v}}_{k-2}$ give the relationships

$$\mathbf{x}^{\mathrm{v}}_k \approx \tfrac{1}{2}\delta_t^{-1}\Big(3\mathbf{x}^{\mathrm{p}}_k - 4\mathbf{x}^{\mathrm{p}}_{k-1} + \mathbf{x}^{\mathrm{p}}_{k-2}\Big) \tag{4.11}$$

$$\mathbf{x}^{\mathrm{v}}_k \approx 3\delta_t^{-1}\Big(\mathbf{x}^{\mathrm{p}}_k - \mathbf{x}^{\mathrm{p}}_{k-2}\Big) - \Big(4\mathbf{x}^{\mathrm{v}}_{k-1} + \mathbf{x}^{\mathrm{v}}_{k-2}\Big) \tag{4.12}$$

Note that the cubic fit to the velocity profile in Eq. (4.10) is equivalent to the quartic fit to the position profile in Eq. (4.12). These approximations are shown in Fig. 4.3(b) by the blue and red curves respectively.

It is important to note that using additional information from previous timesteps and higher order polynomials will not necessarily give better approximations of the integral or derivative. This is clearly shown in Fig. 4.3 where we see that the higher order polynomial fit (thick red lines) produces a worse approximation of the required integral and derivative than the lower order fit (thick blue lines). We shall see in the following experiments whether this issue occurs in practice.

4.3.3 Example: Pendulum

Setup

In order to analyse these approximation schemes, the torque-limited pendulum swing up problem was considered. This simple system is shown in Fig. 4.4 where the pendulum is considered to be

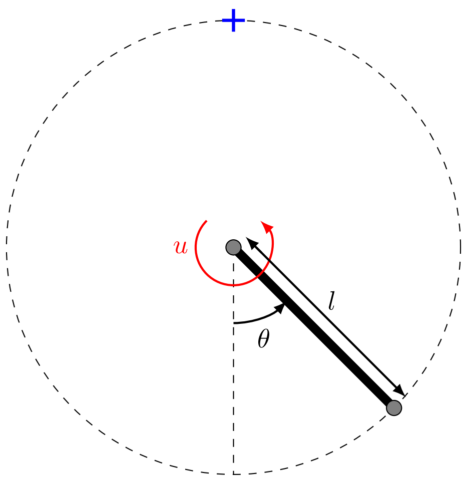


Figure 4.4: Torque-limited pendulum with angle from the down position θ, length l and input torque u.

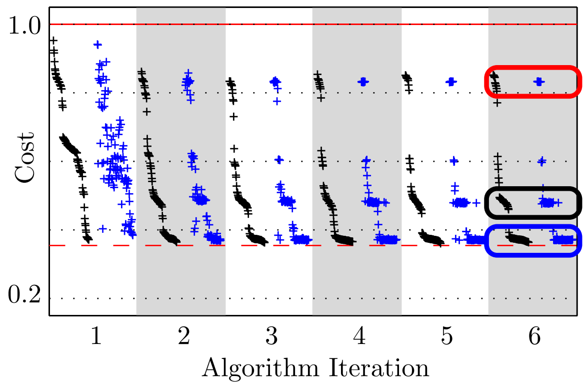
a pole of uniform density. The equation of motion for this system is therefore given by

$$\tfrac{1}{3}ml^2\ddot{\theta} = u - b\dot{\theta} - \tfrac{1}{2}mlg\sin\theta$$

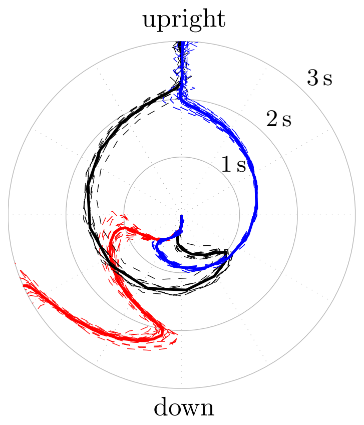
The constants we used can be found in Appendix B.1. The input torque u is constrained and is insufficient to swing the pendulum up directly. The cost function used for learning was the angular distance to the upright position $c(\mathbf{x}) = \frac{1}{2}(\cos\theta + 1) \in [0, 1]$. This cost makes no distinction between swinging the pendulum up to $\theta = \pi$ or $\theta = -\pi$. The prediction horizon was set to $T = 3\,\mathrm{s}$ with a discrete timestep $\delta_t = 0.1\,\mathrm{s}$. Note that this discretisation is relatively short given that the natural period of the pendulum is $T_0 = 2\pi/\omega_0 \approx 1.6\,\mathrm{s}$ where $\omega_0 = \sqrt{3g/2l}$. The control policy was a radial basis function with 50 Gaussian kernels in which the positions, widths and magnitudes of each kernel were free to be optimised. The output of this policy was then passed through the approximate saturating block defined in Eq. (3.66) in order to satisfy the action constraints.

The algorithm was given two independent Gaussian processes with squared exponential kernels to learn the unknown system dynamics. We tasked these GPs to learn a mapping from $[\dot{\theta}_{k-1}, \sin\theta_{k-1}, \cos\theta_{k-1}]^\top$ to the differences $(\theta_k - \theta_{k-1})$ and $(\dot{\theta}_k - \dot{\theta}_{k-1})$. This input representation was chosen to encode the fact that the angle θ is the same as the angle $\theta + 2\pi$. Differences were predicted so that a zero mean prior would be suitable. Of course this equivalent to incorporating a fixed linear mean function. The input representation at the following timestep $[\dot{\theta}_k,\ \sin\theta_k, \cos\theta_k]^\top$ could then be reconstructed using linear and trigonometric relationships, which satisfy the criteria for the moment matching approximation. The learned dynamics were then initialised with a training data set of $3\,\mathrm{s}$ which was obtained by applying random inputs to the pendulum starting with it hanging down.

This is an interesting problem due to the existence of locally optimal solutions. The optimal solution consists of a single swing back followed by the swing to the upright position. This can be verified, for example, by Dynamic Programming. However other solutions involving multiple



(a) Distribution of costs after each iteration



(b) Three locally optimal solutions

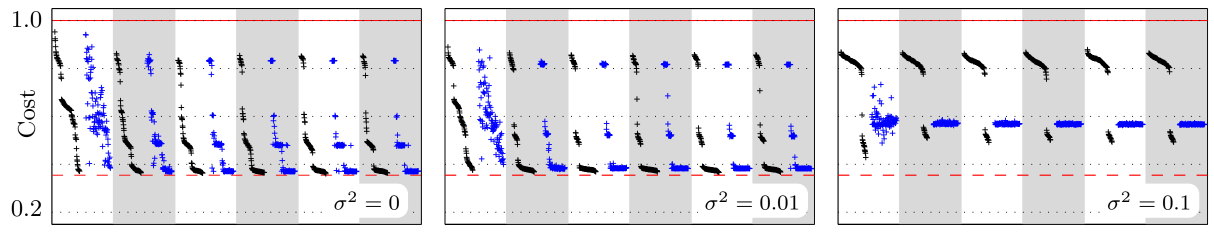
Figure 4.5: The graph in (a) depicts the cost trajectories over six iterations of 50 Monte Carlo runs of the standard learning algorithm to the pendulum problem. The black crosses show the actual costs incurred and the blue crosses show the associated mean predicted costs. The maximum cost and the global optimum are shown by the red line and the red dashed line respectively. The three solutions that the algorithm tended to settle for are shown by the trajectories in (b). The blue is close the global optimum solution, consisting of a single swing-back, while the black is a local optimum consisting of two swings. Finally, the red is a locally optimal solution of simply applying full actuation torque over the whole horizon.

swings or swinging the pendulum round and round many times also exist. We will investigate the propensity for the algorithm to fall into one of these locally optimal solutions in the presence of approximate models for the position-velocity relationship.

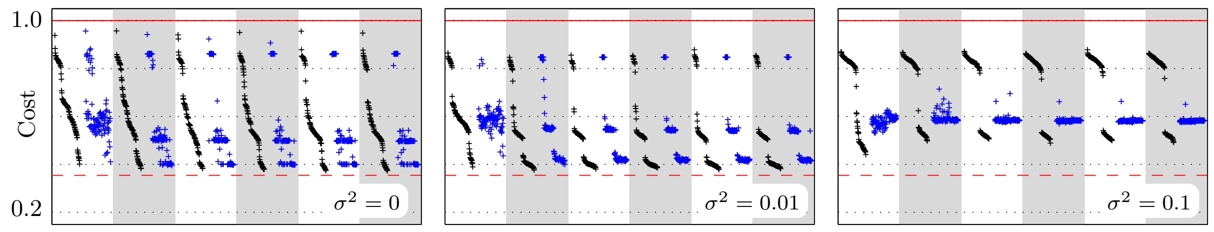
Standard Method

The results of applying the learning algorithm of Deisenroth & Rasmussen (2011), which we shall refer to as the “standard method”, to the pendulum are shown in Fig. 4.5(a). This is clearly a relatively easy task in terms of system identification since the predicted costs match the actual costs after only two iterations (6 s of data). In terms of learning the control task, many of the runs can learn a policy that works after just the first iteration (3 s of data) while the majority of runs achieve the swing up task by the second iteration.

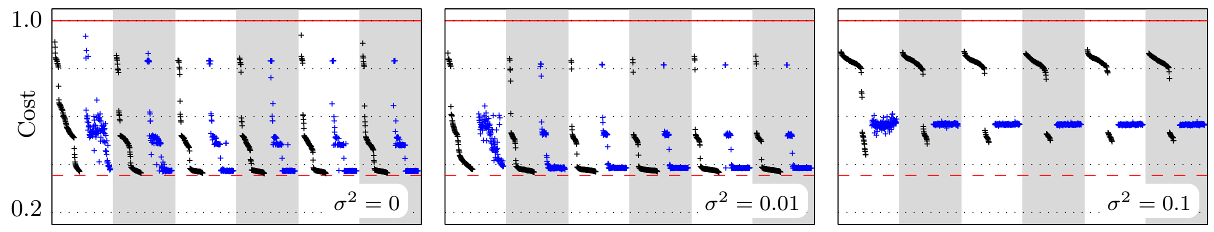
What makes this an interesting problem is not the learning speed but that there are clearly three solutions in which the algorithm gets stuck. Note that the algorithm was run as long as 10 iterations and the distribution of solutions did not change. The three locally optimal solutions are shown in Fig. 4.5(b). The first, shown in blue, is an ideal solution consisting of a single-swing back before the swing-up. This is close to the theoretical limit, shown by the dashed red line. It is desirable for all runs to end up here. The second mode, shown in black, is a locally optimal solution consisting of two swings before the swing-up. Finally, there is a common failure mode shown in red in which the policy simply applies the maximum control action over the whole horizon and gets stuck here.



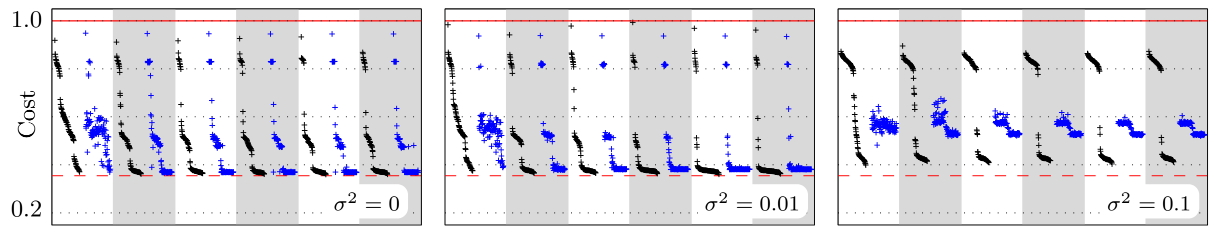
(a) Use GPs to learn evolution of both θ and $\dot{\theta}$.



(b) Reconstruct θ using the Euler method.



(c) Reconstruct θ using the Heun method.



(d) Reconstruct θ using the Cubic method.

Figure 4.6: Plots of the distribution of actual costs (black) and the mean predicted cost (blue) for different levels of additive noise $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ on the prediction of θ. The x-axis displays the algorithm iteration and the y-axis shows the cost $J(\boldsymbol{\psi})$. These plots were constructed from 100 Monte Carlo runs with initial training data sets obtained by applying different random torque actions.

Position Reconstruction

The explicit inclusion of approximate position-velocity information in the pendulum swing-up problem shall now be considered. The results of applying various position reconstruction schemes are shown in Fig. 4.6. The schemes considered shall be referred to as the Euler, Heun and Cubic methods and consist of the following equations respectively

$$\theta_k = \theta_{k-1} + \delta_t \dot{\theta}_k + \epsilon_k \tag{4.13}$$

$$\theta_k = \theta_{k-1} + \tfrac{1}{2}\delta_t\big(\dot{\theta}_k + \dot{\theta}_{k-1}\big) + \epsilon_k \tag{4.14}$$

$$\theta_k = \theta_{k-2} + \tfrac{1}{3}\delta_t\big(\dot{\theta}_k + 4\dot{\theta}_{k-1} + \dot{\theta}_{k-2}\big) + \epsilon_k \tag{4.15}$$

The additive noise term $\epsilon_k \sim \mathcal{N}(0, \sigma^2)$ is used to encode uncertainty in the approximation scheme. We use values of $\sigma^2 = 0.01$ and $\sigma^2 = 0.1$, which correspond to standard deviations of around 6° and 18° respectively. The Euler and Heun methods are so called because of their similarity to the Euler and Heun methods for numerical integration. Note that it is unnecessary to predict differences because we are not learning the relationship. Table 4.1 gives some additional statistics of the learned policies at the end of the six iterations for $\sigma^2 = 0$ and $\sigma^2 = 0.01$. It shows how many runs ended up in the bad local minimum and the number that ended up in the global optimum solution.

Firstly, observe Fig. 4.6(a) which shows the effect of applying additional noise to the Gaussian process prediction of θ. Setting $\sigma^2 = 0$ clearly reproduces the results of the standard algorithm. An interesting thing happens when the noise level is increased to $\sigma^2 = 0.01$ and that is that the algorithm converges on its solution much faster, however more runs end up in the failure mode. This additional uncertainty is in some senses equivalent to asking the algorithm to design a policy that is robust to a greater level of uncertainty. At $\sigma^2 = 0.1$ the noise level is too high for any reliable prediction or learning to take place. However it is interesting to note that it sometime enters a failure mode that involves the pendulum swinging round and round as hard as it can.

Now, consider the results of applying the Euler method in Eq. (4.13) to reconstruct θ at every timestep, shown in Fig. 4.6(b). Taking the raw approximation with $\sigma^2 = 0$ yields much poorer performance than the standard algorithm with only a few of the runs reaching a solution close to the global optimum. The large discrepancy between predicted and actual costs indicate that this approximation is too poor to give reliable performance. Moving up to $\sigma^2 = 0.01$ an interesting thing occurs. The learning performance is comparable to that of the standard algorithm. Note that the predictions are slightly under-confident. Again, with $\sigma^2 = 0.1$ the algorithm fails to achieve the task. This points to the fact that a poor approximation method can still work well given an appropriate amount of "distrust" of the results it is producing.

The Heun method with $\sigma^2 = 0$ performs marginally better than the standard algorithm as shown in Fig. 4.6(c) and Table 4.1 with less runs falling into the bad local optimum but the same number finding the global optimum. Moving to a noise level of $\sigma^2 = 0.01$ we can see that there

	Standard		**Euler**		**Heun**		**Cubic**	
σ^2	0	0.01	0	0.01	0	0.01	0	0.01
Failed	12	17	19	11	7	3	12	8
Suboptimal	33	16	63	38	38	21	30	3
Optimal	55	67	18	51	55	76	58	89

Table 4.1: This table provides the number of learning algorithm runs that have fallen into either the bad local minimum, a suboptimal solution or the global optimum solution at the end of the six trials out of the 100 Monte Carlo runs. They are compared across the different methods for position reconstruction and values of additive predictive noise. The values correspond to those depicted in Fig. 4.6.

is actually a marked improvement in performance in terms of the number of runs finding the global optimum and avoiding the bad local minimum. As a side note, the approximation based on a quadratic fit given in Eq. (4.9) produced very similar results to the Heun method. Finally, the results of the Cubic method are shown in Fig. 4.6(d). The results of noise free application has very similar performance to both the Standard and Heun methods. However, moving to $\sigma^2 = 0.01$ we see an even greater improvement in performance than observed with the Heun method with nearly 90% of the runs finding the global optimum solution but with more falling into the bad local optimum.

What can be drawn from these results is that the incorporation of time-derivative prior information can be helpful in avoiding locally optimal solutions to a given control task. But the most interesting feature is that the inclusion of uncertainty, i.e. an explicit suspicion of the predictions produced by a prior model, can potentially lead to better performance, or indeed much worse if the uncertainty level is too high.

Velocity Reconstruction

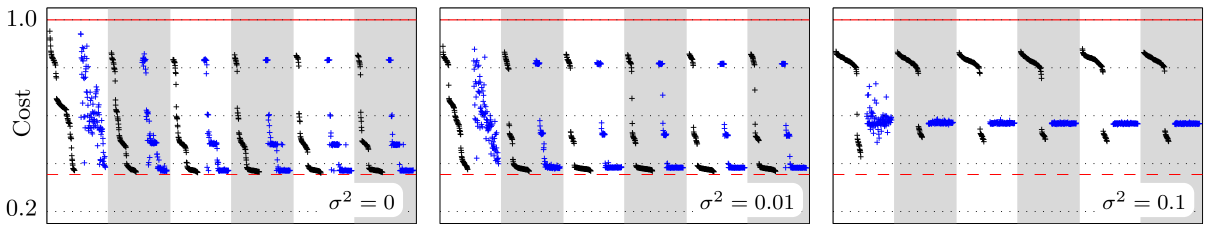
Now consider the use of velocity reconstruction to obtain predictions for $\dot{\theta}$ based on GP predictions of θ. The Euler and Heun equations in this case are given by

$$\dot{\theta}_k = \delta_t^{-1}\big(\theta_k - \theta_{k-1}\big) + \epsilon_k \tag{4.16}$$

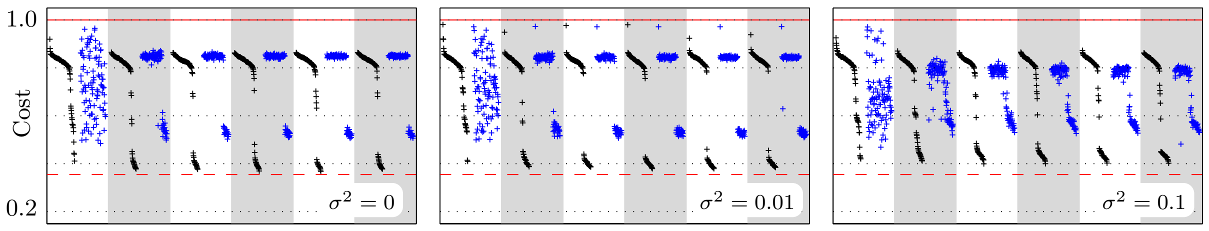
$$\dot{\theta}_k = 2\delta_t^{-1}\big(\theta_k - \theta_{k-1}\big) - \dot{\theta}_{k-1} + \epsilon_k \tag{4.17}$$

respectively. We again use noise variances of $\sigma^2 = 0.01$ and $\sigma^2 = 0.1$, which correspond, in this case, to standard deviations of $0.1\,\mathrm{rad\,s^{-1}}$ and $0.316\,\mathrm{rad\,s^{-1}}$ respectively. Higher order approximations have not actually been considered here since applying them to this problem led to an explosion in the covariance of the predicted state at later timesteps and therefore numerically unstable results. This occurred because the variance in the initial estimates of θ was quite large, but this uncertainty was then amplified many times since the coefficients of the linear model were relatively large.

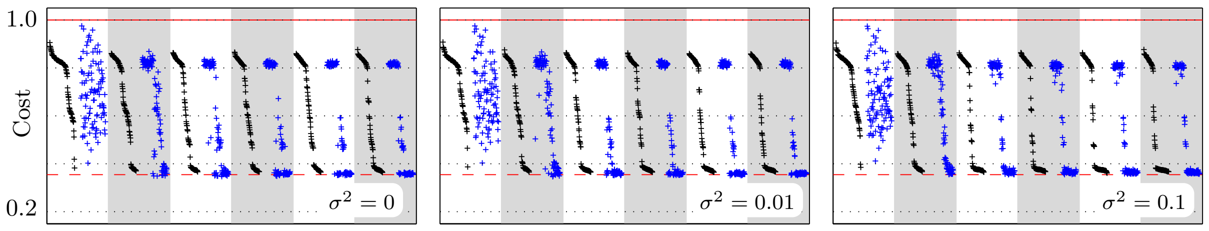
The results of the Euler and Heun approximation schemes are given in Fig. 4.7. It is quite clear



(a) Use GPs to learn evolution of both θ and $\dot{\theta}$.



(b) Reconstruct $\dot{\theta}$ using the Euler method.



(c) Reconstruct $\dot{\theta}$ using the Heun method.

Figure 4.7: Plots of the distribution of actual costs (black) and the mean predicted cost (blue) for different levels of additive noise $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ on the prediction of $\dot{\theta}$. The x-axis displays the algorithm iteration and the y-axis shows the cost $J(\boldsymbol{\psi})$. These plots were constructed from 100 Monte Carlo runs with initial training data sets obtained by applying different random torque actions.

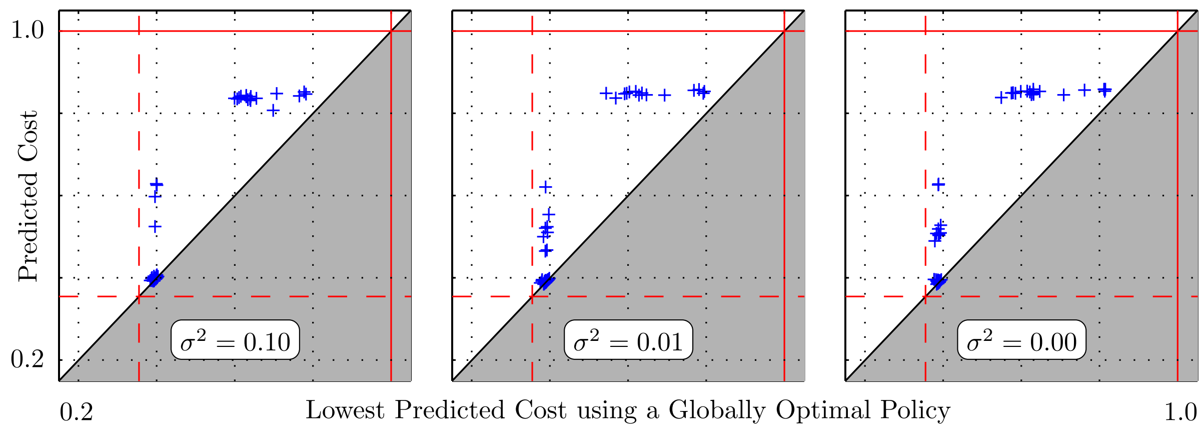


Figure 4.8: These plots are directly related to the Monte Carlo runs shown in Fig. 4.7(c) in which a Heun-type method has been used to achieve velocity reconstruction. They show the lowest predicted cost predicted by a given internal model for a globally optimal policy against the policy the algorithm actually converged on using this model. Points appearing in the gray region would indicate that the model was predicting that the globally optimal solution was performing worse than its own solution.

from these plots that the use of velocity reconstruction leads to much poorer performance than the standard algorithm in all cases. The Euler results shown in Fig. 4.7(b) fail in almost all the runs and the discrepancy between the predicted and actual costs is large. However, even in the face of such a poor approximation, some of the runs are still able to solve the task.

The effect of additional uncertainty when using a Heun approximation leads to progressively improved results with the best performance achieved when $\sigma^2 = 0.1$. Increasing noise appears to have the effect of stopping the runs falling into "weak" local minima such as the two-swing solution and polarises the results into the "strong" minima of the global optimum and the failure mode.

An important question to be raised is why does velocity reconstruction perform so much poorer than position reconstruction. It is evidently not due to the predictive quality of the predictions since for the Heun method these are clearly quite accurate. To investigate this phenomenon a further study was carried out on the Heun Monte Carlo runs shown in Fig. 4.7(c). The learned model at the end of the sixth iteration for every run was taken and every policy which achieved the globally optimal solution was evaluated with respect to this model to see whether the policy it had converged on was genuinely optimal with respect to the learned model or whether the algorithm had simply got stuck in a local minimum. The results are shown in Fig. 4.8. Clearly the problem here is that the use of velocity reconstruction in the pendulum problem produces a problem with stronger local minima than with position reconstruction.

The reason for this strange behaviour could be due to the fact that the dynamics of position variables tend to be slower than the associated velocity. This is because integration in time has a low-pass filtering effect. Therefore, trying to reconstruct the quickly varying dynamics from the slowly varying ones has more potential for erroneous results.

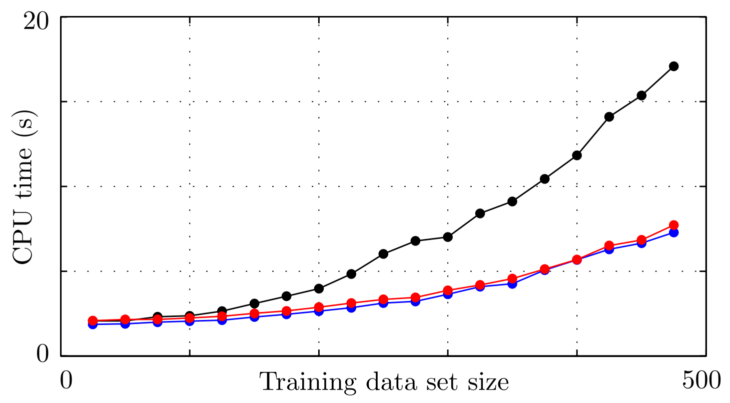


Figure 4.9: Comparison of CPU time for a single cost function evaluation (with derivatives) for the pendulum. The black line is for the prediction of θ and $\dot{\theta}$ using two GPs. The blue line is for the case where one of the states is reconstructed using information back to time $k-1$ and the red line for information back to $k-2$. The x-axis depicts the size of training data set available to the GP.

Computational Savings

One of the major advantages of using prior information in this way is the computational savings in prediction and cost function evaluation. Since we are removing one of the two Gaussian process models (for both position and velocity reconstruction) and replacing it with a linear mapping we expect the computation time to be approximately halved. Fig. 4.9 shows a plot of the CPU time required to evaluate $J(\boldsymbol{\psi})$ (and its derivatives with respect to policy parameters) against the size of the training data set available to the GP. Below about $n = 100$ there is no significant difference between the three except that it costs slightly more to include the delayed states for approximations based on information at time $k-2$. However, beyond $n = 200$ the computational savings are significant, with the inclusion of delayed states coming at essentially no extra cost. The trends do indeed show a linear saving in the number of GPs we can replace with linear mappings.

4.3.4 Example: Unicycle

Setup

The scheme was then tested on a highly nontrivial control problem: balancing a simulated robotic unicycle. The layout of this problem is shown in Fig. 4.10. What makes this such a hard problem is that the rolling motion of the unicycle can only be affected indirectly through application of a torque to the turntable on top of the unicycle. The equations of motion for this system were derived by Forster (2009) and can be found in Appendix B.3, along with a full derivation.

The state is given by $\mathbf{x} = [\phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}, \dot{\phi}_\mathrm{w}, \dot{\psi}_\mathrm{t}, x_\mathrm{c}, y_\mathrm{c}]^\top \in \mathbb{R}^{10}$ with pitch angle ϕ, roll angle θ, yaw angle ψ, wheel angle ϕ_w, turntable angle ψ_t and the location of the origin $(x_\mathrm{c}, y_\mathrm{c})$ with respect to a body-fixed reference frame. These are depicted in Fig. 4.10. We note that the

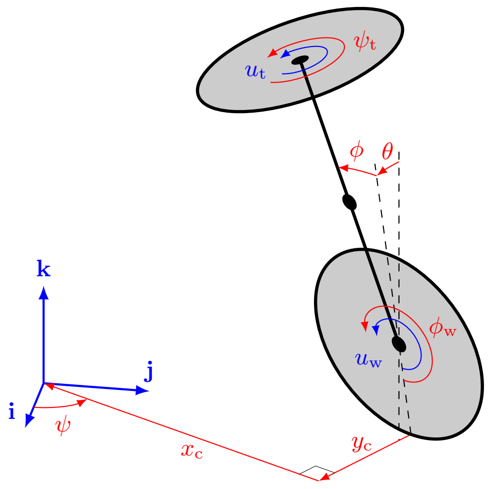


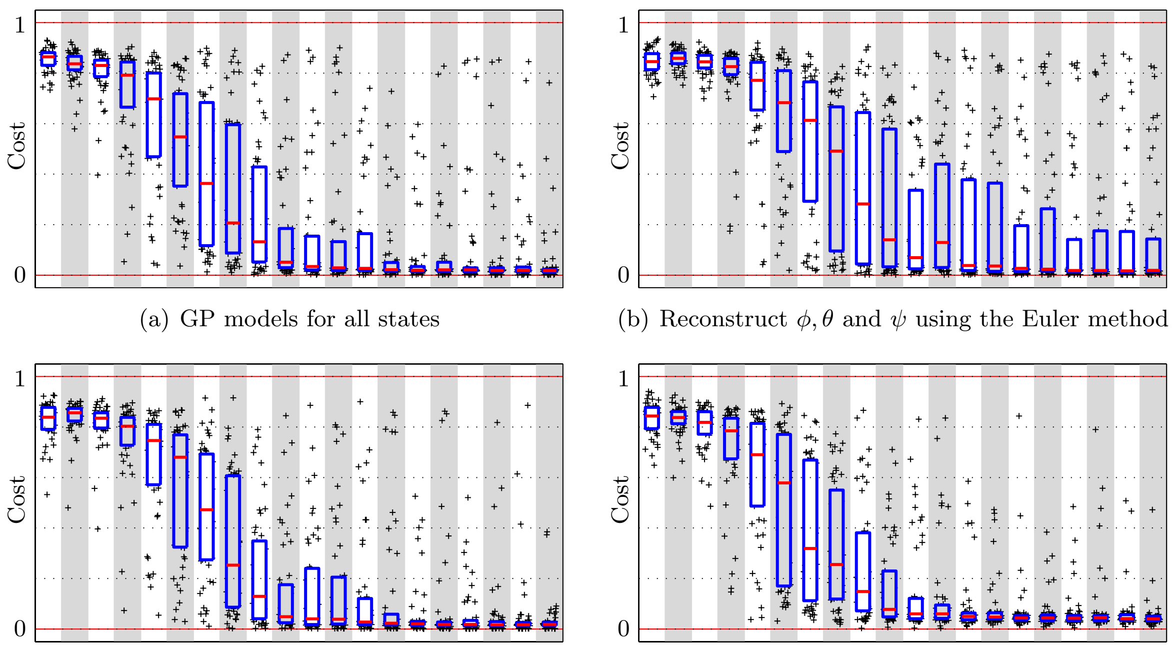
Figure 4.10: The robotic unicycle. The spatial position of the unicycle is defined by the pitch angle ϕ, roll angle θ, yaw angle ψ, wheel angle ϕ_w, turntable angle ψ_t and the body-centred coordinates of the global origin (x_c, y_c). The controlled inputs are the motor torque applied to the wheel u_w and the motor torque applied to the turntable u_t.

dynamics are clearly independent of ϕ_w and ψ_t for a wheel and turntable with uniform radial distribution of mass. The constrained action vector is $\mathbf{u} = [u_w, u_t]^\top \in \mathbb{R}^2$, where u_w is the torque applied to the wheel and u_t is the torque applied to the turntable.

The policy itself was an affine policy $\mathbf{u} = \mathbf{K}\mathbf{x} + \mathbf{k}$ with gain matrix $\mathbf{K}$ and offset $\mathbf{k}$. Note that the distinction between these terms and the covariance matrix of a GP will be clear from the context. The discrete timestep was set to $\delta_t = 0.15\,\text{s}$. This is a large timestep given the dynamics of the unicycle and therefore makes the task of control relatively hard. However, a long timestep is necessary to ensure the policy learning time is not in the order of weeks.

The state measurement was corrupted with uncorrelated Gaussian white noise. Realistic values for the standard deviation of the noise on each state are around 0.01 rad for the roll angle, 0.03 rad for all other angles, $0.03\,\text{rad}\,\text{s}^{-1}$ for the angular velocities and 3 cm for the coordinates states. With this level of noise, the learning task is too hard for the algorithm since the Gaussian processes overestimate the noise variance significantly. This is because a standard assumption of GP modelling is that the input training data is noise free. This issue has been addressed by McHutchon & Rasmussen (2011) who provide a method called Noisy Input Gaussian Processes (NIGP) which takes account of noise on the inputs during training. Using the NIGP method within the learning algorithm, it is able to learn how to balance the unicycle with this level of noise. However, it is unclear at this stage how to combine NIGP with the multiple dynamics models framework. Therefore we consider a problem in which the standard deviation of the noise on measurements of the angles and positions is reduced by a factor of ten.

The control task was to balance the unicycle at the origin $(x_c, y_c) = (0, 0)$ given an initial position drawn from a Gaussian distribution $x_c, y_c \sim \mathcal{N}(0, 0.01\mathbf{I})$, i.e. a standard deviation of 10 cm in both x_c and y_c. We use an inverted-Gaussian stage-cost very similar to one derived by



(a) GP models for all states

(b) Reconstruct ϕ, θ and ψ using the Euler method

(c) Reconstruct ϕ, θ and ψ using the Heun method

(d) Reconstruct ϕ, θ and ψ using the Cubic method

Figure 4.11: Boxplots depicting the cost $J(\boldsymbol{\psi})$ of applying the learned control policy after a given iteration of the algorithm. These were constructed from 50 Monte Carlo runs with random initial training data sets of around 10 s. The blue box encloses the 25th to 75th percentile of the data, with the 50th percentile shown by the red line.

Deisenroth *et al.* (2011). The stage-cost is of the form

$$c(\mathbf{x}) = 1 - \exp\Big(-\tfrac{1}{2}d(\mathbf{x})^2\Big) \tag{4.18}$$

where $d(\mathbf{x})^2$ is the squared distance from the top of the unicycle to the upright position over the origin. In particular this term is given by $d(\mathbf{x})^2 = d_x^2 + d_y^2 + d_z^2$ with components

$$\begin{aligned} d_x &= x_\mathrm{c} - r\sin\phi \\ d_y &= y_\mathrm{c} - (r + r_\mathrm{w})\sin\theta \\ d_z &= (r + r_\mathrm{w}) - (r_\mathrm{w} + r\cos\phi)\cos\theta \end{aligned}$$

where r is the unicycle frame length and r_w is the radius of the wheel. The expectation of this stage-cost can be evaluated in exactly the same way as the standard inverted-Gaussian cost by first augmenting the state space with $\sin\phi, \sin\theta, \cos\theta, \cos(\theta+\phi)$ and $\cos(\theta-\phi)$. We note that the stage-cost we used was tuned slightly in order to penalise the runs in which the unicycle falls over more heavily. Specifically, deviations in the vertical were penalised more than deviations in the horizontal by using $d(\mathbf{x})^2 = \frac{1}{2}\big(d_x^2 + d_y^2 + (4d_z)^2\big)$. This encodes the fact that falling over is a more serious deviation than being far away horizontally, since this is a recoverable situation.

To initialise the training data set $\mathcal{D}$, ten trials were carried out in which random torques were applied over each timestep. This resulted in the unicycle falling over so quickly that only about

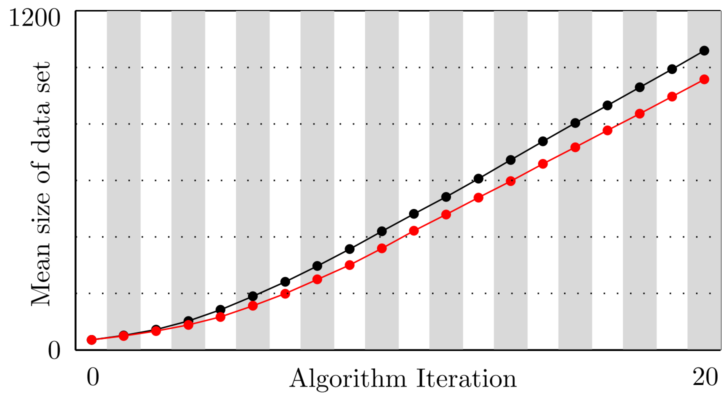


Figure 4.12: Average data set growth corresponding to the Monte Carlo runs shown in Fig. 4.11(a). All the data sets grew at around the same rate (represented by the black line) with the exception of the Euler method which grew more slowly due to its poorer performance.

3-5 data points could be obtained each time. A heuristic that was employed was in terms of the prediction horizon H. The aim was to balance the unicycle for around $T = 10\,\mathrm{s}$ or $H = \lceil 10/\delta_t \rceil$ steps. Now, in order to decrease the offline computation time required the initial horizon was set to $H_1 = \lceil 2/\delta_t \rceil$. This was then increased at each subsequent iteration according to the rule $H_{i+1} = \min\big\{H, \max\{H_i, \lceil 1.5R_i/\delta_t \rceil\}\big\}$, where R_i was the time (in seconds) for which the unicycle was able to balance after the i^{th} policy optimisation. To decrease computation time further the optimisation algorithm was limited to 60 function evaluations. Again, this figure was chosen through trial and error and general experience of the problem at hand.

The dynamics were learned using independent Gaussian process models with zero mean squared exponential kernels. The GPs were trained to map the current state $\mathbf{x}_{k-1}$ to the difference $\mathbf{x}_k - \mathbf{x}_{k-1}$, again because this is a more appropriate representation for a zero mean prior. In this case the angle variables were not given a sine/cosine representation. Due to the size of the data sets encountered, sparse approximations (based on the FITC method outlined in Sec. 3.4.5) were employed when the data sets went above 300 points and utilised a pseudo-training input set of 300 points. This value was again chosen through an experiential and trial and error procedure.

Results

The results of running the algorithm with various position reconstruction methods are shown in Fig. 4.11. No additional additive noise term was applied in these simulations since it was found that no further improvement could be obtained for this problem.

Fig. 4.11(a) shows the cost trajectory of the standard algorithm as the iterations progress. In crude terms it can be seen that most runs have converged on the “best” solution by the 14th or 15th iteration. Moving to the Euler approximation shown in Fig. 4.11(b) it is surprising to see that the algorithm can still solve the learning problem but with generally poorer performance than the standard method. The Heun method produces results comparable to that of the standard method but with arguably slightly faster convergence. Further, the number of runs

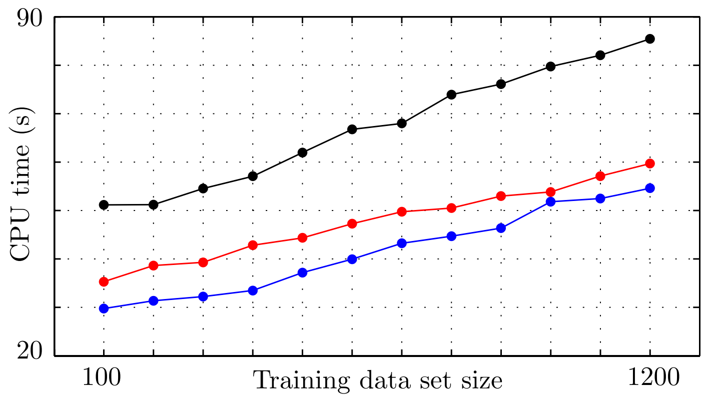


Figure 4.13: Comparison of CPU time for a single cost function evaluation $J(\psi)$ (with derivatives) for the unicycle with $H = \lceil 10/\delta_t \rceil$ and $\delta_t = 0.15\,\text{s}$. The black line is for prediction of θ and $\dot{\theta}$ with GPs. The blue line is for the case where position states are reconstructed using information back to time $k-1$ and the red line for information back to $k-2$. The x-axis depicts the size of training data set available to the GP.

that fail or incur a high cost in the last 5 iterations is much less than is observed when using the standard method.

Finally, interesting results can be observed with use of the Cubic method. This method achieves the fastest convergence to its final solution with almost all the runs completed by the 11th iteration, an improvement over all the other methods. However, the stabilising solution it has found is marginally poorer in terms of cost than the others. This can be put down to overconfidence in the predictive power of the method. We note that using velocity reconstruction led to variance explosion in the state prediction and numerically unstable results for all methods when applied to this problem and therefore learning was impossible.

Computational Savings

As mentioned in the pendulum problem, one of the most appealing aspects of including prior information is in terms of computational saving in the policy optimisation stage of the algorithm. For the unicycle this really is the bottleneck in terms of the speed of the algorithm. During the 20 iterations of the algorithm the data set usually gets up to around 1100 points. The average growth of the data sets for the Monte Carlo runs of the standard algorithm, depicted in Fig. 4.11(a), are shown by the black line in Fig. 4.12. The same growth was observed when using the Heun and the Cubic method with a slightly slower growth observed when the Euler method was employed (depicted by the red line), due to slower learning. As expected from the use of a sparse approximation the plots appear to be following a linear growth. Using the Euler or Heun methods leads to reduction of around 40% in computation while use of state information from $k-2$ leads to around a 30% reduction. These figures indicate a significant time saving since function evaluations take of the order of one minute for the standard method and therefore around one hour is needed for a single policy optimisation consisting of 60 evaluations.

4.4 Reference Tracking

4.4.1 Problem Formulation

An extension of the regulation problem given by Eq. (3.4) is now considered. In particular, the problem of forcing a linear combination of the system states $\mathbf{C}\mathbf{x}^{\mathbf{x}}$ to track some reference $\mathbf{r} = \mathbf{C}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}$ where $\mathbf{x}^{\mathbf{r}} \in \mathbb{R}^{E_r}$ is the underlying reference state. This will be achieved using a parameterised control policy $\boldsymbol{\pi}$ which has access to a subset of the full reference state $\mathbf{x}^{\mathbf{r}}_k[\mathbf{i}]$ where $\mathbf{i}$ indicates which elements the policy can use. Mathematically, this problem can be posed as

$$\text{minimise} \qquad J(\boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{k=0}^{H} c\big(\mathbf{C}\mathbf{x}^{\mathbf{x}}_k - \mathbf{C}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_k, \mathbf{u}_k\big) \,\middle|\, p(\mathbf{x}^{\mathbf{x}}_0, \mathbf{x}^{\mathbf{r}}_0)\right] \tag{4.19}$$

$$\begin{aligned} \text{subject to} \qquad \mathbf{x}^{\mathbf{x}}_k &= \mathbf{f}_{\mathbf{x}}\big(\mathbf{x}^{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}\big) \quad \text{where} \quad \mathbf{f}_{\mathbf{x}} \sim p\big(\mathbf{f}_{\mathbf{x}}|\mathcal{D}, \boldsymbol{\theta}\big) \\ \mathbf{x}^{\mathbf{r}}_k &= \mathbf{f}_{\mathbf{r}}\big(\mathbf{x}^{\mathbf{r}}_{k-1}\big) \\ \mathbf{u}_k &= \boldsymbol{\pi}\big(\mathbf{x}^{\mathbf{x}}_k, \mathbf{x}^{\mathbf{r}}_k[\mathbf{i}]\big) \end{aligned} \tag{4.20}$$

where $\boldsymbol{\tau} := [\mathbf{x}^{\mathbf{x}}_0; \mathbf{x}^{\mathbf{r}}_0; \mathbf{u}_0 \ldots \mathbf{x}^{\mathbf{x}}_H]$ is a sampled state-action and reference trajectory. The system dynamics evolve according to the function $\mathbf{f}_{\mathbf{x}}$ while the reference signal is governed by the reference generator $\mathbf{f}_{\mathbf{r}}$. We note that $\mathbf{f}_{\mathbf{r}}$ may be provided by the user or inferred from data in the same way that we infer the system dynamics.

In the spirit of Bitmead *et al.* (1990), the problem in Eqs. (4.19)–(4.20) can be recast as a regulation problem in terms of an augmented state space and dynamics model. First define the augmented state as $\mathbf{x} := [\mathbf{x}^{\mathbf{x}}; \mathbf{x}^{\mathbf{r}}]$. Next define the augmented dynamical system

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) = \begin{bmatrix} \mathbf{f}_{\mathbf{x}}\big(\mathbf{x}^{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}\big) \\ \mathbf{f}_{\mathbf{r}}\big(\mathbf{x}^{\mathbf{r}}_{k-1}\big) \end{bmatrix}$$

and the stage-cost $c^{\mathbf{r}}\big(\mathbf{x}, \mathbf{u}\big) := c\big([\mathbf{C}, -\mathbf{C}^{\mathbf{r}}]\mathbf{x}, \mathbf{u}\big)$. The recast problem can now be stated as a regulation problem in terms of the augmented state space

$$\text{minimise} \qquad J(\boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{k=0}^{H} c^{\mathbf{r}}\big(\mathbf{x}_k, \mathbf{u}_k\big) \,\middle|\, p(\mathbf{x}_0)\right] \tag{4.21}$$

$$\begin{aligned} \text{subject to} \qquad \mathbf{x}_k &= \mathbf{f}\big(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}\big) \quad \text{where} \quad \mathbf{f} \sim p\big(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta}\big) \\ \mathbf{u}_k &= \boldsymbol{\pi}\big(\mathbf{x}_k[\mathbf{j}]\big) \end{aligned} \tag{4.22}$$

where $\mathbf{j} = [\mathbf{1}; \mathbf{i}]$ is the new indexing vector. This is in exactly the same form as the original regulation problem posed in Eqs. (3.2)–(3.3). Note that if the moments of the stage-cost c are tractable given a Gaussian input $\mathbf{x}, \mathbf{u} \sim \mathcal{N}$ then so will the moments of $c^{\mathbf{r}}$ since the state has simply undergone a linear transformation and therefore remains Gaussian.

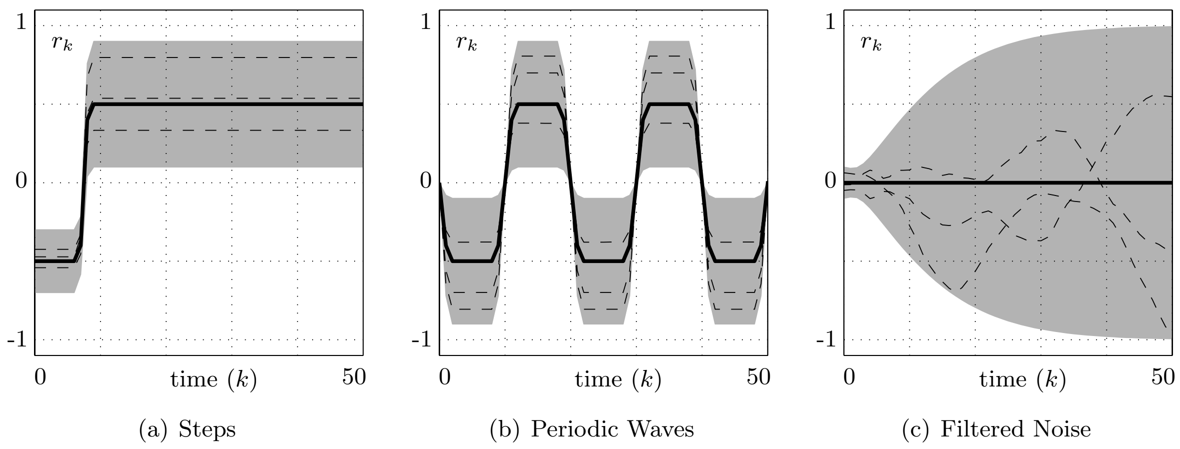


Figure 4.14: Examples of reference signals generated from the linear system $\mathbf{x}^{\mathbf{r}}_k = \mathbf{A}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_{k-1} + \mathbf{b}^{\mathbf{r}}e_{k-1}$ where $r_k = [1, \mathbf{0}]\mathbf{x}^{\mathbf{r}}$. Plots (a)-(c) were generated using $\mathbf{A}^{\mathbf{r}}_1, \mathbf{b}^{\mathbf{r}}_1$, $\mathbf{A}^{\mathbf{r}}_2, \mathbf{b}^{\mathbf{r}}_2$ and $\mathbf{A}^{\mathbf{r}}_3, \mathbf{b}^{\mathbf{r}}_3$ respectively. The means are plotted in thick solid lines, samples shown by dashed lines and the 2σ confidence region shaded in grey.

4.4.2 Preview Horizon

An important point to make is that the problem given in Eqs. (4.19)–(4.20) can handle the situation in which the control policy is allowed to anticipate the impending change in reference by having access to some *preview horizon* of the reference signal. In other words $\mathbf{u}_k$ is allowed to be functionally dependent on $\mathbf{r}_{k+i}$ for $i \in \mathbb{Z}_{[0,H_\mathrm{p}]}$ and a preview horizon of H_p steps. This situation can be encoded by defining the reference state to consist of future values of $\mathbf{r}$ over the preview horizon window $\mathbf{x}^{\mathbf{r}}_k = [\mathbf{r}_k; \mathbf{r}_{k+1} \dots \mathbf{r}_{k+H_\mathrm{p}}]$. Then defining the reference dynamics to consist of

$$\mathbf{x}^{\mathbf{r}}_{k+1} = \mathbf{f}_{\mathbf{r}}\big(\mathbf{x}^{\mathbf{r}}_k\big) = \begin{bmatrix} [\mathbf{0}\ \ \mathbf{I}]\mathbf{x}^{\mathbf{r}}_k \\ \mathbf{f}_r(\mathbf{r}_{k+H_\mathrm{p}}) \end{bmatrix}$$

the problem is back into the form outlined in the previous section. This reference tracking work was originally proposed in Hall *et al.* (2011).

4.4.3 Reference Dynamics

When considering what kind of reference dynamics model to use it is worth noting that many standard reference signals can be generated using a simple linear system of equations. Some examples are given in Fig. 4.14. These were generated using $\mathbf{x}^{\mathbf{r}}_k = \mathbf{A}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_{k-1} + \mathbf{b}^{\mathbf{r}}e_{k-1}$ where $r_k = [1, \mathbf{0}]\mathbf{x}^{\mathbf{r}}$, $\mathbf{x}^{\mathbf{r}} \sim \mathcal{N}$ and $e_k \sim \mathcal{N}(0, 1)$. In particular Fig. 4.14(a-c) were generated using

$$\mathbf{A}^{\mathbf{r}}_1 = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ 0 & [\mathbf{0}\ \ 1] \end{bmatrix} \in \mathbb{R}^{10}, \quad \mathbf{A}^{\mathbf{r}}_2 = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -1 & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{10}, \quad \mathbf{A}^{\mathbf{r}}_3 = \begin{bmatrix} a & b \\ 0 & a \end{bmatrix} \in \mathbb{R}^{2}$$

and $\mathbf{b}^{\mathbf{r}}_1 = \mathbf{b}^{\mathbf{r}}_2 = \mathbf{0}$, $\mathbf{b}^{\mathbf{r}}_3 = [0; c]$ respectively, along with an appropriate distribution over the start state $\mathbf{x}^{\mathbf{r}}_0$. The constants a, b and c were chosen such that $\mathrm{cov}[r_k] \to \left(\frac{1}{2}\right)^2$ as $k \to \infty$ where $|a| < 1$ is necessary for a stable generator.

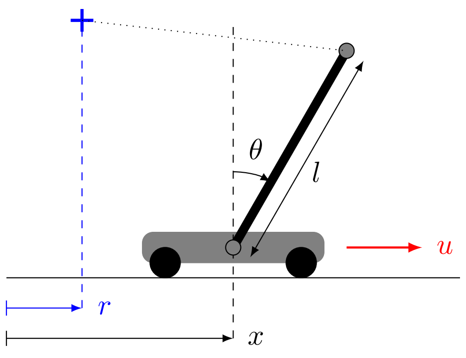


Figure 4.15: Cart-pole setup with position x, pole angle θ, pole length l, action force u and lateral location of the setpoint r. The setpoint is shown by the blue cross and the Euclidean distance from the tip of the pole to the setpoint is shown by the dotted line.

We note that an advantage of the learning framework is that the reference generating system could also be inferred directly from the data, in the same way that standard dynamics are learned. For example, if the reference signals for a system came from a human operator then a Gaussian process model could be trained to imitate the trajectories the operator is producing and used for offline simulation.

4.4.4 Example: Cart-pole

Setup

For illustrative purposes, consider the problem of tracking a moving setpoint using the cart-pole set up in Fig. 4.15. The equations of motion and constant values for this system can be found in Appendix B.2. The tracking task is to follow a moving x, denoted r, position with the pole balanced upright. The initial state of the system is at the origin $\mathbf{x}_0 = \mathbf{0}$. The stage-cost is in the same form as the unicycle cost in Eq. (4.18) in which $c(\mathbf{x}) = 1 - \exp\big(-\frac{1}{2}d(\mathbf{x})^2\big)$ where $d(\mathbf{x})^2$ is the squared distance from the setpoint. The distance, in this case can be decomposed as $d(\mathbf{x})^2 = d_x^2 + d_z^2$ where

$$\begin{aligned} d_x &= (x - r) + \sin\theta \\ d_z &= \cos\theta \end{aligned}$$

The prediction horizon was set to $T = 5\,\mathrm{s}$ with a discretisation time of $\delta_t = 0.1\,\mathrm{s}$ leading to $H = 50$ steps. The trials at each iteration of the algorithm were stopped if the pole fell. The Gaussian process model trained to learn the dynamics was initialised with five trial runs in which random inputs were applied. This led to an overall experience of around 1.5 s. Four independent GPs with squared exponential covariance functions were used to learn the dynamics. Finally, the structure of the policy was linear and pushed through the standard saturation block to satisfy the input constraints.

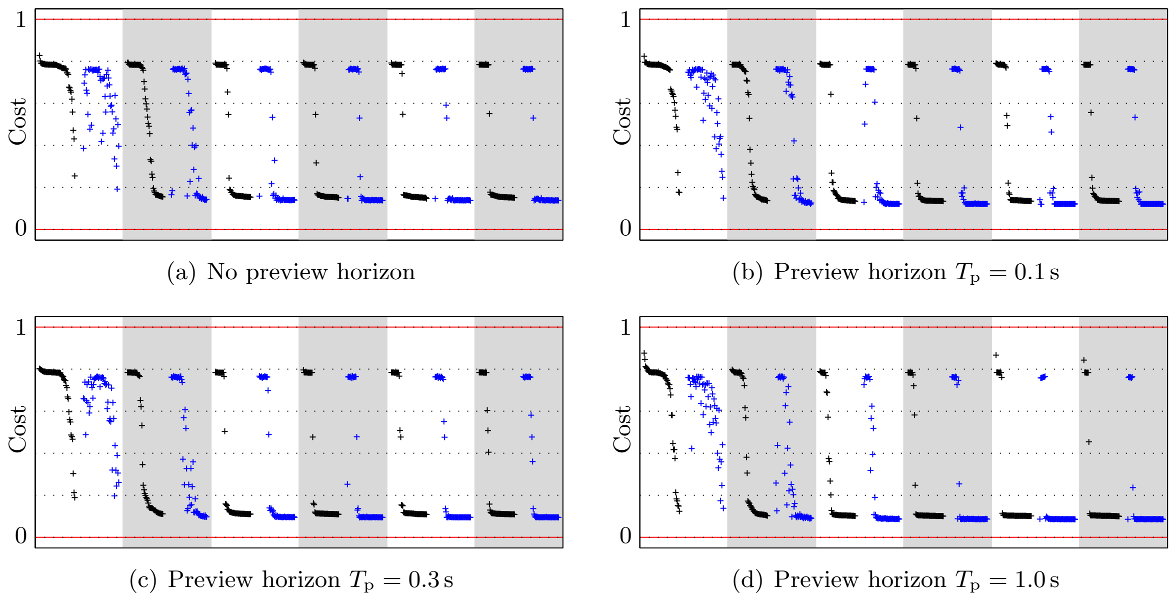


Figure 4.16: Cost incurred by the current policy after a given iteration of the algorithm. The policy has access to different preview horizons T_p of the reference. Black crosses show the actual cost incurred while the blue shows the mean predicted cost.

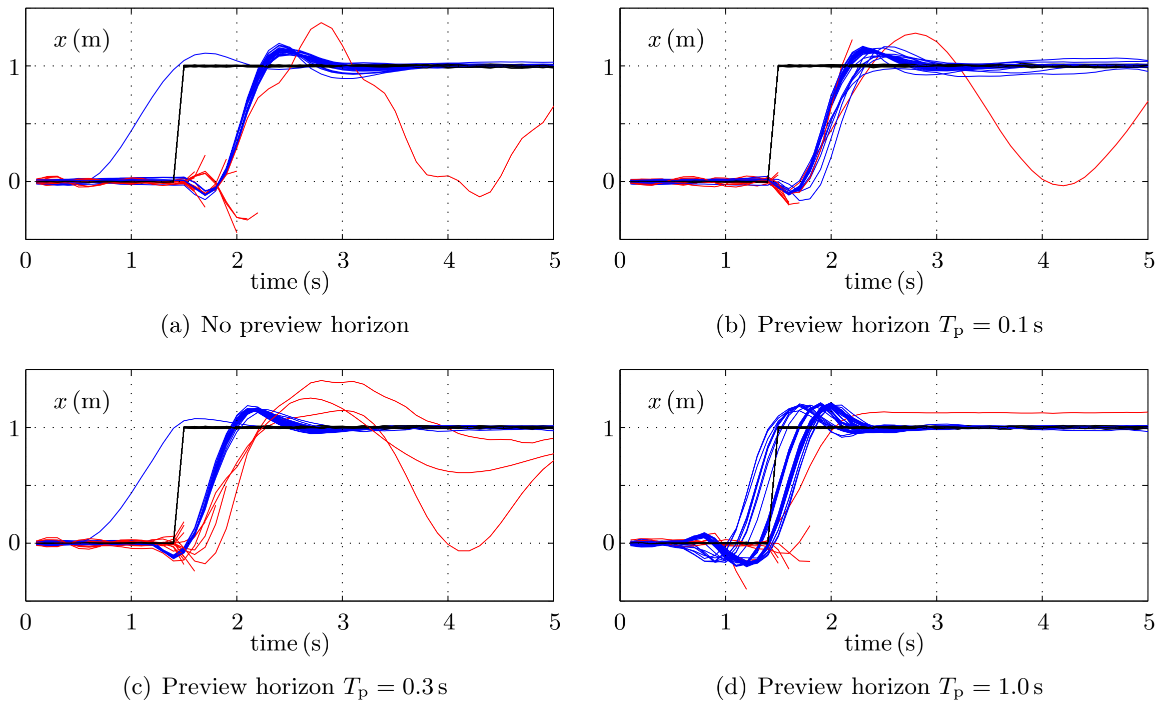
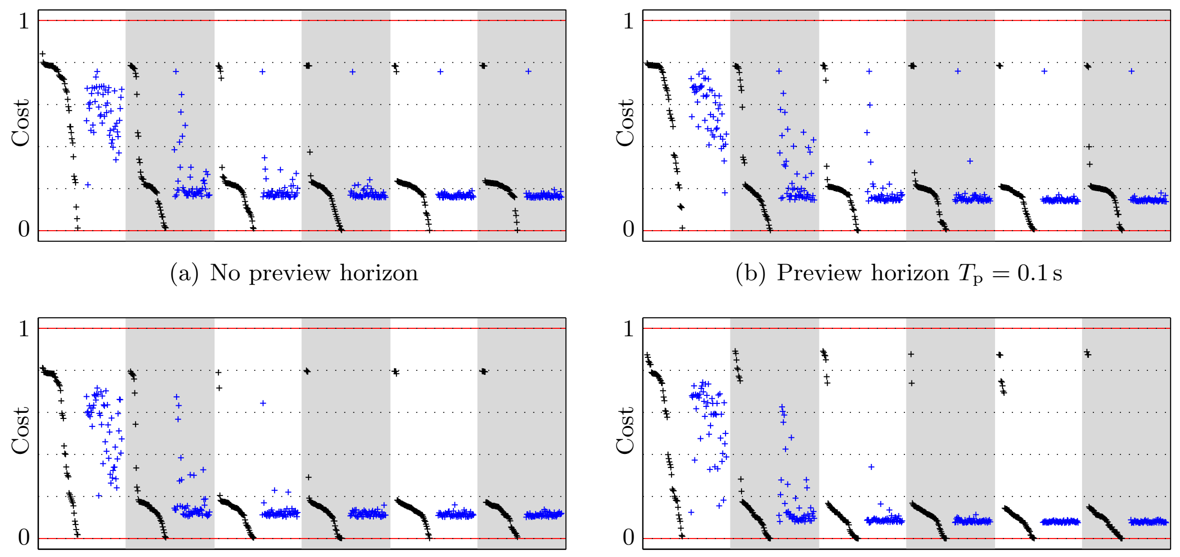


Figure 4.17: Responses of the cart-pole system corresponding to the costs depicted in Fig. 4.16 at the sixth iteration. The policy preview horizon T_p was altered between these four sets of runs. The runs show the step responses of 50 runs after the 10th iteration of the algorithm. Red lines show runs in which the pole falls over or can balance but have not learned to track yet.



(a) No preview horizon

(b) Preview horizon $T_\mathrm{p} = 0.1\,\mathrm{s}$

(c) Preview horizon $T_\mathrm{p} = 0.3\,\mathrm{s}$

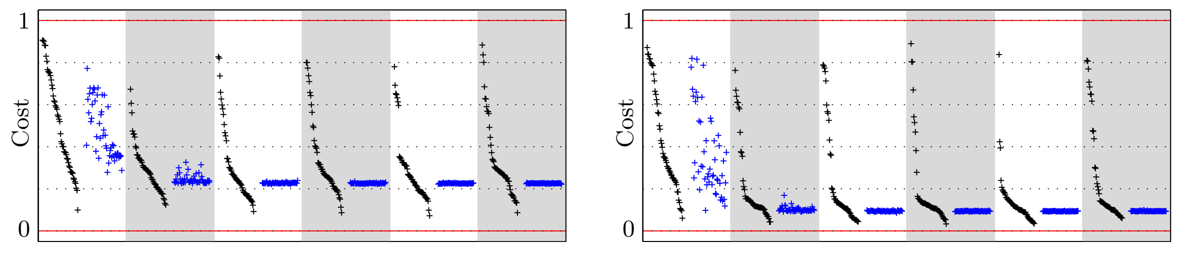
(d) Preview horizon $T_\mathrm{p} = 1.0\,\mathrm{s}$

Figure 4.18: Cost incurred by the algorithm by applying the current best policy after a given iteration (given by the x-axis). Black crosses show the actual cost incurred while the blue shows the predicted cost.

Results

The first task the cart-pole system was tasked with was to learn a policy to track a unit step change in position x occurring at 1.5 s. The available preview horizon T_p was varied from 0 to 1 s to investigate the effect of allowing the policy to anticipate the change in setpoint. The predicted and actual costs of 50 Monte Carlo runs over 6 algorithm iterations are shown in Fig. 4.16. It is clear that many runs reach their solution after only two iterations while most finish learning after just three. Further note the decrease in cost of the best solution as the preview horizon is increased. This demonstrates that, as we would expect, being able to anticipate the change is beneficial. Note that there is a failure mode that a significant number of the runs end up in. In this mode the pole is balanced at the origin until the change in setpoint then the policy allows it to fall over! There are also a couple of runs that can keep the pole balanced but have not yet learned to track the setpoint.

The actual solutions to which the algorithm converged are given in Fig. 4.17. The blue lines correspond to runs that kept the pole upright while the red lines correspond to the failure mode and runs that did not keep the pole balanced. Note that as the preview horizon is increased the benefit of being able to anticipate the change in setpoint is clear. An interesting artefact from the $T_\mathrm{p} = 1.0\,\mathrm{s}$ runs is that there are two optimal solutions, one in which the cartpole moves so that it is in position at the new setpoint when it changes and one in which it is ready to move as the setpoint changes. Another interesting result from using no preview horizon or $T_\mathrm{p} = 0.3\,\mathrm{s}$ can be observed in Fig. 4.16(a) and (c). In these cases the policy has chosen to ignore the setpoint reference signal and simply regulate to the $x = 1\,\mathrm{m}$ position. This behaviour results from only considering a single setpoint change rather than a distribution of possible changes.



(a) Access only to the reference position (b) Access to the full reference state.

Figure 4.19: Cost incurred by the algorithm by applying the current best policy after a given iteration (given by the x-axis). Black crosses show the actual cost incurred while the blue shows the predicted cost.

We can of course optimise over a distribution over trajectories as shown in Fig. 4.14. The next set of simulations was trained using a step change model that changed at the same time as before but the new setpoint was sampled from the distribution $\mathcal{N}(0, 1)$. The results of learning a control policy to achieve this task are shown in Fig. 4.18. The first thing to note is that the resulting spread in the actual costs as learning proceeds is not emulated in the predicted costs because the predictions are of the average cost that will be incurred. Bearing this in mind, the predictions match up well with the actual performance. We can again see a significant improvement in performance as the preview horizon is increased.

The most notable feature of these results is that we no longer fall into the same failure models that we did when we considered only a deterministic change in setpoint. With the exception of two, all these runs learned how to track a given setpoint. The only runs that failed were the ones that had to track setpoint changes greater than around 2 m. This makes sense since this is the 2σ point for the setpoint distribution.

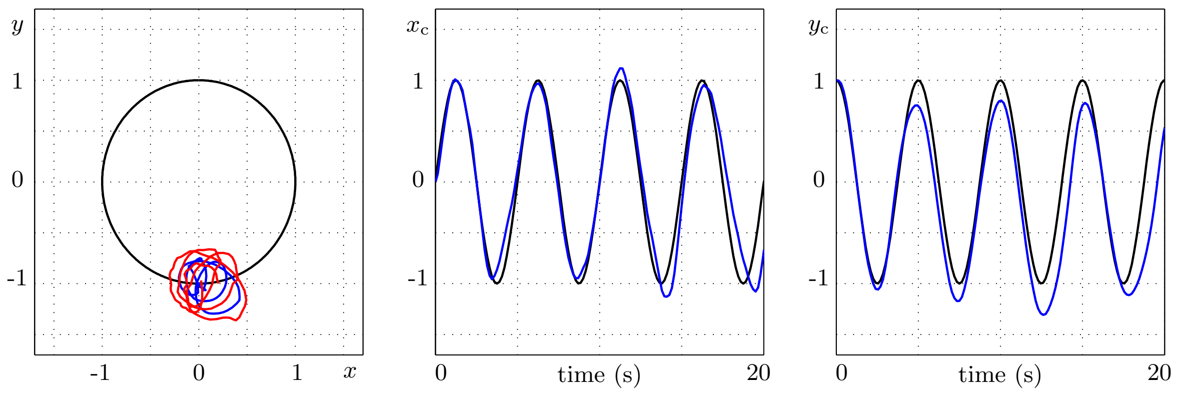
The final set of simulations we carried out was to track references generated by the second order filter depicted in Fig. 4.14(c). In particular, the reference system we considered was given by

$$\mathbf{x}^{\mathbf{r}}_{k+1} = \begin{bmatrix} 0.92 & 2.22 \\ 0 & 0.92 \end{bmatrix} \mathbf{x}^{\mathbf{r}}_{k} + \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} e_k$$

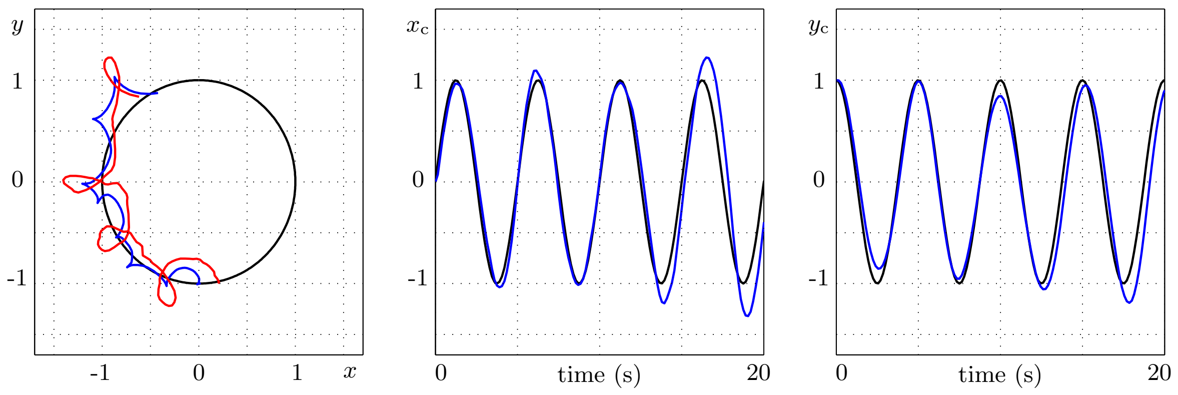
with $\mathbf{x}^{\mathbf{r}}_0 = \mathbf{0}$. In the limit of $k \to \infty$ the standard deviation of the position term saturates to 0.5 m. We first gave the policy access only to the positional term and the results are shown in Fig. 4.19(a). Then we gave the policy access to the full reference state and obtained the results in Fig. 4.19(b). These results clearly show that the learning procedure has exploited the additional information to obtain much improved tracking performance.

4.4.5 Example: Unicycle

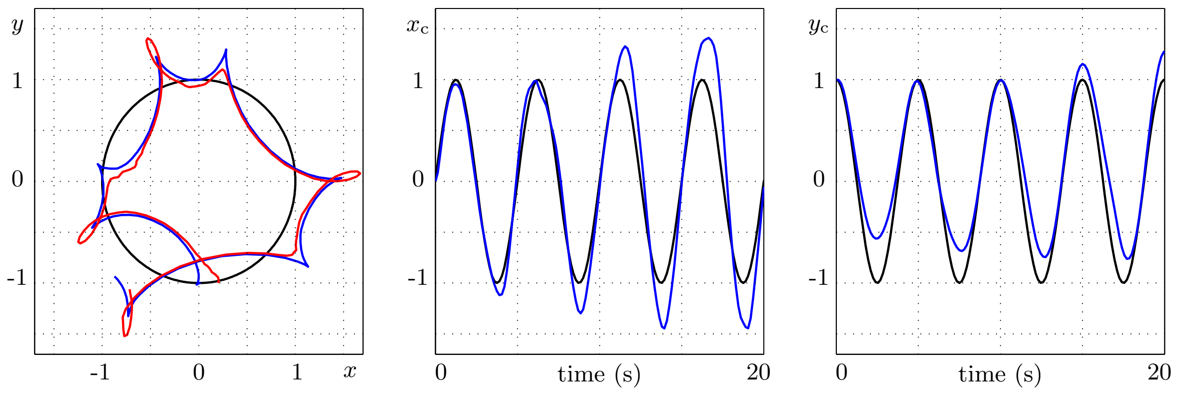
We now turn our attention back to the unicycle problem of Sec. 4.3.4. We were unsuccessful in getting the unicycle to track specific references in an absolute coordinate system (x, y) however we did succeed in tracking references defined in the body-fixed coordinate system (x_c, y_c) shown



(a) Spinning on the spot



(b) Spin whilst moving slowly round the circle



(c) Spin whilst moving rapidly round the circle

Figure 4.20: Plots showing the movement of the base of the balanced unicycle as it tracks a spinning setpoint defined in the body-fixed coordinate system (x_c, y_c). The left hand plots show the actual spatial trajectory of the base of the unicycle in blue whilst the red line shows the trajectory of the front of the wheel in order to demonstrate the spinning motion. The other plots show the reference in black and the relevant state of the unicycle in blue.

in Fig. 4.10. Our reasoning for this is that tracking trajectories in the body-fixed coordinate system could be achieved through an infinite number of possible trajectories in an absolute frame of reference. This is because this coordinate system is invariant to rotations around the global origin. Therefore tracking in body-fixed coordinates is less constraining and less challenging than tracking an equivalent fixed trajectory in an absolute frame. This point will be illustrated more clearly with an example.

We tasked the unicycle to stay on the unit circle in the absolute coordinate system (x, y). In order to do this using absolute coordinates we had to define a reference state space system

$$\begin{bmatrix} x_k^{\mathrm{r}} \\ y_k^{\mathrm{r}} \end{bmatrix} = \begin{bmatrix} \cos\left(\omega_{\mathrm{o}}\delta_t\right) & \sin\left(\omega_{\mathrm{o}}\delta_t\right) \\ -\sin\left(\omega_{\mathrm{o}}\delta_t\right) & \cos\left(\omega_{\mathrm{o}}\delta_t\right) \end{bmatrix} \begin{bmatrix} x_{k-1}^{\mathrm{r}} \\ y_{k-1}^{\mathrm{r}} \end{bmatrix}$$

where $\omega_{\mathrm{o}} = 2\pi/T_{\mathrm{o}}$ with orbital period T_{o} and we set the initial state $(x_0^{\mathrm{r}}, y_0^{\mathrm{r}}) = (0, 1)$. The algorithm failed to learn this task for multiple values of T_{o}. It would either balance at the origin or follow an orbital trajectory of random radius, which are local minima of the problem. Even when we set T_{o} to the value of one of these random orbits the algorithm still failed. However, the problem could be posed in a general way using the body-fixed coordinates $(x_{\mathrm{c}}, y_{\mathrm{c}})$ and simply defining a "setpoint" of $(x_{\mathrm{c}}^{\mathrm{r}}, y_{\mathrm{c}}^{\mathrm{r}}) = (0, 1)$. This "setpoint" defines all locations of the unicycle for which the origin is 1 m from its right hand side. Under this setup the algorithm could learn this task slightly quicker than the task of balancing.

After this investigation we defined a more exotic set of trajectories using the reference state space system in body-fixed coordinates given by

$$\begin{bmatrix} x_{\mathrm{c},k}^{\mathrm{r}} \\ y_{\mathrm{c},k}^{\mathrm{r}} \end{bmatrix} = \begin{bmatrix} \cos\left(\omega_{\mathrm{o}}\delta_t\right) & \sin\left(\omega_{\mathrm{o}}\delta_t\right) \\ -\sin\left(\omega_{\mathrm{o}}\delta_t\right) & \cos\left(\omega_{\mathrm{o}}\delta_t\right) \end{bmatrix} \begin{bmatrix} x_{\mathrm{c},k-1}^{\mathrm{r}} \\ y_{\mathrm{c},k-1}^{r} \end{bmatrix}$$

If we now set $(x_{\mathrm{c},0}^{\mathrm{r}}, y_{\mathrm{c},0}^{\mathrm{r}}) = (0, 1)$, this tasks the unicycle to stay on the unit circle but to spin around with a period of T_{o}. Three of the solutions the algorithm learned for $T_{\mathrm{o}} = 5\,\mathrm{s}$ are shown in Fig. 4.20. These plots clearly show the variety of solutions available to achieve the same task. The policy guiding the unicycle in Fig. 4.20(a) simply spins it on the spot, whereas the policies shown in (b) and (c) also travel around the unit circle, at varying speeds. We note that very few of the learned control policies could keep tracking beyond 20 s. This issue highlights one of the problems with only training on a relatively short prediction horizon, in this case 10 s. We did not attempt greater prediction horizons because of the increased computation time required to do this.

Finally, through careful selection of reference generator systems we can train the unicycle to track a figure-of-eight trajectory. For example, this can be achieved using the following state

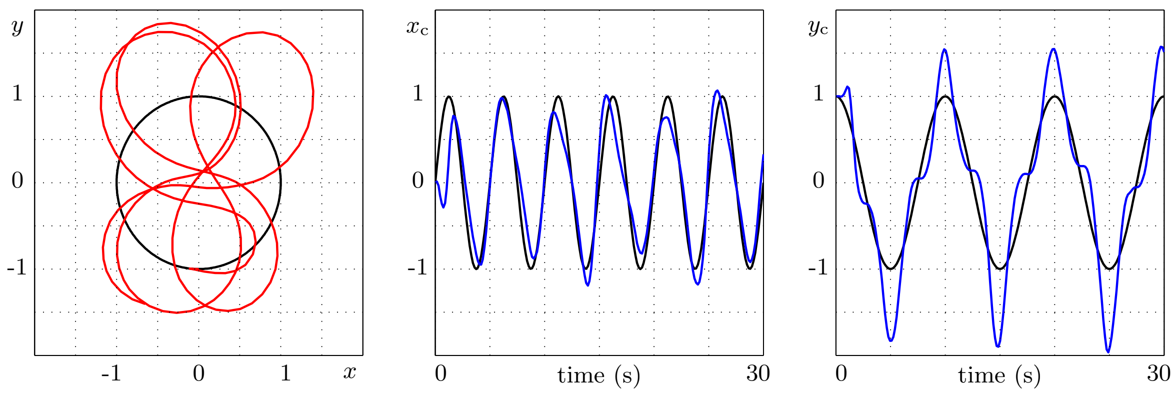


Figure 4.21: Plots showing the movement of the base of the balanced unicycle as it tracks a figure-of-eight in the body-fixed coordinate system $(x_\mathrm{c}, y_\mathrm{c})$. The left hand plots show the actual spatial trajectory of the base of the unicycle in red and the unit circle in black. The other plots show the reference trajectory in black and the relevant state of the unicycle in blue.

space system

$$\mathbf{x}^\mathbf{r}_+ = \begin{bmatrix} \cos\left(2\omega_\mathrm{o}\delta_t\right) & \sin\left(2\omega_\mathrm{o}\delta_t\right) & 0 & 0 \\ -\sin\left(2\omega_\mathrm{o}\delta_t\right) & \cos\left(2\omega_\mathrm{o}\delta_t\right) & 0 & 0 \\ 0 & 0 & \cos\left(\omega_\mathrm{o}\delta_t\right) & \sin\left(\omega_\mathrm{o}\delta_t\right) \\ 0 & 0 & -\sin\left(\omega_\mathrm{o}\delta_t\right) & \cos\left(\omega_\mathrm{o}\delta_t\right) \end{bmatrix} \mathbf{x}^\mathbf{r}$$

$$\begin{bmatrix} x^\mathrm{r}_{\mathrm{c},k} \\ y^\mathrm{r}_{\mathrm{c},k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}^\mathbf{r}$$

with initial reference state $\mathbf{x}^\mathbf{r}_0 = [0, 1, 0, 1]^\top$. With this setup x^r_c varies as a sine wave while y^r_c varies as a cosine wave with twice the period. This was clearly a very difficult problem since we were only able to find one training run, from many attempts, that actually managed to learn a policy which achieved the task. The policy learned after 20 iterations of the algorithm was then run for 30 s and the resulting trajectory is shown in Fig. 4.21. Again, we can see the rotational invariance property of the body-fixed coordinate system as the figure-of-eight is not fixed in absolute coordinates.

4.5 Summary

Prior knowledge about a dynamical system often comes in the form of known, or approximate, relationships between subsets of the state variables. We have outlined a method for incorporating this prior knowledge into the general framework of probabilistic learning control outlined in the previous chapter. To achieve this we view the system state as evolving according to multiple dynamics models, or sub-dynamics. The key contribution was then in how to propagate the uncertainty in the predictions, which employed the use of a slightly more restrictive assumption of joint Gaussianity than the assumptions made in the previous chapter.

This method found its most obvious use in encoding position-velocity relationships between state variables and in the modelling of higher order Markov systems. However, we demonstrated that it can also be used to tackle problems of reference tracking in which the system is tasked to regulate to a moving setpoint. In addition to this we can allow the policy to have access to a preview horizon of the reference state and therefore allow the algorithm to learn how best to anticipate a change of setpoint.

We were also interested in the question of how the incorporation of such information would affect the learning speed of the algorithm, both computationally and convergence to a good policy, and the propensity for finding good (or bad) local minima in a given problem. This was examined for the pendulum swing-up and unicycle balancing tasks where we included various approximate models for the position-velocity relationships. As expected, we found that a crude model led to poorer results. However, we found that more elaborate models did not necessarily lead to better solutions. We also noticed that although sometimes the propensity for finding poor locally optimal solutions was decreased in many cases, it was amplified in others, in particular when reconstructing velocity from predictions of the position!

Finally, we demonstrated the capability of the framework to learn reference tracking policies on a simulation of a cart and pole. The policy was able to exploit any preview horizon we gave it to help it anticipate the change. We further noted that learning to track a distribution of possible reference signals led to improved learning performance in terms of avoiding poor local minima. Reference tracking was also shown to work on the unicycle when the reference signal was defined in the body-fixed coordinate system rather than absolute coordinates.

CHAPTER 5

Priors Over Sampled Systems

5.1 Introduction

Augmented dynamics provides a flexible way of incorporating useful forms of prior knowledge. However, the most intuitive place to encode such information in a Bayesian framework is directly in the prior distribution over dynamics functions. In fact, augmented dynamics could be viewed as an implication on the covariance matrix or adding additional structure to it. The form of prior knowledge that we address in this chapter is the notion that our discrete-time dynamical system is, in many cases, a sampled continuous-time system. How can we define a prior over discrete-time dynamics that encodes this assumption?

We must first ask, what is the motivation for defining such a prior? What will the likely savings be? In answer to former, we note that it is often the case that the continuous-time dynamics contain some nice structure, for example they can be decomposed as a sum of a linear part, a nonlinear additive part and a general nonlinear part. For example, the pendulum dynamics are made up of a linear and nonlinear part (see Appendix B.1) and the cart-pole dynamics consist of a sum of nonlinear combinations of pairs of variables (see Appendix B.2). However, this special structure gets lost when we move to the associated discrete-time dynamics. If we were able to encode or learn this underlying structure then our model would be better equipped to generalise into new areas of the state space. In answer to the latter, the likely savings would be that the trained model should have greater marginal likelihood and its predictive performance should be improved with respect to a naïve treatment of the unknown dynamics as a general nonlinear function.

An obvious example of this is, again, time-derivative related states, in which the exact relationship is clear when written in continuous-time but it becomes a complex function of the other (unknown) dynamics, states and actions when dealing in discrete-time. In general any nice structure in the continuous-time dynamics does not translate into the discrete-time domain. It is this problem that we will tackle through the definition of a new class of priors.

5.2 Problem Outline

In this section we shall address sampled continuous-time systems. These are discrete-time systems which are obtained by sampling some underlying continuous-time system given by

$$\dot{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) \tag{5.1}$$

where $t \in \mathbb{R}$. We shall restrict ourselves, as before, to the consideration of stationary dynamics where $\mathbf{f}$ is only dependent on t through $\mathbf{x}$ and $\mathbf{u}$. Now assume that $\mathbf{f}$ has been drawn from some Gaussian process prior $\mathbf{f} \sim \mathcal{GP}(\mathbf{m}, \mathbf{K})$. Or in other words

$$\begin{aligned}\mathbf{m}(\mathbf{z}_t) &= \mathbb{E}_{\mathbf{f}}\big[\mathbf{f}(\mathbf{z}_t)\big]\\ \mathbf{K}(\mathbf{z}_t, \tilde{\mathbf{z}}_\tau) &= \mathrm{cov}_{\mathbf{f}}\big[\mathbf{f}(\mathbf{z}_t), \mathbf{f}(\tilde{\mathbf{z}}_\tau)\big]\end{aligned}$$

remembering that $\mathbf{z} := [\mathbf{x}; \mathbf{u}]$. This GP prior can encode prior knowledge we have regarding the continuous-time dynamics e.g. invariances, linear relationships and additive nonlinear relationships. We are interested in evaluating the associated prior over a sampled version of the dynamics $\mathbf{f}_\Delta \sim \mathcal{GP}(\mathbf{m}_\Delta, \mathbf{K}_\Delta)$ where

$$\begin{aligned}\mathbf{m}_\Delta(\mathbf{z}_t) &= \mathbb{E}_{\mathbf{f}}\big[\mathbf{f}_\Delta(\mathbf{z}_t)\big]\\ \mathbf{K}_\Delta(\mathbf{z}_t, \tilde{\mathbf{z}}_\tau) &= \mathrm{cov}_{\mathbf{f}}\big[\mathbf{f}_\Delta(\mathbf{z}_t), \mathbf{f}_\Delta(\tilde{\mathbf{z}}_\tau)\big]\end{aligned}$$

Note that the expectation and covariance are still taken with respect to the continuous-time dynamics. Now the actual relationship between continuous and the sampled functions is

$$\mathbf{x}_{t+\delta_t} = \mathbf{f}_\Delta(\mathbf{x}_t, \mathbf{u}_t) := \mathbf{x}_t + \int_t^{t+\delta_t} \mathbf{f}(\mathbf{x}_\tau, \mathbf{u}_\tau)\mathrm{d}\tau \tag{5.2}$$

with sample time δ_t. We want to find the relationship between the continuous-time mean and covariance functions $(\mathbf{m}, \mathbf{K})$ and the associated sampled functions $(\mathbf{m}_\Delta, \mathbf{K}_\Delta)$. Inevitably we shall have to be satisfied with an approximation since this integral is in general analytically intractable (with the notable exception of linear systems).

In addition, it would be useful to evaluate the posterior GP when we are given non-uniformly

	Euler	**Midpoint**	**Heun**	**Kutta**	**RK4**
a_{ip}	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\ \frac{1}{2} & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 0\\ \frac{1}{2} & 0 & 0\\ -1 & 2 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 0 & 0\\ \frac{1}{2} & 0 & 0 & 0\\ 0 & \frac{1}{2} & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}$
b_i	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}0 & 1\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{6} & \frac{2}{3} & \frac{1}{6}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}\end{bmatrix}$

Table 5.1: Settings of a_{ip} and b_i for some common Runge-Kutta methods.

sampled data $\mathcal{D} = \{\mathbf{Z}, \mathbf{Y}\}$

$$\mathbf{Z} = \begin{bmatrix}\mathbf{x}_{t_1}^\top & \mathbf{u}_{t_1}^\top \\ \vdots & \vdots \\ \mathbf{x}_{t_n}^\top & \mathbf{u}_{t_n}^\top\end{bmatrix} \in \mathbb{R}^{n\times D} \qquad \text{and} \qquad \mathbf{Y} = \begin{bmatrix}\mathbf{x}_{t_1+\delta_1}^\top \\ \vdots \\ \mathbf{x}_{t_n+\delta_n}^\top\end{bmatrix} \in \mathbb{R}^{n\times E} \tag{5.3}$$

where the δ_i can be different. Finally, in order to incorporate it into the predictive framework, moment matching must be possible. We now consider a novel solution to this problem using an approximation based on numerical integration techniques.

5.3 Numerical Integration

5.3.1 Taylor-Series Expansion

For clarity of presentation we shall restrict our attention to the autonomous case in which $\dot{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_t)$. The full non-autonomous case is addressed in Sec. 5.4.3. One way of dealing with the intractable integral in Eq. (5.2) is by expanding out the dynamics function according to its Taylor-series. A first order expansion would yield

$$\begin{aligned}\mathbf{f}(\mathbf{x}_{t+\tau}) &\approx \mathbf{f}(\mathbf{x}_t) \;+\; \tau \frac{\mathrm{d}\mathbf{f}(\mathbf{x}_{\tilde{\tau}})}{\mathrm{d}\tilde{\tau}}\bigg|_{\tilde{\tau}=t} \\ &= \mathbf{f}(\mathbf{x}_t) \;+\; \tau \mathbf{f}_{\mathbf{x}}(\mathbf{x}_t)\dot{\mathbf{x}}_t \\ &= \mathbf{f}(\mathbf{x}_t) \;+\; \tau \mathbf{f}_{\mathbf{x}}(\mathbf{x}_t)\mathbf{f}(\mathbf{x}_t)\end{aligned}$$

where $\mathbf{f}_{\mathbf{x}}(\mathbf{x}_t) := \frac{\mathrm{d}\mathbf{f}(\mathbf{x})}{\mathrm{d}\mathbf{x}}\big|_{\mathbf{x}=\mathbf{x}_t} \in \mathbb{R}^{E\times E}$. The relationship given by Eq. (5.2) can then be rewritten as

$$\mathbf{f}_\Delta(\mathbf{x}_t) \approx \mathbf{x}_t + \delta_t \mathbf{f}(\mathbf{x}_t) + \tfrac{1}{2}\delta_t^2 \mathbf{f}_{\mathbf{x}}(\mathbf{x}_t)\mathbf{f}(\mathbf{x}_t) \tag{5.4}$$

Therefore the approximate relationships between $(\mathbf{m}_\Delta, \mathbf{K}_\Delta)$ and $(\mathbf{m}, \mathbf{K})$ can be characterised by

$$\begin{aligned}\mathbf{m}_\Delta(\mathbf{x}_t) &\approx \mathbf{x}_t + \mathbb{E}_\mathbf{f}\left[\delta_t \mathbf{f}(\mathbf{x}_t) + \tfrac{1}{2}\delta_t^2 \mathbf{f}_\mathbf{x}(\mathbf{x}_t)\mathbf{f}(\mathbf{x}_t)\right]\\ \mathbf{K}_\Delta(\mathbf{x}_t, \tilde{\mathbf{x}}_\tau) &\approx \mathrm{cov}_\mathbf{f}\left[\delta_t \mathbf{f}(\mathbf{x}_t) + \tfrac{1}{2}\delta_t^2 \mathbf{f}_\mathbf{x}(\mathbf{x}_t)\mathbf{f}(\mathbf{x}_t),\ \delta_\tau \mathbf{f}(\tilde{\mathbf{x}}_\tau) + \tfrac{1}{2}\delta_\tau^2 \mathbf{f}_\mathbf{x}(\tilde{\mathbf{x}}_\tau)\mathbf{f}(\tilde{\mathbf{x}}_\tau)\right]\end{aligned}$$

Note that δ_t need not be the same as δ_τ. There is an issue of prohibitive algebraic complexity, even for this simple approximation. It would involve the calculation of high order moments such as $\mathrm{cov}_\mathbf{f}\left[\mathbf{f}_\mathbf{x}(\mathbf{x}_t)\mathbf{f}(\mathbf{x}_t),\ \mathbf{f}_\mathbf{x}(\tilde{\mathbf{x}}_\tau)\mathbf{f}(\tilde{\mathbf{x}}_\tau)\right]$ in which it is not clear how this relates to the functions $\mathbf{m}$ and $\mathbf{K}$. Therefore we turn our attention to a widely used and iterative form of numerical integration, the Runge-Kutta (RK) family of methods.

5.3.2 Runge-Kutta Methods

Runge-Kutta methods consist of a simple set of iterative equations which calculate the Taylor-series approximation exactly for a given order S. These can be written in the general form

$$\mathbf{f}_\Delta(\mathbf{x}_t) = \mathbf{x}_t + \delta_t \sum_{i=1}^{S} b_i \mathbf{f}(\mathbf{p}_i) \tag{5.5}$$

$$\mathbf{p}_i = \mathbf{x}_t + \delta_t \sum_{p=1}^{i-1} a_{ip} \mathbf{f}(\mathbf{p}_p) \tag{5.6}$$

for stationary dynamics where a_{ip}, b_i and c_i are the constants associated with a given method. These constants are subject to the constraint $\sum_i b_i = 1$ if the method is to be consistent. By way of example, the constants for some common Runge-Kutta methods are given in Table 5.1. These are Euler's method, the Midpoint method, Heun's method, Kutta's method and the popular Runge-Kutta fourth order method (RK4). Now under this class of methods for numerical integration we can form a tractable approximate relationship from continuous-time to discrete-time as we shall now describe.

5.4 Priors Over Sampled Continuous-Time Systems

5.4.1 Illustrative Examples

In order to find the relationship between $\mathbf{m}$, $\mathbf{K}$ and $\mathbf{m}_\Delta$, $\mathbf{K}_\Delta$ consider an approximation using a Runge-Kutta method, given by Eqs. (5.5)–(5.6). It is useful at this stage to work through the procedure for a couple of simple methods before moving onto the general framework. If we consider the simplest case of an Euler scheme we have the relationship

$$\mathbf{f}_\Delta(\mathbf{x}) = \mathbf{x} + \delta_t \mathbf{f}(\mathbf{x})$$

It is clear that in this instance the sampled mean and covariance function are simply scalings and translations of the continuous mean and covariance functions

$$\begin{aligned} \mathbf{m}_\Delta(\mathbf{x}) &= \mathbb{E}_\mathbf{f}[\mathbf{f}_\Delta(\mathbf{x})] \\ &= \mathbf{x} + \delta_t \mathbb{E}_\mathbf{f}[\mathbf{f}(\mathbf{x})] \\ &= \mathbf{x} + \delta_t \mathbf{m}(\mathbf{x}) \end{aligned} \qquad \begin{aligned} \mathbf{K}_\Delta(\mathbf{x}, \tilde{\mathbf{x}}) &= \mathrm{cov}_\mathbf{f}[\mathbf{f}_\Delta(\mathbf{x}), \mathbf{f}_\Delta(\tilde{\mathbf{x}})] \\ &= \delta_t \delta_\tau \mathrm{cov}_\mathbf{f}[\mathbf{f}(\mathbf{x}), \mathbf{f}(\tilde{\mathbf{x}})] \\ &= \delta_t \delta_\tau \mathbf{K}(\mathbf{x}, \tilde{\mathbf{x}}) \end{aligned}$$

Therefore, since the core structure of the sampled mean and covariance remain the same as in continuous-time, we can conclude that *applying the Euler method does not provide any improvement over the standard method in terms of exploiting continuous-time structure.* Now we will work through the Midpoint method, given by

$$\mathbf{f}_\Delta(\mathbf{x}) = \mathbf{x} + \delta_t \mathbf{f}\big(\mathbf{x} + \tfrac{1}{2}\delta_t \mathbf{f}(\mathbf{x})\big)$$

First summarise the term in brackets as $\mathbf{p} = \mathbf{x} + \frac{1}{2}\delta_t \mathbf{f}(\mathbf{x})$. It is clear that this will be Gaussian distributed with mean $\mathbf{x} + \frac{1}{2}\delta_t \mathbf{m}(\mathbf{x})$ and covariance $\frac{1}{4}\delta_t^2 \mathbf{K}(\mathbf{x}, \mathbf{x})$ for a given value of $\mathbf{x}$. Therefore the sampled mean and covariance functions will be given by

$$\begin{aligned} \mathbf{m}_\Delta(\mathbf{x}) &= \mathbf{x} + \delta_t \mathbb{E}_{\mathbf{f},\mathbf{p}}\big[\mathbf{f}(\mathbf{p})\big] \\ &= \mathbf{x} + \delta_t \mathbb{E}_\mathbf{p}[\mathbf{m}(\mathbf{p})] \end{aligned} \qquad \begin{aligned} \mathbf{K}_\Delta(\mathbf{x}, \tilde{\mathbf{x}}) &= \delta_t \delta_\tau \mathrm{cov}_{\mathbf{f},\mathbf{p},\tilde{\mathbf{p}}}\big[\mathbf{f}(\mathbf{p}), \mathbf{f}(\tilde{\mathbf{p}})\big] \\ &= \delta_t \delta_\tau \Big(\mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[\mathbf{K}(\mathbf{p}, \tilde{\mathbf{p}}) + \mathbf{m}(\mathbf{p})\mathbf{m}(\tilde{\mathbf{p}})^\top\big] \\ &\qquad - \mathbb{E}_\mathbf{p}[\mathbf{m}(\mathbf{p})]\mathbb{E}_{\tilde{\mathbf{p}}}[\mathbf{m}(\tilde{\mathbf{p}})]^\top \Big) \end{aligned}$$

where integrals over the continuous mean and covariance function with respect to the random variables $\mathbf{p}$ and $\tilde{\mathbf{p}}$ are required. This is a very similar requirement to the moment matching condition, although now we require cross-covariance terms. The effect of this structural change to the prior will be discussed in Sec. 5.4.5. We shall now work through how to do this for a general Runge-Kutta method.

5.4.2 General Framework

Let us employ the assumption that $\mathbf{f}(\mathbf{p}_i)$ and $\mathbf{f}(\tilde{\mathbf{p}}_j)$ for $i, j \in \mathbb{Z}_{[1,S]}$ are jointly Gaussian distributed given $\mathbf{p}_i, \tilde{\mathbf{p}}_j \sim \mathcal{N}$. Under this assumption we can apply the process outlined in the illustrative examples for arbitrary orders of the Runge-Kutta method. Note that an implication of this assumption is that $\mathbf{p}_i$ and $\mathbf{p}_j$ will be jointly Gaussian since they are related linearly through Eq. (5.6). In this case we can write

$$\mathbf{m}_\Delta(\mathbf{x}) = \mathbf{x} + \delta_t \sum_{i=1}^{S} b_i \, \mathbb{E}\big[\mathbf{f}(\mathbf{p}_i)\big] \tag{5.7}$$

$$\mathbf{K}_\Delta(\mathbf{x}, \tilde{\mathbf{x}}) = \delta_t \delta_\tau \sum_{i=1}^{S} \sum_{j=1}^{S} b_i b_j \, \mathrm{cov}\big[\mathbf{f}(\mathbf{p}_i), \mathbf{f}(\tilde{\mathbf{p}}_j)\big] \tag{5.8}$$

where $\mathbf{p}_i$ and $\tilde{\mathbf{p}}_j$ denote $\mathbf{p}_i(\mathbf{x}_t)$ and $\mathbf{p}_j(\tilde{\mathbf{x}}_\tau)$ respectively and the integrals are taken with respect to the random vectors $\mathbf{p}_i, \tilde{\mathbf{p}}_j$ and function $\mathbf{f}$. Similarly for Eq. (5.6) we can use the following iterative rule

$$\mathbb{E}[\mathbf{p}_i] = \mathbf{x} + \delta_t \sum_{p=1}^{i-1} a_{ip}\, \mathbb{E}\big[\mathbf{f}(\mathbf{p}_p)\big] \tag{5.9}$$

$$\operatorname{cov}[\mathbf{p}_i, \tilde{\mathbf{p}}_j] = \delta_t \delta_\tau \sum_{p=1}^{i-1} \sum_{q=1}^{j-1} a_{ip} a_{jq} \operatorname{cov}\big[\mathbf{f}(\mathbf{p}_p), \mathbf{f}(\tilde{\mathbf{p}}_q)\big] \tag{5.10}$$

Now all that remains is to evaluate the expectation $\mathbb{E}\big[\mathbf{f}(\mathbf{p}_i)\big]$ and covariance $\operatorname{cov}\big[\mathbf{f}(\mathbf{p}_i), \mathbf{f}(\tilde{\mathbf{p}}_j)\big]$ in these equations. These can be calculated using a moment matching type procedure similar to that of Sec. 3.5

$$\mathbb{E}_{\mathbf{p},\mathbf{f}}\big[\mathbf{f}(\mathbf{p})\big] = \mathbb{E}_{\mathbf{p}}\big[\mathbf{m}(\mathbf{p})\big] \tag{5.11}$$

$$\operatorname{cov}_{\mathbf{p},\tilde{\mathbf{p}},\mathbf{f}}\big[\mathbf{f}(\mathbf{p}), \mathbf{f}(\tilde{\mathbf{p}})\big] = \mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[\mathbf{K}(\mathbf{p}, \tilde{\mathbf{p}}) + \mathbf{m}(\mathbf{p})\mathbf{m}(\tilde{\mathbf{p}})^\top\big] - \mathbb{E}_{\mathbf{p}}\big[\mathbf{m}(\mathbf{p})\big]\mathbb{E}_{\tilde{\mathbf{p}}}\big[\mathbf{m}(\tilde{\mathbf{p}})\big]^\top \tag{5.12}$$

where the integrals are again taken with respect to the random vectors $\mathbf{p}_i, \tilde{\mathbf{p}}_j$ and function $\mathbf{f}$. The difference between this and the procedure in Sec. 3.5 is that a cross covariance is required in this calculation rather than the covariance of a single random vector $\operatorname{cov}\big[\mathbf{f}(\mathbf{p}_i)\big]$. However, any mean and covariance function which satisfies the moment matching criteria will also satisfy this criterion and can be used in this context.

The full structural definition of a new set of priors over sampled autonomous continuous-time dynamical systems is given by Eqs. (5.7)–(5.12). The more general non-autonomous case is discussed in the following section and some examples of kernels for which Eqs. (5.11)–(5.12) can be evaluated are given in Sec. 5.4.4.

5.4.3 Non-Autonomous Case

Obviously from a control perspective we are most interested in dealing with systems with external inputs $\mathbf{u}$. So how do we deal with them in this framework? First imagine that our $\mathbf{p}$ values are expanded to include a state and an action part $\mathbf{p} = [\mathbf{p}_\mathbf{x}; \mathbf{p}_\mathbf{u}]$. If the action is applied independently of the state over the timestep, for example if it is zero-order hold, first order hold or some other time-dependent function then Eqs. (5.9)–(5.10) can be extended trivially as

$$\mathbb{E}[\mathbf{p}_i] = \begin{bmatrix} \mathbf{x} \\ \mathbf{u} + \mathbf{e}(c_i, \mathbf{u}) \end{bmatrix} + \delta_t \sum_{p=1}^{i-1} a_{ip} \begin{bmatrix} \mathbb{E}\big[\mathbf{f}(\mathbf{p}_p)\big] \\ \mathbf{0} \end{bmatrix} \tag{5.13}$$

$$\operatorname{cov}[\mathbf{p}_i, \tilde{\mathbf{p}}_j] = \delta_t \delta_\tau \sum_{p=1}^{i-1} \sum_{q=1}^{j-1} a_{ip} a_{jq} \begin{bmatrix} \operatorname{cov}\big[\mathbf{f}(\mathbf{p}_p), \mathbf{f}(\tilde{\mathbf{p}}_q)\big] & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{5.14}$$

where $c_i \in \mathbb{R}_{[0,1]}$ and $\mathbf{e}(c_i, \mathbf{u}) = \mathbf{0}$ for a zero-order hold scheme and $\mathbf{e}(c_i, \mathbf{u}) = c_i(\mathbf{u}_+ - \mathbf{u})$, where $\mathbf{u}_+$ is the action at the following timestep, for a first-order hold scheme. Note that this is not the same as saying that there is not a feedback policy acting on the system, but rather that the policy does not make any state-dependent changes within the sampling window.

Things get a little more complicated if we consider that the action is governed by some policy $\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$ over the time interval. In this case we have the following relationships

$$\mathbb{E}[\mathbf{p}_i] = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix} + \delta_t \sum_{p=1}^{i-1} a_{ip} \begin{bmatrix} \mathbb{E}\big[\mathbf{f}(\mathbf{p}_p)\big] \\ \mathbb{E}\big[\boldsymbol{\pi}(\mathbf{p}_p)\big] \end{bmatrix}$$

$$\mathrm{cov}[\mathbf{p}_i, \tilde{\mathbf{p}}_j] = \delta_t \delta_\tau \sum_{p=1}^{i-1} \sum_{q=1}^{j-1} a_{ip} a_{jq} \begin{bmatrix} \mathrm{cov}\big[\mathbf{f}(\mathbf{p}_p), \mathbf{f}(\tilde{\mathbf{p}}_q)\big] & \mathrm{cov}\big[\mathbf{f}(\mathbf{p}_p), \boldsymbol{\pi}(\tilde{\mathbf{p}}_q)\big] \\ \mathrm{cov}\big[\boldsymbol{\pi}(\mathbf{p}_p), \mathbf{f}(\tilde{\mathbf{p}}_q)\big] & \mathrm{cov}\big[\boldsymbol{\pi}(\mathbf{p}_p), \boldsymbol{\pi}(\tilde{\mathbf{p}}_q)\big] \end{bmatrix}$$

where the mean $\mathbb{E}\big[\boldsymbol{\pi}(\mathbf{p}_p)\big]$ and covariance $\mathrm{cov}\big[\boldsymbol{\pi}(\mathbf{p}_p), \boldsymbol{\pi}(\tilde{\mathbf{p}}_q)\big]$ are dependent only on the form of the function $\boldsymbol{\pi}$. However, there is no general way of calculating the cross covariance term $\mathrm{cov}\big[\mathbf{f}(\mathbf{p}_p), \boldsymbol{\pi}(\tilde{\mathbf{p}}_q)\big]$. Fortunately in this case we can appeal to the results of Sec. 4.2 to obtain the approximate relationship

$$\mathrm{cov}\big[\mathbf{f}(\mathbf{p}_p), \boldsymbol{\pi}(\tilde{\mathbf{p}}_q)\big] = \mathrm{cov}\big[\mathbf{f}(\mathbf{p}_p), [\mathbf{p}_p; \tilde{\mathbf{p}}_q]\big]\mathrm{cov}\big[[\mathbf{p}_p; \tilde{\mathbf{p}}_q]\big]^{-1}\mathrm{cov}\big[[\mathbf{p}_p; \tilde{\mathbf{p}}_q], \boldsymbol{\pi}(\tilde{\mathbf{p}}_q)\big]$$

to provide a complete framework.

5.4.4 Priors Over Useful Function Classes

Class of Functions

First we must consider the kind of structure we would like to draw out when we carry out this inference. We believe that a useful space of continuous-time functions to search over is the sum of a full nonlinear part, an additive part and a linear part

$$\begin{aligned} \mathbf{f}(\mathbf{z}) &= \mathbf{f}_{\text{full}}(\mathbf{z}) + \mathbf{f}_{\text{add}}(\mathbf{z}) + \mathbf{f}_{\text{lin}}(\mathbf{z}) \\ &= \mathbf{f}_{\text{full}}(\mathbf{z}) + \sum_{i=1}^{D} \mathbf{f}_{\text{add},i}\big(z[i]\big) + \mathbf{A}\mathbf{z} \end{aligned} \tag{5.15}$$

If we make the unrestrictive assumption that $\mathbf{f}_{\text{full}}(\mathbf{z})$, $\mathbf{f}_{\text{add}}(\mathbf{z})$ and $\mathbf{f}_{\text{lin}}(\mathbf{z})$ are each drawn from independent Gaussian process priors then the GP parameterising the distribution over the whole function $\mathbf{f}$ consists simply of the sum of the individual mean and covariance functions

$$\mathbf{f} \sim \mathcal{GP}\Big(\mathbf{m}_{\text{full}} + \mathbf{m}_{\text{add}} + \mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{full}} + \mathbf{K}_{\text{add}} + \mathbf{K}_{\text{lin}}\Big)$$

We shall only consider a useful subset of this prior given by $\mathcal{GP}\big(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{full}} + \mathbf{K}_{\text{add}}\big)$. MATLAB code to evaluate this prior is provided in Appendix C.2 and the function `prior.m`. We will now evaluate Eqs. (5.11)–(5.12) for each term in this prior.

Full Nonlinear Part

A general prior over the space of smooth nonlinear functions is the zero mean, squared exponential covariance prior. In this instance Eqs. (5.11)–(5.12) become

$$\begin{aligned} \mathbb{E}_{\mathbf{p},\mathbf{f}}\big[\mathbf{f}(\mathbf{p})\big] &= \mathbf{0} \\ \operatorname{cov}_{\mathbf{p},\tilde{\mathbf{p}},\mathbf{f}}\big[\mathbf{f}(\mathbf{p}),\mathbf{f}(\tilde{\mathbf{p}})\big] &= \mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[\mathbf{K}(\mathbf{p},\tilde{\mathbf{p}})\big] \end{aligned}$$

Again, we shall restrict our attention to a diagonal kernel. The $(a,a)^{\text{th}}$ element of the expectation of the prior covariance $K[a,a]$ with respect to a joint distribution over the two random variables $\hat{\mathbf{p}} := [\mathbf{p};\tilde{\mathbf{p}}] \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$ is then given by

$$\begin{aligned} \operatorname{cov}_{\mathbf{p},\tilde{\mathbf{p}},\mathbf{f}}\big[f[a](\mathbf{p}),f[a](\tilde{\mathbf{p}})\big] &= \int K[a,a](\hat{\mathbf{p}})\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}} \\ &= \int \alpha_a^2 \exp\Big(-\tfrac{1}{2}(\mathbf{p}-\tilde{\mathbf{p}})^\top\boldsymbol{\Lambda}_a^{-1}(\mathbf{p}-\tilde{\mathbf{p}})\Big)\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}} \\ &= \int \alpha_a^2 \exp\left(-\tfrac{1}{2}\hat{\mathbf{p}}^\top \underbrace{\begin{bmatrix}\mathbf{I}\\-\mathbf{I}\end{bmatrix}\boldsymbol{\Lambda}_a^{-1}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}}_{\hat{\boldsymbol{\Lambda}}_a^{-1}}\hat{\mathbf{p}}\right)\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}} \\ &= \alpha_a^2(2\pi)^{D/2}|\hat{\boldsymbol{\Lambda}}_a|^{-1/2}\int \mathcal{N}(\hat{\mathbf{p}}|\mathbf{0},\hat{\boldsymbol{\Lambda}}_a)\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu},\boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}} \\ &= \alpha_a^2\big|\boldsymbol{\Sigma}\hat{\boldsymbol{\Lambda}}_a^{-1}+\mathbf{I}\big|^{-1/2}\exp\Big(-\tfrac{1}{2}\boldsymbol{\mu}^\top(\hat{\boldsymbol{\Lambda}}_a+\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}\Big) \end{aligned} \tag{5.16}$$

Now since $\hat{\boldsymbol{\Lambda}}_a^{-1}$ is not full rank it should not appear in its un-inverted form. However, we can simply rewrite the term $(\hat{\boldsymbol{\Lambda}}_a+\boldsymbol{\Sigma})^{-1}$ as $\hat{\boldsymbol{\Lambda}}_a^{-1/2}\big(\hat{\boldsymbol{\Lambda}}_a^{-1/2}\boldsymbol{\Sigma}\hat{\boldsymbol{\Lambda}}_a^{-1/2}+\mathbf{I}\big)^{-1}\hat{\boldsymbol{\Lambda}}_a^{-1/2}$ where $\hat{\boldsymbol{\Lambda}}_a^{-1/2} = \frac{1}{\sqrt{2}}[\mathbf{I},-\mathbf{I}]^\top\boldsymbol{\Lambda}_a^{-1/2}[\mathbf{I},-\mathbf{I}]$ for a numerically stable and symmetric calculation.

Additive Nonlinear Part

As shown in Sec. 3.5 the propagation of uncertainty equations for the general additive SE kernel is closely related to the equations for the standard kernel. The same is clearly true in this instance.

Linear Part

We have already shown that a distribution of linear systems can be parameterised as

$$\begin{aligned} \mathbf{m}(\mathbf{p}) &= (\mathbf{I}\otimes\mathbf{p}^\top)\boldsymbol{\eta} \\ \mathbf{K}(\mathbf{p},\tilde{\mathbf{p}}) &= (\mathbf{I}\otimes\mathbf{p}^\top)\boldsymbol{\Omega}(\mathbf{I}\otimes\tilde{\mathbf{p}}) \end{aligned}$$

It is clear by looking at the standard moment matching equations provided in Eqs. (3.47)–(3.48) that the expressions for Eqs. (5.11)–(5.12) are very closely related. In fact they are given by

$$\begin{aligned}\mathbb{E}_{\mathbf{p},\mathbf{f}}\big[\mathbf{f}(\mathbf{p})\big] &= (\mathbf{I}_E \otimes \mathbf{1}_D^\top)\Big((\mathbf{1}_E \otimes \boldsymbol{\mu}) \circ \boldsymbol{\eta}\Big) \\ \operatorname{cov}_{\mathbf{p},\tilde{\mathbf{p}},\mathbf{f}}\big[\mathbf{f}(\mathbf{p}), \mathbf{f}(\tilde{\mathbf{p}})\big] &= (\mathbf{I}_E \otimes \mathbf{1}_D^\top)\Big(\big(\mathbf{1}_{E\times E} \otimes (\bar{\boldsymbol{\Sigma}} + \boldsymbol{\mu}\tilde{\boldsymbol{\mu}}^\top)\big)\big(\boldsymbol{\Omega} + \boldsymbol{\eta}\boldsymbol{\eta}^\top\big)\Big)(\mathbf{I}_E \otimes \mathbf{1}_D) \\ &\qquad - \mathbb{E}_{\mathbf{p},\mathbf{f}}\big[\mathbf{f}(\mathbf{p})\big]\mathbb{E}_{\tilde{\mathbf{p}},\mathbf{f}}\big[\mathbf{f}(\tilde{\mathbf{p}})\big]^\top\end{aligned}$$

where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{p}], \tilde{\boldsymbol{\mu}} = \mathbb{E}[\tilde{\mathbf{p}}]$ and $\bar{\boldsymbol{\Sigma}} = \operatorname{cov}[\mathbf{p}, \tilde{\mathbf{p}}]$. Of course we only require the expression for the mean since we restrict our attention to the prior $\mathcal{GP}\big(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{add}} + \mathbf{K}_{\text{full}}\big)$.

5.4.5 Structural Implications on the Prior

In order to gain a little more insight into the way the structure of the continuous-time prior is actually altered through this process we shall return to the illustrative example of the midpoint method. We shall consider a uniformly sampled autonomous system with a zero mean squared exponential prior over the continuous-time dynamics. In this case the elements of the sampled mean and covariance functions are

$$m_\Delta[a](\mathbf{x}) = x[a] \tag{5.17}$$

$$K_\Delta[a,a](\mathbf{x},\tilde{\mathbf{x}}) = \alpha_{\Delta a}^2 \exp\Big(-\tfrac{1}{2}(\mathbf{x}-\tilde{\mathbf{x}})^\top \boldsymbol{\Lambda}_{\Delta a}^{-1}(\mathbf{x}-\tilde{\mathbf{x}})\Big) \tag{5.18}$$

where $a \in \mathbb{Z}_{[1,E]}$. The new output variance $\alpha_{\Delta a}^2$ and length scale terms $\boldsymbol{\Lambda}_{\Delta a}^{-1}$ are functions of the states $\mathbf{x}$ and $\tilde{\mathbf{x}}$. Specifically

$$\begin{aligned}\alpha_{\Delta a}^2 &= \delta_t^2 \alpha_a^2 \left| \tfrac{1}{4}\delta_t^2 \begin{bmatrix}\mathbf{K}(\mathbf{x},\mathbf{x}) & \mathbf{K}(\mathbf{x},\tilde{\mathbf{x}}) \\ \mathbf{K}(\tilde{\mathbf{x}},\mathbf{x}) & \mathbf{K}(\tilde{\mathbf{x}},\tilde{\mathbf{x}})\end{bmatrix} \hat{\boldsymbol{\Lambda}}_a^{-1} + \mathbf{I}\right|^{-1/2} \\ \boldsymbol{\Lambda}_{\Delta a}^{-1} &= \tfrac{1}{2}\boldsymbol{\Lambda}_a^{-1/2}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\left(\tfrac{1}{4}\delta_t^2 \hat{\boldsymbol{\Lambda}}_a^{-1/2}\begin{bmatrix}\mathbf{K}(\mathbf{x},\mathbf{x}) & \mathbf{K}(\mathbf{x},\tilde{\mathbf{x}}) \\ \mathbf{K}(\tilde{\mathbf{x}},\mathbf{x}) & \mathbf{K}(\tilde{\mathbf{x}},\tilde{\mathbf{x}})\end{bmatrix}\hat{\boldsymbol{\Lambda}}_a^{-1/2} + \mathbf{I}\right)^{-1}\begin{bmatrix}\mathbf{I} \\ -\mathbf{I}\end{bmatrix}\boldsymbol{\Lambda}_a^{-1/2}\end{aligned}$$

We can see that the sampled prior in Eqs. (5.17)–(5.18) is made up of a linear mean function and a covariance function in the form of a squared exponential but with state-dependant output variance and length-scales. Further, we can see that a cross coupling between the continuous-time hyperparameters has occurred since $\alpha_{\Delta a}$ and $\boldsymbol{\Lambda}_{\Delta a}$ are functions of all the hyperparameters in $\mathbf{K}$, not just α_a and $\boldsymbol{\Lambda}_a$.

Finally, we can observe that this prior has the intuitive property that as we let $\delta_t \to 0$ then $\alpha_{\Delta a}^2 \to \delta_t^2\alpha_a^2$ and $\boldsymbol{\Lambda}_{\Delta a}^{-1} \to \boldsymbol{\Lambda}_a^{-1}$. In other words the cross coupling and state dependence of the discrete hyperparameters is turned off as we consider smaller and smaller sampling times.

5.4.6 Computational Complexity

It is important at this stage to include a discussion of the computational complexity of training with this new class of priors. In either case we must first evaluate the prior over the training data parameterised by $\mathbf{m}_\Delta(\mathbf{Z})$ and $\mathbf{K}_\Delta(\mathbf{Z},\mathbf{Z})$. Clearly $\mathbf{K}_\Delta(\mathbf{Z},\mathbf{Z})$ is the limiting factor. For the first iteration of any of these methods we are simply calculating the continuous-time prior $\mathbf{m}(\mathbf{Z})$ and $\mathbf{K}(\mathbf{Z},\mathbf{Z})$. This has a complexity of $\mathcal{O}(En^2)$ for a diagonal covariance function and excluding constant terms. We then need to make $S + \frac{1}{2}S(S-1) - 1$ calculations of terms of the form $\mathbb{E}_{\mathbf{P},\tilde{\mathbf{P}}}\big[\mathbf{K}(\mathbf{P},\tilde{\mathbf{P}})\big]$ where $\mathbf{P},\tilde{\mathbf{P}} \in \mathbb{R}^{n\times D}$. If the continuous-time covariance function is a squared exponential then calculation of this term requires $\mathcal{O}(D^3En^2)$ since a $2D \times 2D$ matrix needs to be inverted for every pair of data points. If it is a first order additive covariance function then we have complexity of $\mathcal{O}(DEn^2)$ since we need to carry out D inversions of a 2×2 matrix for every pair of data points.

In summary, if we include a squared exponential in the covariance function of the continuous-time prior then the evaluation of our discrete-time prior has complexity of the order $\mathcal{O}(S^2D^3En^2)$ for $S \geq 2$ and excluding constant terms. If we do not have a squared exponential but a first order additive squared exponential then this reduces to $\mathcal{O}(S^2DEn^2)$ for $S \geq 2$.

5.4.7 Example: Pendulum

Inferring Linear Plus Nonlinear Structure

Recall the pendulum system from Sec. 4.3.3 which has the equation of motion

$$\tfrac{1}{3}ml^2\ddot{\theta} = u - b\dot{\theta} - \tfrac{1}{2}mlg\sin\theta \tag{5.19}$$

with constants $m = 1\,\mathrm{kg}$, $l = 1\,\mathrm{m}$, $b = 0.1\,\mathrm{N\,s^{-1}}$ and $g = 9.81\,\mathrm{m\,s^{-2}}$ from Appendix B.1. Initially, we shall consider the unconstrained version of this problem. In continuous-time these dynamics can be decomposed into a linear and a nonlinear part as follows

$$\begin{bmatrix}\dot{\theta}\\ \ddot{\theta}\end{bmatrix} = \underbrace{\begin{bmatrix}\dot{\theta}\\ -0.3\dot{\theta}+3u\end{bmatrix}}_{\text{Linear}} + \underbrace{\begin{bmatrix}0\\ -1.5g\sin\theta\end{bmatrix}}_{\text{Nonlinear}} \tag{5.20}$$

The training input data set we considered consisted of n independent samples from the distribution $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, 9\mathbf{I})$. Although this inference problem looks very simple it is in fact an interesting problem. This is because in continuous-time θ is given by a simple linear relationship, but in discrete-time this relationship becomes nonlinear as the dynamics get coupled together. It would be useful if our methods could pick out this continuous-time information given the discrete-time data.

The first set of simulations was run with a base GP which parameterised a distribution over

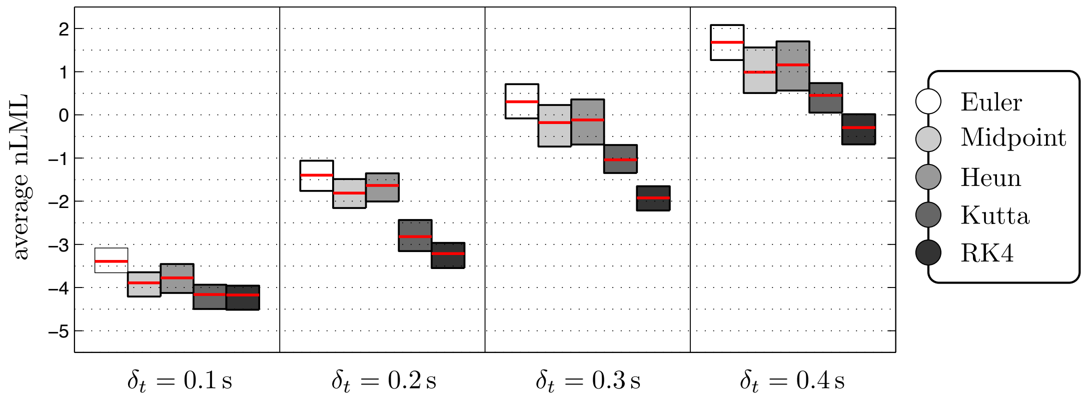


Figure 5.1: Average negative log-marginal likelihoods for discrete GPs with the continuous-time prior $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{full}})$ trained on data from the *unconstrained* pendulum. The graph is obtained from 50 Monte-Carlo runs for different types of sampled prior and values of δ_t. Each box depicts the 10th to 90th percentile of the data with the red line showing the 50th percentile.

		Parameters for $\mathbf{m}_{\text{lin}}$			Parameters for $\mathbf{K}_{\text{full}}$			
		η_θ	$\eta_{\dot\theta}$	η_u	λ_θ	$\lambda_{\dot\theta}$	λ_u	α
Euler	θ	0.07	**0.95**	**0.58**	1.38	8.26	26.25	**2.15**
	$\dot\theta$	0.41	**-0.42**	**2.67**	1.14	**5.55**	**28.03**	9.53
Midpoint	θ	0.05	0.86	0.19	1.38	61.41	44.80	0.81
	$\dot\theta$	0.51	0.01	2.68	1.43	20.07	74.10	10.28
Heun	θ	0.28	0.77	0.19	3.16	54.80	79.22	1.08
	$\dot\theta$	-0.02	0.30	2.65	1.65	15.35	102.02	13.85
Kutta	θ	0.03	0.98	0.19	2.55	6.43	4.36	0.04
	$\dot\theta$	0.36	-0.46	2.38	1.45	176.05	39.47	11.23
RK4	θ	0.12	**1.00**	**-0.01**	0.98	3.51	3.40	**0.02**
	$\dot\theta$	-0.21	**-0.54**	**3.06**	1.39	**211.30**	**315.35**	10.13

Table 5.2: The average of the learned hyperparameters for methods trained using a $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{full}})$ continuous-time prior and a setting of $\delta_t = 0.4\,\mathrm{s}$ on the *unconstrained* pendulum data. The numbers in bold depict important changes from the Euler to the RK4 method.

		Parameters for $\mathbf{m}_{\text{lin}}$			Parameters for $\mathbf{K}_{\text{full}}$			
		η_θ	$\eta_{\dot\theta}$	η_u	λ_θ	$\lambda_{\dot\theta}$	λ_u	α
Euler	θ	0.01	0.99	**0.15**	1.53	90.64	140.62	**0.62**
	$\dot\theta$	-0.02	-0.26	2.97	1.86	**21.49**	233.97	14.07
RK4	θ	0.01	1.00	**0.00**	2.21	1.29	4.37	$\mathbf{10^{-6}}$
	$\dot\theta$	-0.28	-0.31	3.00	1.88	**421.26**	429.24	13.52

Table 5.3: The average of the learned hyperparameters for methods trained using a $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{full}})$ continuous-time prior and a setting of $\delta_t = 0.1\,\text{s}$ on the *unconstrained* pendulum data. The numbers in bold depict important changes from the Euler to the RK4 method.

all linear functions plus a smooth nonlinear part $\mathbf{f} \sim \mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{full}})$. We sampled 50 different training data sets for various settings of δ_t. For each data set we trained the hyperparameters of the different Runge-Kutta based discrete-time priors outlined in the previous section using gradient descent on the average negative Log-Marginal Likelihood (nLML) per data point. The reasoning for the use of this model selection criterion is outlined in Sec. 3.4.4.

This form of prior encodes our knowledge about the form of the continuous-time dynamics and should therefore lead to a significant reduction in the learned nLML. This trend is clearly observed in Fig. 5.1. As we would expect the saving becomes more significant for larger values of δ_t and the difference in performance between higher order approximations becomes more pronounced. We note that although the Heun and Midpoint methods are both of order two the Midpoint method consistently outperforms the Heun method across all our simulations. The underlying reason for this eludes us.

The average hyperparameters associated with the $\delta_t = 0.4\,\text{s}$ runs are given in Table 5.2. The main parameters responsible for the savings in nLML have been highlighted in bold. Let us first examine the parameters associated with the evolution of θ, shown by the grey rows. We know that in continuous-time the evolution of θ is given by a purely linear relationship, therefore we would hope that the α parameter in $\mathbf{K}_{\text{full}}$ would tend to zero. This is the general trend that is in fact observed. We can further note that the linear parameters tend to be closer to the actual continuous-time parameters as the order is increased.

Now we turn our attention to the parameters associated with the evolution of $\dot\theta$. The savings here can be observed by looking at the length-scales λ associated with the $\dot\theta$ and u dimensions. Ideally we would like to see these become very large, indicating that the linear component for these dimensions is sufficient to explain the data. Again, this is the trend we can observe. Looking at the linear parameters, we can see that these get closer to the actual values with the exception of the $\dot\theta$ weight. Finally, although the linear parameter for the θ dimension has not become zero, this is acceptable because it is using this linear part in conjunction with the nonlinearity provided by the SE kernel to explain the sinusoidal relationship.

If we look at the parameters for a setting of $\delta_t = 0.1\,s$, given in Table 5.3 we can see that the parameters learned by the RK4 method correspond very closely to the idealised relationship.

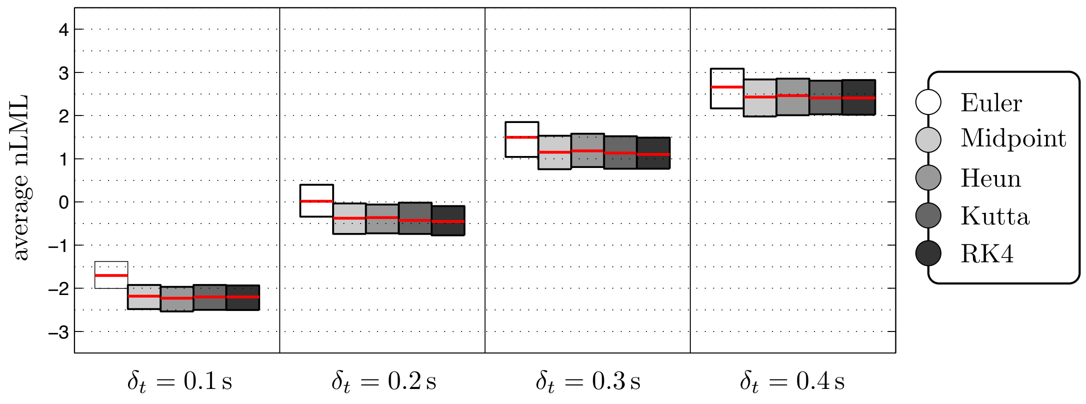


Figure 5.2: Average negative log-marginal likelihoods for discrete GPs with the continuous-time prior $\mathcal{GP}(\mathbf{0}, \mathbf{K}_{\text{full}})$ trained on data from the *unconstrained* pendulum. The graph is obtained from 50 Monte-Carlo runs for different types of sampled prior and values of δ_t. Each box depicts the 10th to 90th percentile of the data with the red line showing the 50th percentile.

		Parameters for $\mathbf{K}_{\text{full}}$			
		λ_θ	$\lambda_{\dot\theta}$	λ_u	α
Euler	θ	**3.97**	25.31	110.29	12.26
	$\dot\theta$	2.53	**37.65**	31.19	43.53
RK4	θ	**288.86**	90.31	307.09	59.78
	$\dot\theta$	2.43	**210.03**	34.53	49.85

Table 5.4: The average of the learned hyperparameters for methods trained using a $\mathcal{GP}(\mathbf{0}, \mathbf{K}_{\text{full}})$ continuous-time prior and a setting of $\delta_t = 0.1\,\mathrm{s}$ on the *unconstrained* pendulum data. The numbers in bold depict important changes from the Euler to the RK4 method.

The parameters which lead to the largest savings are again highlighted in bold. Note that the nonlinear term on θ has been almost completely switched off. For this smaller timestep we can also observe that the crude Euler method has parameters that correspond more closely with our idealised values, as we would expect.

The second set of simulations was for a training data set of size $n = 30$ but used only a zero-mean GP with a squared-exponential kernel $\mathbf{f} \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_{\text{full}})$ as a prior over continuous-time dynamics. The resulting spread in average nLML for a given method and setting of δ_t is depicted in Fig. 5.2. It is clear that without prior knowledge of the structure the saving is much less pronounced.

From Fig. 5.2 we can see that moving from a simple Euler scheme to a higher order method yields a reasonable decrease in nLML. The reason for this saving is clear if we again look at the learned hyperparameters. The average hyperparameters learned over the 50 Monte Carlo runs for $\delta_t = 0.1\,\mathrm{s}$ are given in Table 5.4 for the Euler and RK4 methods. Only the Euler and RK4 results are given because the values learned for RK4 are not significantly different from those learned for the Midpoint, Heun and Kutta methods. If we observe the numbers highlighted in bold i.e. the length-scale associated with θ and $\dot\theta$ acting on the θ and $\dot\theta$ state-dimensions respectively then we can see that the higher order methods have assigned these much higher values. In the case of the effect of θ acting on θ this is because the RK method has discovered an invariance that was hidden to the simple Euler scheme. Similarly, the higher order RK methods have discovered that the evolution of $\dot{\theta}$ is in fact only weakly correlated with $\dot{\theta}$ (a linear scaling of 0.3) and therefore can assign it a long length scale.

The improvement in predictive performance is much less in this case because the algorithm must explain the linear part of the function using only the nonlinear kernel. This means that the only possible saving is from finding invariance or weakly correlated relationships in continuous-time, for example the independence of the rate of change of θ with respect to θ itself and the weak correlation of the rate of change of $\dot{\theta}$ with $\dot{\theta}$ itself.

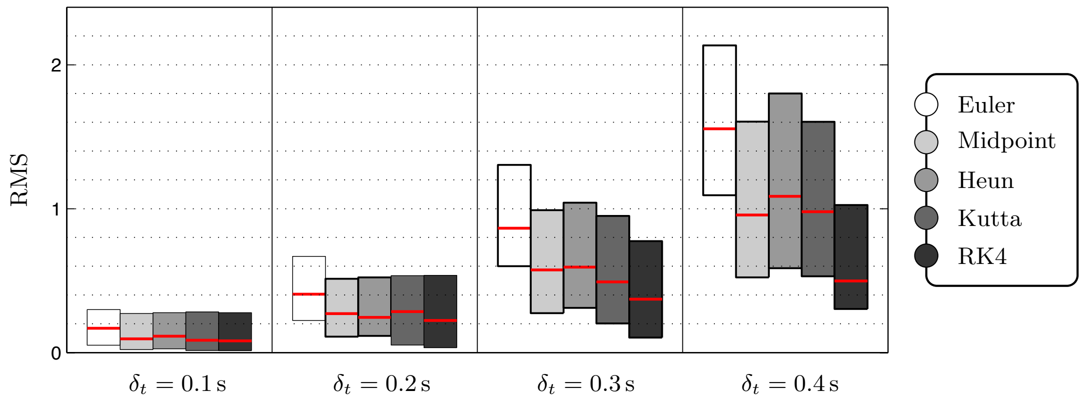
Up till now we have been using only the training data set to compare the quality of a given model. We shall now compare the predictive performance on a separate validation data set. To do this we provide a set of $n_v = 50$ data points, again sampled independently from $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, 9\mathbf{I})$ on which to validate the learned models. Comparison will take place based on two metrics: the *Root Mean Square (RMS) error* and the *negative Log-Predictive Density (nLPD)*. These are given by

$$\text{RMS} = \sqrt{\frac{1}{n_v}\sum_{i=1}^{n_v}(\boldsymbol{\mu}_i - \mathbf{f}_{\Delta i})^\top(\boldsymbol{\mu}_i - \mathbf{f}_{\Delta i})} \tag{5.21}$$

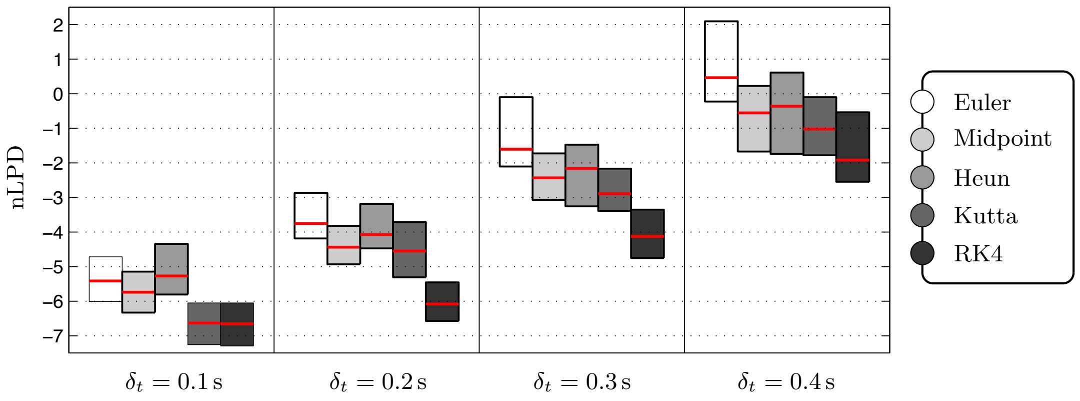
$$\text{nLPD} = \frac{1}{n_v}\left(\sum_{i=1}^{n_v}\tfrac{1}{2}(\boldsymbol{\mu}_i - \mathbf{f}_{\Delta i})^\top\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{\mu}_i - \mathbf{f}_{\Delta i}) + \tfrac{1}{2}\log\left|\boldsymbol{\Sigma}_i\right|\right) + \tfrac{E}{2}\log(2\pi) \tag{5.22}$$

respectively where $\mathbf{f}_{\Delta i} = \mathbf{f}_\Delta(\mathbf{z}_i)$ are the underlying function values, $\boldsymbol{\mu}_i = \mathbf{m}_{\Delta+}(\mathbf{z}_i)$ and $\boldsymbol{\Sigma}_i = \mathbf{K}_{\Delta+}(\mathbf{z}_i, \mathbf{z}_i)$ are the posterior mean and covariance predictions with $\boldsymbol{\mu}_{\Delta+}$ and $\mathbf{K}_{\Delta+}$ as the posterior sampled mean and covariance functions. We note that although the RMS error is the most commonly used performance measure, it does not take account of the uncertainty measure provided by a Gaussian process. Therefore, we consider the nLPD error to be a more appropriate measure in this probabilistic setting.

The validation results associated with the previous set of experiments for a continuous-time prior of $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{full}})$ are shown in Fig. 5.3. Across both metrics we can observe that there are substantial improvements in predictive performance as we progress to higher order Runge-Kutta methods and increase δ_t. As we move from the Euler to the RK4 method for $\delta_t = 0.4\,\text{s}$ we can see that we can get a saving of around a 60% in the RMS predictive performance. We note that for this particular problem the improvement in RMS when moving from the Midpoint method to the Kutta method is not significant. However, when compared against the nLPD metric, where the uncertainty prediction is also taken into account, we can observe a reasonable improvement, especially for $\delta_t = 0.1\,\text{s}$. In this particular set of experiments, the Heun method actually performs worse than the Euler method for $\delta_t = 0.1\,\text{s}$ with respect to the nLPD. This anomaly could be because the optimisation of the hyperparameters for the Heun method had not yet converged, or because it had fallen into some local minimum.



(a) Root mean square errors



(b) Negative log-predictive density errors

Figure 5.3: Comparison of the predictive performance of the RK-based methods given a continuous-time prior $\mathcal{GP}(\mathbf{m}_{\mathrm{lin}}, \mathbf{K}_{\mathrm{full}})$ using the RMS and nLPD metrics and based on data from the *unconstrained* pendulum. The graph is obtained from 50 Monte-Carlo runs for different types of sampled prior and values of δ_t. Each box depicts the 10th to 90th percentile of the data with the red line showing the 50th percentile.

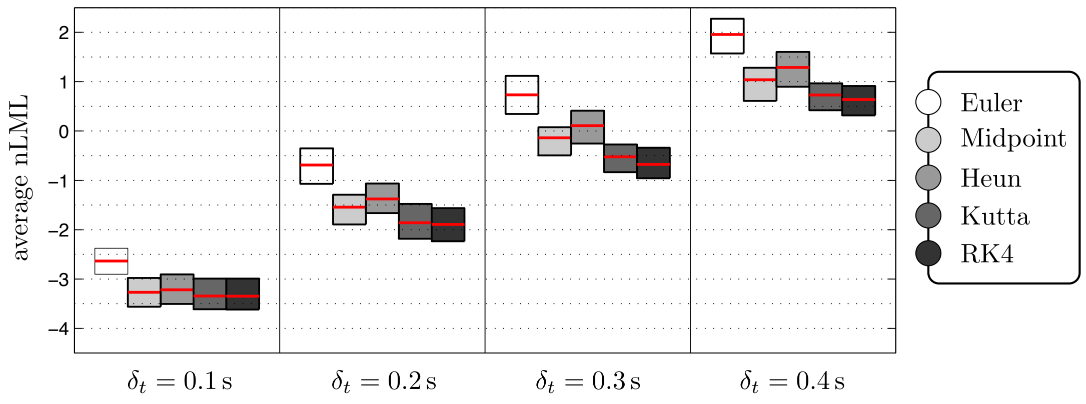


Figure 5.4: Average negative log-marginal likelihoods for discrete GPs with the continuous-time prior $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{add}} + \mathbf{K}_{\text{full}})$ trained on data from the *artificially constrained* pendulum. The graph is obtained from 50 Monte-Carlo runs for different types of sampled prior and values of δ_t. Each box depicts the 10th to 90th percentile of the data with the red line showing the 50th percentile.

Inferring Linear Plus Additive Structure

In order to demonstrate the usefulness of the additive kernel, consider a modification to the pendulum equations. Instead of assuming that the input we actually apply is in the range $u \in [-3, 3]\,\mathrm{N}$ we can put this constraint in as another nonlinearity and leave u unconstrained. This leads to the state space equation

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} \dot{\theta} \\ -0.3\dot{\theta} \end{bmatrix}}_{\text{Linear}} + \underbrace{\begin{bmatrix} 0 \\ -1.5g\sin\theta + 9\,\mathrm{sat}(\frac{1}{3}u) \end{bmatrix}}_{\text{Additive}} \tag{5.23}$$

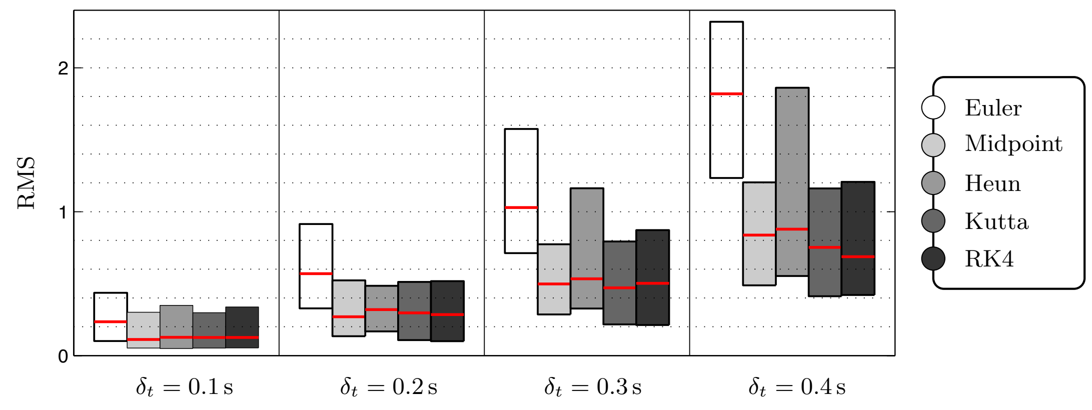
The power of the additive kernel will be in realising that the nonlinearity on θ and u is linearly separable. This is a "tighter" space of functions than a completely general nonlinearity over both dimensions.

As before, we obtained a training data set consisting of 30 data points sampled independently from $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, 9\mathbf{I})$. This is sufficient to excite the ranges of both nonlinear terms. We then trained discrete-time GPs based on a continuous-time prior consisting of linear, additive and full nonlinear terms $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{add}} + \mathbf{K}_{\text{full}})$. The results are shown in Fig. 5.4. We can clearly see that the higher order approximations have been able to exploit the underlying structure to produce a model with greater likelihood. Again, the improvement decreases with the sampling time δ_t.

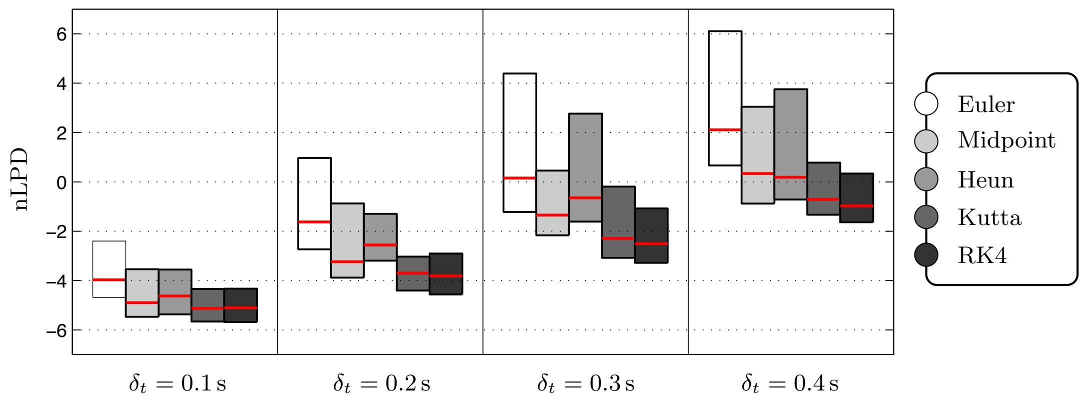
Let us now examine the learned hyperparameters associated with the additive and full nonlinear kernels. These are shown in Table 5.5 for the Euler and RK4 methods at $\delta_t = 0.4\,\mathrm{s}$. We can observe where the performance improvement comes from considering the output variance parameters α shown in bold. Ideally we would like to see that the only non-zero terms are the α_θ and α_u parameters on the $\dot{\theta}$ dimension. The Euler method has failed to pick up this structure and in fact assigns almost all of the weight to the fully nonlinear term. This is in contrast to the

		Parameters for $\mathbf{K}_{\text{add}}$						Parameters for $\mathbf{K}_{\text{full}}$			
		λ_θ	$\lambda_{\dot\theta}$	λ_u	α_θ	$\alpha_{\dot\theta}$	α_u	λ_θ	$\lambda_{\dot\theta}$	λ_u	α
Euler	θ	1.92	7.11	2.86	**0.65**	**0.11**	**0.45**	1.68	9.67	71.76	**1.53**
	$\dot\theta$	2.07	4.38	3.09	2.59	**0.61**	2.82	1.40	7.69	43.23	**10.46**
RK4	θ	8.93	5.40	7.81	**0.18**	**0.008**	**0.05**	1.03	0.81	1.78	**0.005**
	$\dot\theta$	1.56	2.06	2.23	10.96	**0.09**	2.94	1.16	1.04	3.54	**0.015**

Table 5.5: The average of the learned hyperparameters for methods trained using a $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{add}} + \mathbf{K}_{\text{full}})$ continuous-time prior and a setting of $\delta_t = 0.3\,\text{s}$ on the *artificially constrained* pendulum data. The numbers in bold depict important changes from the Euler to the RK4 method.



(a) Root mean square errors



(b) Negative log-predictive density errors

Figure 5.5: Comparison of the predictive performance of the RK-based methods given a continuous-time prior $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{add}} + \mathbf{K}_{\text{full}})$ using the RMS and nLPD metrics and based on data from the *artificially constrained* pendulum. The graph is obtained from 50 Monte-Carlo runs for different types of sampled prior and values of δ_t. Each box depicts the 10th to 90th percentile of the data with the red line showing the 50th percentile.

RK4 method which assigns nothing to the general nonlinearity and removes the irrelevant terms from the additive part.

Finally, we compare the predictive performance of these methods given a validation data set of 50 points with inputs sampled from $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, 9\mathbf{I})$ as before. The results are shown in Fig. 5.5. In terms of RMS error we can observe a significant improvement from the Euler to the Midpoint and Heun methods, but not much further improvement beyond that. However, if we compare under the negative log-predictive likelihood we see that the higher order methods do in fact provide a significant improvement based on their estimate of the predictive uncertainty.

5.5 Combining Time-Derivative and Sampled Data

5.5.1 Single Data Points

Now let us suppose that we actually have access to samples of states, actions and time derivatives of the states themselves. In other words our data set now has the form

$$\mathbf{Z} = \begin{bmatrix} \mathbf{x}_{t_1}^\top & \mathbf{u}_{t_1}^\top \\ \vdots & \vdots \\ \mathbf{x}_{t_m}^\top & \mathbf{u}_{t_m}^\top \\ \mathbf{x}_{t'_1}^\top & \mathbf{u}_{t'_1}^\top \\ \vdots & \vdots \\ \mathbf{x}_{t'_n}^\top & \mathbf{u}_{t'_n}^\top \end{bmatrix} \in \mathbb{R}^{(m+n)\times D} \qquad \text{and} \qquad \mathbf{Y} = \begin{bmatrix} \dot{\mathbf{x}}_{t_1}^\top \\ \vdots \\ \dot{\mathbf{x}}_{t_m}^\top \\ \mathbf{x}_{t'_1+\delta_1}^\top \\ \vdots \\ \mathbf{x}_{t'_n+\delta_n}^\top \end{bmatrix} \in \mathbb{R}^{(m+n)\times E} \tag{5.24}$$

It would be ideal if we could use the whole data set for training and prediction with our Gaussian process. But how do we combine these two kinds of data? Since we already have a means of evaluating continuous data (using the specified continuous GP) and sampled data (using the RK-based method from Sec. 5.4) the only thing we need to derive is the cross covariance term $\mathbf{K}_\times(\mathbf{z}_t, \tilde{\mathbf{z}}_\tau) := \operatorname{cov}_{\mathbf{f}}[\mathbf{f}_\Delta(\mathbf{z}_t), \mathbf{f}(\tilde{\mathbf{z}}_\tau)]$. It is clear from the equations we derived in Sec. 5.4 that this will be equal to

$$\mathbf{K}_\times(\mathbf{z}_t, \tilde{\mathbf{z}}_\tau) = \delta_t \sum_{i=1}^{S} b_i \operatorname{cov}_{\mathbf{p}_i,\mathbf{f}}\big[\mathbf{f}(\mathbf{p}_i), \mathbf{f}(\tilde{\mathbf{z}}_\tau)\big] \tag{5.25}$$

given a certain RK scheme. The covariance term in this equation can be calculated using a simplification of the equations derived in Sec. 5.4.4 where the distribution is only over the first argument $\mathbf{p}_i$. The distribution over $\mathbf{p}_i$ can be found using Eqs. (5.9)–(5.10) by setting $\tilde{\mathbf{p}}_j = \mathbf{p}_i$. In the non-autonomous case Eqs. (5.13)–(5.14) should be used.

5.5.2 State-Action Derivative Observations

Overview

It was observed by Solak *et al.* (2003) that the derivative of a Gaussian process remains a Gaussian process. This means that we can combine not only observed points from our unknown function but also gradient observations. In fact this notion can be generalised to any linear operation and is presented in its general form by Särkkä (2011). We shall restrict our attention to gradient observations since this is the most useful in the context of dynamical systems.

The incorporation of local linear models as derivative models in a Gaussian process modelling framework for dynamical systems is not a new concept. They have been used for system identification by Ažman & Kocijan (2011) and in a model predictive control framework by Ažman & Kocijan (2008). These papers consider only ARX models, where the system has a single output and input and the state is made up of lagged versions of these. The work of this thesis extends the use of derivative observations to general state-spaces and to the incorporation of known continuous-time linear models.

General Kernels

The covariances between a pair of data points and the associated derivatives are simply given by the corresponding derivatives of the covariance function itself

$$\begin{aligned}
\operatorname{cov}_{\mathbf{f}}\Big[\mathbf{f}(\mathbf{z}),\mathbf{f}(\tilde{\mathbf{z}})\Big] &= \mathbf{K}(\mathbf{z},\tilde{\mathbf{z}}) \\
\operatorname{cov}_{\mathbf{f}}\left[\frac{\partial \mathbf{f}(\mathbf{s})}{\partial s[i]}\bigg|_{\mathbf{s}=\mathbf{z}}, \mathbf{f}(\tilde{\mathbf{z}})\right] &= \mathbf{K}^{i}(\mathbf{z},\tilde{\mathbf{z}}) := \frac{\partial \mathbf{K}(\mathbf{s},\tilde{\mathbf{z}})}{\partial s[i]}\bigg|_{\mathbf{s}=\mathbf{z}} \\
\operatorname{cov}_{\mathbf{f}}\left[\frac{\partial \mathbf{f}(\mathbf{s})}{\partial s[i]}\bigg|_{\mathbf{s}=\mathbf{z}}, \frac{\partial \mathbf{f}(\tilde{\mathbf{s}})}{\partial \tilde{s}[j]}\bigg|_{\tilde{\mathbf{s}}=\tilde{\mathbf{z}}}\right] &= \mathbf{K}^{i,j}(\mathbf{z},\tilde{\mathbf{z}}) := \frac{\partial^2 \mathbf{K}(\mathbf{s},\tilde{\mathbf{s}})}{\partial s[i]\partial \tilde{s}[j]}\bigg|_{\mathbf{s}=\mathbf{z},\tilde{\mathbf{s}}=\tilde{\mathbf{z}}}
\end{aligned}$$

From these definitions we can also define the block matrices

$$\mathbf{K}'(\mathbf{z},\tilde{\mathbf{z}}) := \begin{bmatrix}\mathbf{K}^1(\mathbf{z},\tilde{\mathbf{z}})\\ \vdots \\ \mathbf{K}^D(\mathbf{z},\tilde{\mathbf{z}})\end{bmatrix} \quad \text{and} \quad \mathbf{K}''(\mathbf{z},\tilde{\mathbf{z}}) := \begin{bmatrix}\mathbf{K}^{1,1}(\mathbf{z},\tilde{\mathbf{z}}) & \dots & \mathbf{K}^{1,D}(\mathbf{z},\tilde{\mathbf{z}})\\ \vdots & \ddots & \vdots \\ \mathbf{K}^{D,1}(\mathbf{z},\tilde{\mathbf{z}}) & \dots & \mathbf{K}^{D,D}(\mathbf{z},\tilde{\mathbf{z}})\end{bmatrix}$$

The final piece of information we need in order to use it as a base kernel for our sampled continuous-time prior is to find the expectations $\mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[\mathbf{K}'_i(\mathbf{p},\tilde{\mathbf{p}})\big]$ and $\mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[\mathbf{K}''_{ij}(\mathbf{p},\tilde{\mathbf{p}})\big]$ for $\hat{\mathbf{p}} := [\mathbf{p};\tilde{\mathbf{p}}] \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$.

Squared Exponential Kernel

As usual, we shall restrict our attention to a diagonal kernel where the $(a, a)^{\text{th}}$ element of the prior covariance is $k_a := K[a, a]$. The expressions for $k'_a(\mathbf{z}, \tilde{\mathbf{z}})$ and $k''_a(\mathbf{z}, \tilde{\mathbf{z}})$ in this case are

$$\begin{aligned}
k'_a(\mathbf{z}, \tilde{\mathbf{z}}) &= -k_a(\mathbf{z}, \tilde{\mathbf{z}})\boldsymbol{\Lambda}_a^{-1}(\mathbf{z} - \tilde{\mathbf{z}}) \\
k''_a(\mathbf{z}, \tilde{\mathbf{z}}) &= k_a(\mathbf{z}, \tilde{\mathbf{z}})\Big(\boldsymbol{\Lambda}_a^{-1} - \boldsymbol{\Lambda}_a^{-1}(\mathbf{z} - \tilde{\mathbf{z}})(\mathbf{z} - \tilde{\mathbf{z}})^\top\boldsymbol{\Lambda}_a^{-1}\Big)
\end{aligned}$$

By defining $\hat{\mathbf{z}} := [\mathbf{z}; \tilde{\mathbf{z}}]$ we can rewrite these equations in the form

$$\begin{aligned}
k'_a(\hat{\mathbf{z}}) &= -k_a(\hat{\mathbf{z}})\boldsymbol{\Lambda}_a^{-1}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\hat{\mathbf{z}} \\
k''_a(\hat{\mathbf{z}}) &= k_a(\hat{\mathbf{z}})\left(\boldsymbol{\Lambda}_a^{-1} - \boldsymbol{\Lambda}_a^{-1}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\hat{\mathbf{z}}\hat{\mathbf{z}}^\top\begin{bmatrix}\mathbf{I} \\ -\mathbf{I}\end{bmatrix}\boldsymbol{\Lambda}_a^{-1}\right)
\end{aligned}$$

This will help us in evaluating the expectation of the first expression

$$\begin{aligned}
\mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[k'_a(\mathbf{p}, \tilde{\mathbf{p}})\big] &= -\boldsymbol{\Lambda}_a^{-1}\int\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\hat{\mathbf{p}}k_a(\hat{\mathbf{p}})\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}} \\
&= -\alpha_a^2(2\pi)^{D/2}|\hat{\boldsymbol{\Lambda}}_a|^{1/2}\boldsymbol{\Lambda}_a^{-1}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\int\hat{\mathbf{p}}\mathcal{N}(\hat{\mathbf{p}}|\mathbf{0}, \hat{\boldsymbol{\Lambda}}_a)\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}} \\
&= -\mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[k_a(\mathbf{p}, \tilde{\mathbf{p}})\big]\boldsymbol{\Lambda}_a^{-1}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\hat{\boldsymbol{\mu}} \qquad (5.26)
\end{aligned}$$

where $\hat{\boldsymbol{\mu}} := \boldsymbol{\Sigma}(\hat{\boldsymbol{\Lambda}}_a + \boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}$. The first step is achieved by noting the similarity with Eq. (5.16) and the second using the identity for the multiplication of two Gaussian densities given in Appendix A.1. We reiterate from our discussion of Eq. (5.16) that the term $(\hat{\boldsymbol{\Lambda}}_a + \boldsymbol{\Sigma})^{-1}$ can be calculated in a numerically stable manner as $\boldsymbol{\Lambda}_a^{-1}(\boldsymbol{\Sigma}\hat{\boldsymbol{\Lambda}}_a^{-1} + \mathbf{I})^{-1}$. Now turning our attention to the second integral we get

$$\begin{aligned}
\mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[k''_a(\mathbf{p}, \tilde{\mathbf{p}})\big] &= \boldsymbol{\Lambda}_a^{-1}\int k_a(\hat{\mathbf{p}})\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}} \\
&\qquad - \boldsymbol{\Lambda}_a^{-1}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\int\hat{\mathbf{p}}\hat{\mathbf{p}}^\top k_a(\hat{\mathbf{p}})\mathcal{N}(\hat{\mathbf{p}}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\mathrm{d}\hat{\mathbf{p}}\begin{bmatrix}\mathbf{I} \\ -\mathbf{I}\end{bmatrix}\boldsymbol{\Lambda}_a^{-1} \\
&= \mathbb{E}_{\mathbf{p},\tilde{\mathbf{p}}}\big[k_a(\mathbf{p}, \tilde{\mathbf{p}})\big]\left(\boldsymbol{\Lambda}_a^{-1} - \boldsymbol{\Lambda}_a^{-1}\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}\Big(\hat{\boldsymbol{\Sigma}} + \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^\top\Big)\begin{bmatrix}\mathbf{I} \\ -\mathbf{I}\end{bmatrix}\boldsymbol{\Lambda}_a^{-1}\right) \qquad (5.27)
\end{aligned}$$

where $\hat{\boldsymbol{\Sigma}} := \hat{\boldsymbol{\Lambda}}_a(\hat{\boldsymbol{\Lambda}}_a + \boldsymbol{\Sigma})^{-1}\boldsymbol{\Sigma}$ is obtained from the results in Appendix A.1. Again, we can write this term as $(\boldsymbol{\Sigma}\hat{\boldsymbol{\Lambda}}_a^{-1} + \mathbf{I})^{-1}\boldsymbol{\Sigma}$ for a numerically stable calculation.

Additive Squared Exponential Kernel

The derivation of these equations is again closely related to the squared exponential and can be obtained by following a similar procedure to Sec. 3.5.

5.5.3 Local Model Validation

The framework outlined in the previous section can be used to validate a given continuous-time local linear model given sampled observations. First, consider a local linear model of the form

$$\dot{\mathbf{x}} = \mathbf{o}_{\mathrm{m}} + \mathbf{W}(\mathbf{z} - \mathbf{z}_{\mathrm{m}})$$

based around the operating point $\mathbf{z}_{\mathrm{m}}$ where the offset term $\mathbf{o}_{\mathrm{m}}$ is zero if $\mathbf{z}_{\mathrm{m}}$ is an equilibrium of the system. We could create an artificial data set $\mathcal{D} = \{\mathbf{Z}_{\mathrm{m}}, \mathbf{Y}_{\mathrm{m}}\}$ defined by

$$\mathbf{Z}_{\mathrm{m}} = \begin{bmatrix} \bar{\mathbf{z}}^{\top} \\ \bar{\mathbf{z}}^{\top} \otimes \mathbf{1}_D \end{bmatrix} \in \mathbb{R}^{(D+1)\times D} \quad \text{and} \quad \mathbf{Y}_{\mathrm{m}} = \begin{bmatrix} \bar{\mathbf{o}}^{\top} \\ \mathbf{W}^{\top} \end{bmatrix} \in \mathbb{R}^{(D+1)\times E}$$

The framework outlined in the previous section then allows us to calculate the joint prior distribution with a standard data set

$$\begin{bmatrix} \vec{\mathbf{Y}}_{\mathrm{m}} \\ \vec{\mathbf{Y}} \end{bmatrix} \Bigg| \mathbf{Z}_{\mathrm{m}}, \mathbf{Z}, \boldsymbol{\theta} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}(\mathbf{Z}_{\mathrm{m}}) \\ \mathbf{m}(\mathbf{Z}) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(\mathbf{Z}_{\mathrm{m}}, \mathbf{Z}_{\mathrm{m}}) + c_{\mathrm{m}}^{-1}\mathbf{I} & \mathbf{K}(\mathbf{Z}_{\mathrm{m}}, \mathbf{Z}) \\ \mathbf{K}(\mathbf{Z}, \mathbf{Z}_{\mathrm{m}}) & \mathbf{K}(\mathbf{Z}, \mathbf{Z}) + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \otimes \mathbf{I} \end{bmatrix} \right)$$

where $\mathbf{m}$ and $\mathbf{K}$ apply the appropriate operations described in the previous sections. The key thing to note here is the introduction of a “confidence” parameter c_{m} which we can consider as an additional parameter appended to the hyperparameter vector $\boldsymbol{\theta}$. Whilst training, this parameter can be automatically tuned through maximisation of the marginal likelihood as described in Sec. 3.4.4. The learned parameter will then reflect how well the given local model agrees with the observed data. For example, if $c_{\mathrm{m}} \to 0$ the training process has determined that the given model does not agree with the data. Conversely, if c_{m} becomes very large, the given model closely agrees with the data.

5.6 Summary

We have defined a new class of Gaussian process priors for sampled continuous-time systems. The problem we were addressing was how we could exploit useful structure in the continuous-time dynamics of a system in which we only have access to sampled data and must make predictions in discrete-time. In particular, we wanted to exploit this structure for training models with greater marginal likelihood and improved predictive performance. We achieved this objective by finding an approximate relationship, based on the Runge-Kutta family of methods, between a Gaussian process prior placed over the continuous dynamics and the associated prior over the sampled system.

The effectiveness of this method was demonstrated on data sets obtained from the pendulum system. The continuous-time pendulum dynamics can be decomposed into a linear part plus an (additive) nonlinear part. We showed that a standard discrete-time prior often failed to pick out this structure and the invariances in the equations. On the other hand, our Runge-Kutta class of priors were able to learn hyperparameters that corresponded very closely to the underlying continuous-time structure, which led to models with greater marginal likelihood than the standard method. As we would expect, this improvement increased with the order of the RK method being utilised, but at the expense of increased computational load. We observed improvement not only in the marginal likelihood of the model but also in terms of predictive performance.

The class of priors we have defined can additionaly handle non-uniformly sampled data sets and data sets with mixtures of continuous and sampled data. This may be especially useful in the case where we would like to test how well a given continuous local-linear model of a system agrees with the observed sampled data. We ended by designing an algorithm to achieve such a test and to return a parameter which puts a numerical value on how well a given local model agrees with the observed data.

CHAPTER 6 Case Studies

6.1 Introduction

In this chapter we provide a more detailed discussion on how to apply the learning algorithm we have been developing to a given problem. Firstly, we shall provide some general guidelines for tuning the learning control approach. This will be followed by a comparison of our method with a standard control theoretic approach to the same problem. These comparisons will consist of a discussion regarding the entire design process from beginning to end and the component parts involved. We aim to provide insight into the advantages and disadvantages associated with applying each methodology, as a non-expert, to a given problem. The case studies we investigate are the unicycle control problem, introduced in Sec. 4.3.4, and a simulated process control problem: the evaporator. As we have already discussed the unicycle control task we shall give this problem less attention than the evaporator. With the evaporator we additionally aim to demonstrate the power of the learning control approach in two ways: the ability to learn more exotic control policies than a standard affine structure and to demonstrate a level of robustness in the face of erroneous prior information.

6.2 Guidelines for Tuning

We begin this chapter with a breakdown of the elements of the learning algorithm that need to be chosen a priori and some guidelines for how to choose or initialise them.

- **Prior Over Dynamics.** Choosing an appropriate prior over dynamics functions is one of the most involved aspects of setting up the algorithm. It can be broken up as follows:
  - *State Representation.* This is, in general, a difficult question in which we usually rely on domain specific knowledge of the problem in order to define the state. Once we have the state we need to determine how to map the current state and action to the next state. We usually do this in two stages: predict the difference between the next and current state using a zero mean Gaussian process prior then reconstruct the next state from this prediction.
  - *Known Relationships.* If there are known relationships between a subset of the state variables these can be included directly using the framework of Chapter 4.
  - *Angular Variables.* These can be dealt with using a sine/cosine representation of the angle at the input to the GP model which can then be reconstructed at the next timestep using simple linear and trigonometric relationships.
  - *Hyperparameters.* To initialise the length-scale and output variance hyperparameters of a standard squared exponential kernel a good starting place is to set them to the standard deviations of the input and output training data sets respectively.
  - *Pseudo-Inputs.* The number of pseudo-inputs used when sparse approximations are employed depends on how much computational saving is required. For the unicycle problem we found that 300 was more than ample to capture all salient features. The locations were initialised as a random subset of the full training data set.
- **Cost Function.** As discussed in Sec. 3.2.1, an appropriate form of stage-cost is the inverted-Gaussian as it has useful properties in terms of the exploration-exploitation tradeoff. The width of this stage-cost should be chosen in such a way as to cover the "normal" region of operation of the system, so that the gradients involved in the optimization stage are not negligible. The prediction horizon should be set as long as possible under the computational constraints of the offline simulation phase.
- **Policy Structure.** The rule of thumb we use when picking a policy structure is to be as simple as possible. As we increase the complexity from a standard affine structure to a that of a radial basis function there can be a huge increase in the number of parameters that need to be optimised, therefore a larger space to search for a control solution and a potential increase of local optima. However, for some problems a nonlinear policy structure is necessary e.g. the pendulum swing-up task.
- **Sampling Time.** The sampling time should be chosen as long enough such that the computation in the prediction, or evaluation, phase is as low as possible but short enough for the control task to be achievable. This can be a tricky parameter to tune. For example, if we used a value of 0.2 s for the unicycle then the control task appeared to be impossible, but a value of 0.1 s led to impractical computation time.

- **Interaction Phase.** The goal of the interaction phase is to run the current control policy long enough to capture useful new information but not long enough so that the training data set gets too big too quickly. Again, the upper limit is controlled by the computational complexity of the simulation phase. An example of one heuristic we used to determine this run-time for the unicycle example is given in Sec. 4.3.4.

- **Simulation Phase.** A final important parameter is the number of function evaluations, or line searches, we limit the gradient descent optimisation scheme to. One rule of thumb when choosing this parameter is that in a standard conjugate gradient scheme it takes around the same number of function evaluations as there are policy parameters to be optimised in order to reach a reasonable approximation of the Hessian matrix. Therefore, setting the number of function evaluations to at least twice this number worked well in the problems we tackled.

With these tuning rules in mind we shall now move onto the case studies themselves to gain some further practical insight into this method.

6.3 Unicycle

6.3.1 Problem Overview

We shall now discuss the issues involved in learning control policies for the unicycle problem we outlined in Sec. 4.3.4. For ease of reference we include the figure from Sec. 4.3.4 again here, in Fig. 6.1. Remember that the only control actions we may apply are a torque on the wheel u_w and a torque on the turntable at the top of the unicycle u_t. Therefore we do not have any direct control of the rolling motion θ, which makes the control task intrinsically difficult. In order to understand more fully the difficulty of this control problem we begin with working through a procedure of how to apply standard linear control theory to the problem. This will be followed by a discussion of the associated issues with applying our learning algorithm. We note that all the following analysis is carried out with respect to the nonlinear equations of motion derived by Forster (2009). These equations and their derivation are given a full treatment in Appendix B.3.

6.3.2 Linear Control Theory

Linearised Model Selection

From a standard control theory perspective, the first step towards designing a stabilising policy is usually to obtain a linear model of the system which describes the dynamics in the region of some operating point $\bar{\mathbf{z}}$ in the state-action space. Recall that for the unicycle, the state is given by $\mathbf{x} = [\phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}, \dot{\phi}_\mathrm{w}, \dot{\psi}_\mathrm{t}, x_\mathrm{c}, y_\mathrm{c}]^\top \in \mathbb{R}^{10}$ with pitch angle ϕ, roll angle θ, yaw angle ψ, wheel

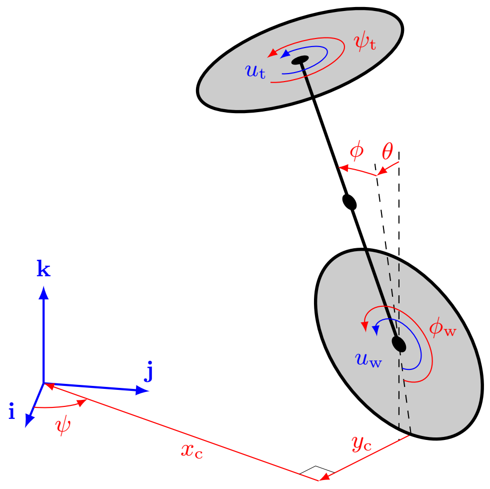


Figure 6.1: The robotic unicycle. The spatial position of the unicycle is defined by the pitch angle ϕ, roll angle θ, yaw angle ψ, wheel angle ϕ_w, turntable angle ψ_t and the body-centred coordinates of the global origin $(x_\mathrm{c}, y_\mathrm{c})$. The controlled inputs are the motor torque applied to the wheel u_w and the motor torque applied to the turntable u_t.

angle ϕ_w, turntable angle ψ_t and the location of the origin $(x_\mathrm{c}, y_\mathrm{c})$ with respect to a body-fixed reference frame. These are depicted in Fig. 6.1.

A reasonable state-action location around which to find a linearised model of the system is $\bar{\mathbf{z}} = \mathbf{0}$. In this state the unicycle is stationary and upright. To obtain a linearised model of the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ from the full nonlinear equations $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ we apply the following procedure of numerical integration to get the columns of the $\mathbf{A}$ and $\mathbf{B}$ matrices

$$
\begin{aligned}
A[:, i] &= \Big(\mathbf{f}(\bar{\mathbf{x}} + \epsilon \mathbf{e}_i, \bar{\mathbf{u}}) - \mathbf{f}(\bar{\mathbf{x}} - \epsilon \mathbf{e}_i, \bar{\mathbf{u}})\Big)/2\epsilon, i \in \mathbb{Z}_{[1,E]} \\
B[:, j] &= \Big(\mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}} + \epsilon \mathbf{e}_j) - \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}} - \epsilon \mathbf{e}_j)\Big)/2\epsilon, j \in \mathbb{Z}_{[1,F]}
\end{aligned}
$$

where ϵ is some small scalar perturbation and $\mathbf{e}_i$ is the i^th column of the identity matrix of appropriate dimensions. Applied to the unicycle equations and using $\epsilon = 10^{-6}$ leads to the linearised state space system

$$
\underbrace{\begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \dot{\theta} \\ \ddot{\theta} \\ \dot{\psi} \\ \ddot{\psi} \\ \ddot{\phi}_\mathrm{w} \\ \ddot{\psi}_\mathrm{t} \\ \dot{x}_\mathrm{c} \\ \dot{y}_\mathrm{c} \end{bmatrix}}_{\dot{\mathbf{x}}}
=
\underbrace{\begin{bmatrix}
\cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
79.4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 10.7 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
-207 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -0.225 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}}_{\mathbf{A}}
\underbrace{\begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \\ \dot{\phi}_\mathrm{w} \\ \dot{\psi}_\mathrm{t} \\ x_\mathrm{c} \\ y_\mathrm{c} \end{bmatrix}}_{\mathbf{x}}
+
\underbrace{\begin{bmatrix}
\cdot & \cdot \\
\cdot & -1.40 \\
\cdot & \cdot \\
\cdot & \cdot \\
\cdot & \cdot \\
-2.23 & \cdot \\
\cdot & 4.21 \\
7.23 & \cdot \\
\cdot & \cdot \\
\cdot & \cdot
\end{bmatrix}}_{\mathbf{B}}
\underbrace{\begin{bmatrix} u_\mathrm{w} \\ u_\mathrm{t} \end{bmatrix}}_{\mathbf{u}}
$$

where the dots indicate zero terms. It is clear from inspection that this is an uncontrollable system of equations. In particular, we cannot control the roll angle θ or the velocity $\dot{\theta}$ since they are independent of the rest of the dynamics and cannot be influenced directly through the available actuators. This would be ok if they formed a stable sub-system, however this is clearly not the case. To see why, consider a small positive perturbation of θ from its equilibrium, this leads to a positive acceleration through the 10.7 term and therefore θ will continue to increase in an unbounded fashion. In other words, the unicycle will fall over.

From the above analysis we evidently require a different operating point in order to find a stabilisable linear system. From inspection of the nonlinear equations of motion, found in Appendix B.3, we can observe that the rotational motion governed by $\psi, \dot{\psi}, \dot{\psi}_\mathrm{t}$ and u_t is decoupled from the other states when the unicycle is upright i.e. $\theta = \phi = 0$. This linearised subsystem is governed by the integrator equations

$$\begin{bmatrix} \dot{\psi} \\ \ddot{\psi} \\ \ddot{\psi}_\mathrm{t} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \psi \\ \dot{\psi} \\ \dot{\psi}_\mathrm{t} \end{bmatrix} + \begin{bmatrix} 0 \\ -2.23 \\ 7.23 \end{bmatrix} \begin{bmatrix} u_\mathrm{t} \end{bmatrix}$$

We could then set the unicycle spinning at some constant velocity by applying some pulse action through u_t. For example, applying $u_\mathrm{t} = 10\,\mathrm{N\,m}$ for $0.1\,\mathrm{s}$ would lead to $\dot{\psi} = -2.23\,\mathrm{rad\,s^{-1}}$ and $\dot{\psi}_\mathrm{t} = 7.23\,\mathrm{rad\,s^{-1}}$. Note that this system is only controllable along a single dimension of the two dimensional space. Now if we linearised around this operating point the other states would be governed by the following system of equations

$$\underbrace{\begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \dot{\theta} \\ \ddot{\theta} \\ \ddot{\phi}_\mathrm{w} \\ \dot{x}_\mathrm{c} \\ \dot{y}_\mathrm{c} \end{bmatrix}}_{\dot{\mathbf{x}}_\mathrm{rot}} = \underbrace{\begin{bmatrix} \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 85.2 & \cdot & \cdot & -4.94 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 3.40 & 15.7 & \cdot & 0.548 & \cdot & \cdot \\ -209 & \cdot & \cdot & -3.12 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & -0.225 & \cdot & -2.23 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 2.23 & \cdot \end{bmatrix}}_{\mathbf{A}_\mathrm{rot}} \underbrace{\begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \dot{\phi}_\mathrm{w} \\ x_\mathrm{c} \\ y_\mathrm{c} \end{bmatrix}}_{\mathbf{x}_\mathrm{rot}} + \underbrace{\begin{bmatrix} \cdot \\ -1.40 \\ \cdot \\ \cdot \\ 4.21 \\ \cdot \\ \cdot \end{bmatrix}}_{\mathbf{B}_\mathrm{rot}} \begin{bmatrix} u_\mathrm{w} \end{bmatrix} \tag{6.1}$$

To investigate the controllability of this new system of equations we evaluated the controllability matrix $\mathcal{C} = \left[\mathbf{B}_\mathrm{rot}, \mathbf{A}_\mathrm{rot}\mathbf{B}_\mathrm{rot}, \ldots, \mathbf{A}_\mathrm{rot}^6\mathbf{B}_\mathrm{rot}\right]$. The condition for controllability is that this matrix has full row rank, and means that there exists a set of actions that can force the system into any part of the state space. This new system leads to a controllability matrix with full rank and can therefore be used in the design of a control policy for this subset of state variables. The turntable torque would simply be used to maintain the rotational equilibrium values. It is worth noting that although, in this linear model, the rotational dynamics are decoupled from the rest of the dynamics, the rotational setpoint values for $\dot{\psi}$ and $\dot{\psi}_\mathrm{t}$ will still affect the constants in the $\mathbf{A}_\mathrm{rot}$ matrix.

The process of finding an appropriate linear model to use as the basis for standard control methods clearly requires a relatively in depth knowledge of the dynamic equations governing the system. We would like to make the additional point that in this study the solution of linearising around a spinning trajectory came about when we considered the control policy obtained by our learning algorithm! The learned solution would always rotate on the spot instead of coming to a complete rest and it was actually this behaviour that pointed us to analyse the nonlinear equations to find a controllable form.

Policy Design

Now that a suitable linear model has been obtained we can think about control policy design. There is a plethora of multivariable linear control design methodologies available to us at this stage. Let us consider the commonly used Linear Quadratic Regulator (LQR) approach to the problem. The discrete-time LQR is the solution to the following optimisation problem

$$\text{minimise} \qquad J = \sum_{k=0}^{H} \mathbf{x}_k^\top \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^\top \mathbf{R} \mathbf{u}_k \tag{6.2}$$

$$\text{subject to} \qquad \mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} \tag{6.3}$$

for some suitable quadratic cost function parameterised by the matrices $\mathbf{Q}$ and $\mathbf{R}$. The optimal solution turns out to be a causal, linear control law $\boldsymbol{\pi}$ which is dependant only on the current state and may be calculated in closed form, even for the infinite horizon, $H \to \infty$, case. This problem and its solution appeared in the seminal paper by Kalman (1960). The only design choice in this case is how to choose the $\mathbf{Q}$ and $\mathbf{R}$ matrices. This is generally a heuristic process since we need to tune these parameters in order to create a policy which addresses

- **Performance:** by which we mean that the policy can actually achieve the task it was designed for
- **Constraints:** ensures that state and action constraints are not violated under "normal" operating conditions
- **Model Mismatch:** the policy must achieve the control task under the knowledge that the actual system may be significantly different

Ideally we would like to decouple these requirements and address them individually rather than simultaneously through a given set of parameters. This can be done in a variety of ways which we shall now discuss.

One way to separate the problem of constraint violation from the stage cost is to use Model Predictive Control (MPC). In a standard MPC algorithm we do not find a closed form for the feedback policy $\boldsymbol{\pi}$ but instead perform a constrained version of Eqs. (6.2)–(6.3) online over the future action sequence $\{\mathbf{u}_0 \ldots \mathbf{u}_H\}$. We then apply the first action in this optimal sequence and repeat the procedure at the following timestep. Obviously this is more computationally intensive in terms of online operation. However, if the constraints are linear then this forms a quadratic programming problem which is convex and there exist fast and efficient algorithms for solving such problems e.g. Boyd & Vandenberghe (2004).

Addressing the issue of model mismatch in the third bullet point is a more difficult issue. Methods from the field of Robust Control can tackle this issue by defining a nominal system and considering a bounded set of perturbations from this system. Control policies can then be derived which will satisfy some stability and/or performance criteria across all possible systems in the set. We note that determining an appropriate set of perturbed models can be a difficult problem.

Another common approach, instead of using a single linearised model and perturbations around it, is to find linearisations around a few operating points, solve the given LQR problem at each operating point and then smooth, or switch, between the solutions when applying them online. This is known as Gain Scheduling. How to choose the operating points and the smoothing function are of course heuristic problems. Alternatively, re-linearisation could be done in an adaptive manner online in the spirit of the self-tuning regulator except that the new model need not be based on the observed data but a linearisation of an internal nonlinear one. Again, an issue with this would be how we could guarantee the feasibility of the resulting solution. In our example of the unicycle, how do we guarantee that the linearised system is actually controllable?

We now turn to the problem of choosing a suitable sampling time δ_t. The only constraint in the case of LQR is the speed at which the feedback policy can be evaluated, which will be very fast indeed since it will simply be a single matrix multiplication. For MPC this will be more of a bottleneck since a full optimisation program needs to be solved at each timestep. However, if the problem is a quadratic program then for a problem the size of the unicycle modern algorithms should be able solve it on the order of 0.01 to 0.001 s and therefore should not be a problem in this case.

6.3.3 Learning Control

We now consider the design process for the learning algorithm we have proposed in previous chapters. The major design choices available here are the form of the stage cost c, the sampling time δ_t and the form of prior over the unknown dynamics. Since this framework makes only high level assumptions regarding the form of the dynamics and it naturally incorporates modelling uncertainty the problem of model-system mismatch has been rigorously addressed. Furthermore, action constraints can be dealt with in the manner outlined in Sec. 3.6.2. Therefore we need only use our stage-cost to encode the performance criteria.

For the case of learning a policy to balance the unicycle we can simply define a quadratic stage-cost that penalises the squared Euclidean distance of the top of the unicycle to the upright position at the origin. This is a very intuitive form of cost. However, as we discussed in Sec. 3.2.1, the quadratic stage cost may not be conducive for learning control. Therefore we use a saturated quadratic (or inverted-Gaussian) stage cost. The remaining free parameters are then to do with the width of this cost. Again, a natural choice would be to choose the two standard deviation region to cover the likely "area of operation". In practice, this intuitive choice of cost had to be tuned slightly to penalise deviation in the vertical direction more heavily to encode the notion that falling over is a more costly mistake than not being in the right location.

The second issue is how to choose the sampling time δ_t. In this instance the sampling time is very much a bottleneck of the algorithm. In order to keep the offline policy optimisations in the order of hours rather than days we were constrained to choose sampling times greater than 0.1 s. However, beyond 0.2 s the control problem becomes too difficult and our algorithm cannot find a stabilising policy using this sampling time. We emphasise that this is not a problem for the online implementation (which is simply a linear policy) but for the offline optimisation.

One of the major advantages of this data-driven approach is that we avoid many of the problems associated with using a first-principles model of the system. Attacking the problem using a first-principles model will inevitably require much more expert-knowledge on the dynamics of the system at hand and a good understanding of which particular physical assumptions are likely to be poor. The data-driven approach moves many of the assumptions to a more conceptual level and therefore the domain-specific knowledge required in terms of modelling is greatly reduced.

6.3.4 Comparison of the LQR and Learned Policies

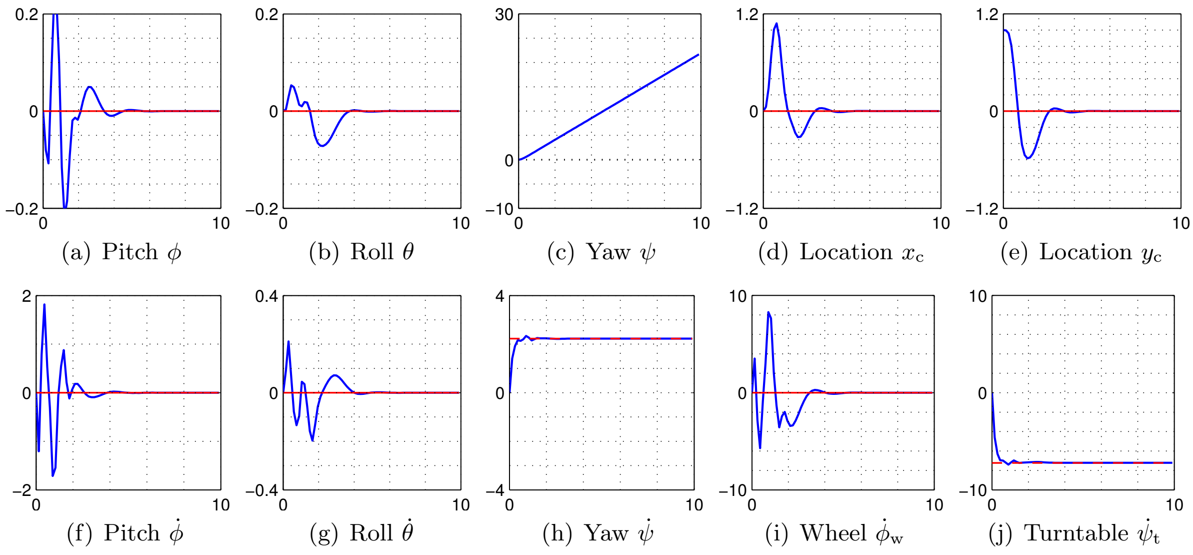
We now provide a brief comparison of the performance and policy structures obtained by the LQR approach and the learning approach outlined in the previous sections. To aid in a direct comparison we chose $\delta_t = 0.15\,\mathrm{s}$ as the sampling time for the LQR policy. To obtain the LQR policy we chose a spinning operating point of $\dot{\psi} = -2.23\,\mathrm{rad\,s^{-1}}$ and $\dot{\psi}_\mathrm{t} = 7.23\,\mathrm{rad\,s^{-1}}$ and all other states at the origin. This gave us the linear model in Eq. (6.1). A simple affine control policy mapping $\dot{\psi}$ to u_t was then designed (using LQR with a state weight of 1 and action weight of 0.1) to regulate the spinning motion.

The next step is to design an LQR policy for the other states and actions. We then chose the state weighting matrix $\mathbf{Q}$ to penalise the squared distance of the top of the unicycle to the upright position over the origin, similar to how we chose the stage-cost for the learning algorithm in Eq. (4.18). This gives us

$$\begin{aligned}\mathbf{x}_\mathrm{rot}^\top \mathbf{Q} \mathbf{x}_\mathrm{rot} &= \big(x_\mathrm{c} - r\phi\big)^2 + \big(y_\mathrm{c} - (r + r_\mathrm{w})\theta\big)^2 \\ &\approx \big(x_\mathrm{c} - r\sin\phi\big)^2 + \big(y_\mathrm{c} - (r + r_\mathrm{w})\sin\theta\big)^2 + \big((r + r_\mathrm{w}) - (r_\mathrm{w} + r\cos\phi)\cos\theta\big)^2\end{aligned}$$

for ϕ and $\theta \approx 0$. As before, r is the unicycle frame length and r_w is the radius of the wheel. We then tuned R by gradually increasing it from zero until the resulting control policy was stabilising. As a result of this procedure we chose $R = 0.001$.

The result of applying this LQR control policy to the unicycle for the task of moving from a


(a) Pitch ϕ (b) Roll θ (c) Yaw ψ (d) Location x_c (e) Location y_c

(f) Pitch $\dot{\phi}$ (g) Roll $\dot{\theta}$ (h) Yaw $\dot{\psi}$ (i) Wheel $\dot{\phi}_w$ (j) Turntable $\dot{\psi}_t$

Figure 6.2: State trajectories over 10 s of the unicycle when an LQR policy is applied in order to move from the initial position $(x_c, y_c) = (0, 1)$ to the origin. The policy is derived from a linear model obtained from a spinning operating point. The values of this operating point for each state are shown in red, and dashed lines for the spinning states $\dot{\psi}$ and $\dot{\psi}_t$.

starting position $(x_c, y_c) = (0, 1)$ to the origin is shown in Fig. 6.2. We can see that the control policy is quite aggressive but converges to the spinning operating point within 4 s. Further tuning, could of course improve this performance but will not be pursued further here. The result of applying one of the learned control policies from Sec. 4.3.4 is shown in Fig. 6.3. In this case, the trajectory it follows converges on a tight circular orbit, with a radius of around 20 cm and a roll angle of 0.1 rad, rather than spinning exactly on the origin. This solution was found the vast majority of the time when using this cost function. We note that this trajectory does not appear to be much worse in terms of stage-cost since the values $(x_c, y_c) = (0, 0.2)$ m and $\theta = 0.1$ rad yield a very small stage-cost of around 0.005. This simply highlights the issue that it is often hard to design an intuitive cost that produces a good control policy.

Finally, the resulting policy gain matrices produced by the LQR and learning methods are

$$\mathbf{K}_1^\top = \begin{bmatrix} 73.4 & 0 \\ 20.7 & 0 \\ -125 & 0 \\ -21.78 & 0 \\ 0 & 0 \\ 0 & 1.91 \\ 3.54 & 0 \\ 0 & 0 \\ -2.82 & 0 \\ 4.43 & 0 \end{bmatrix} \qquad \mathbf{K}_2^\top = \begin{bmatrix} 79.8 & -4.18 \\ 11.8 & 4.51 \\ 26.9 & -93.7 \\ 1.31 & -28.7 \\ 0.08 & -0.09 \\ -3.35 & 4.72 \\ 1.15 & 0.98 \\ -0.69 & 0.90 \\ -4.37 & 1.15 \\ -3.09 & 4.08 \end{bmatrix}$$

respectively. Note that the columns of the gain matrices (rows of the transposed matrices)

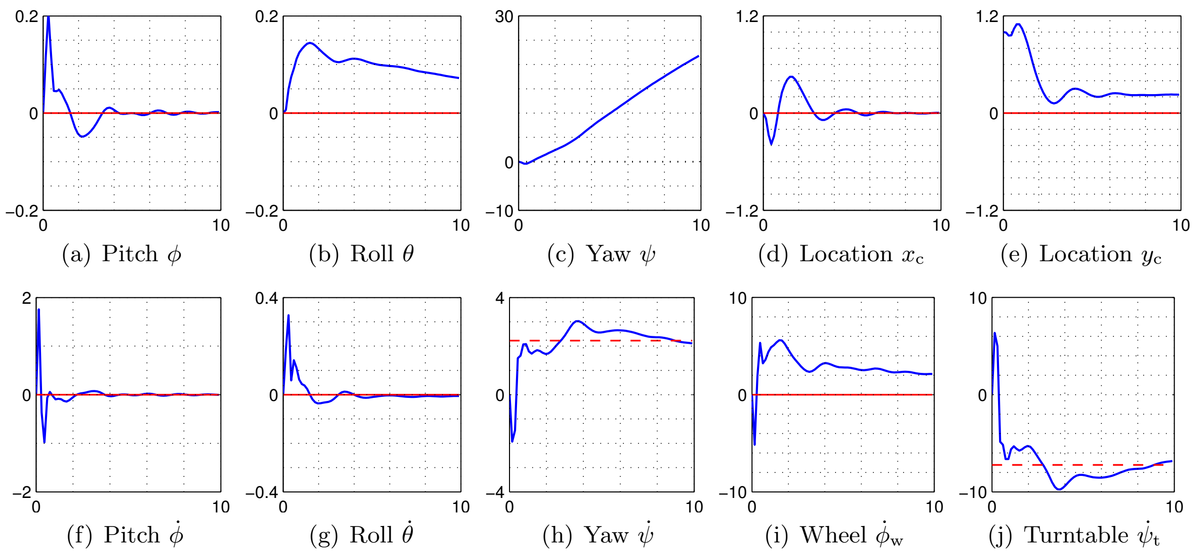


Figure 6.3: State trajectories over 10 s of the unicycle when a learned affine policy is applied in order to move from the initial position $(x_c, y_c) = (0, 1)$ to the origin. The values of the operating point used for the LQR policy, from Fig. 6.2, for each state are shown in red, and dashed lines for the spinning states $\dot{\psi}$ and $\dot{\psi}_t$.

correspond to elements of the state vector $\mathbf{x} = [\phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}, \dot{\phi}_w, \dot{\psi}_t, x_c, y_c]^\top$. We can observe that the learning algorithm has utilised all the states available to it as feedback terms whereas, due to the way we structured the problem the LQ policy is sparse. We also note an issue with the the learned gain matrix $\mathbf{K}_2$. Since there are non-zero feedback gains on the yaw angle ψ, the orbital trajectory will not go on indefinitely since these feedback terms will begin to dominate the control actions (in fact the unicycle falls over at around 30 s). However, since this is not an issue over the ten second horizon it is not picked up by the algorithm. This is an example of one of the potential issues with optimising over a finite prediction horizon.

6.4 Process Control

6.4.1 Problem Overview

We now consider a significantly different control problem, that of a forced-circulation industrial evaporator as outlined by Newell & Lee (1989) and depicted in Fig. 6.4. This problem was used to demonstrate the capabilities of nonlinear MPC in Maciejowski (2002). The full equations of motion and values for the physical constants can be found in Appendix B.4.

The system works as follows: a feed stream enters the system with concentration X_1, temperature T_1 and flow rate F_1. This is mixed with recirculating liquid and is pumped through the evaporator at a flow rate F_3. The evaporator is a heat exchanger with internal pressure P_2 and is heated by steam with flow rate F_{100}, pressure P_{100} and temperature T_{100}. The feed and recirculated liquid is boiled inside the evaporator and the resulting mixture of vapour and liquid enters the

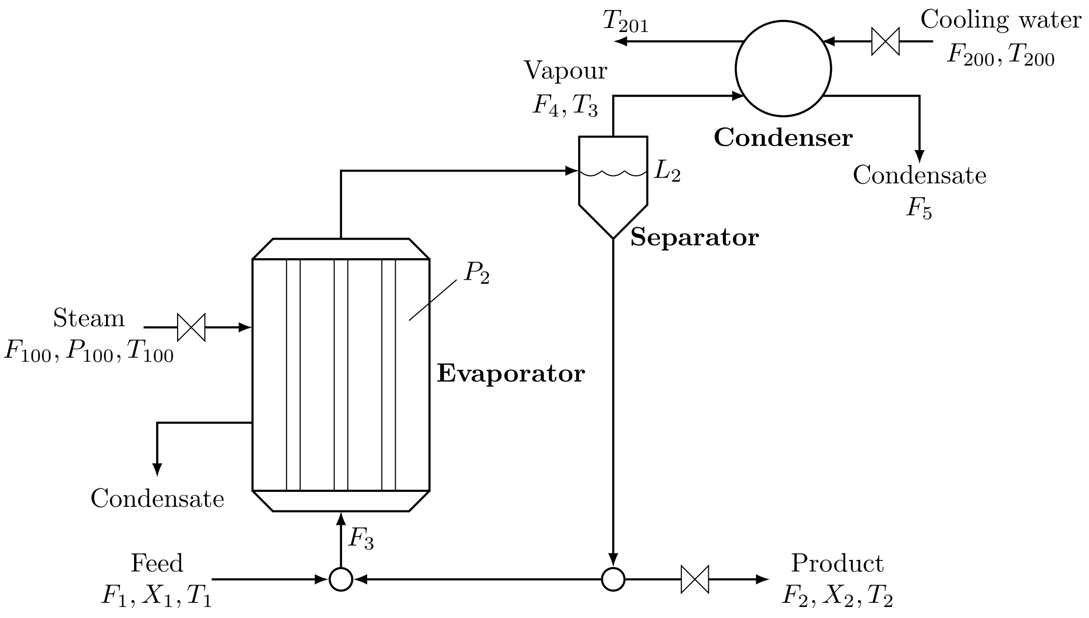


Figure 6.4: Schematic of the forced-circulation evaporator model of Newell & Lee (1989). The model consists of three distinct components: the evaporator, separator and condenser. The state of the system is made up of the product composition X_2, the evaporator pressure P_2 and the separator level L_2. The controlled inputs are product flow rate F_2, steam pressure P_{100} and the cooling water flow rate F_{200} which are applied through servomechanisms.

separator with liquid level L_2. A small proportion of the liquid from the separator is drawn off as product, with concentration X_2, temperature T_2 at flow rate F_2. The vapour drawn off from the separator flows to a condenser at flow rate F_4 and temperature T_3 where it is condensed by cooling water with flow rate F_{200}, entry temperature T_{200} and exit temperature T_{201}.

This system has three states consisting of the product composition X_2, the evaporator pressure P_2 and the separator level L_2. The choice of these states is clear when we consider the equations in Appendix B.4. The control actions are chosen to be the product flow rate F_2, steam pressure P_{100} and the cooling water flow rate F_{200}. The remaining inputs act as unmeasureable disturbances on the system.

The control task is to regulate the product composition X_2 and the pressure inside the evaporator P_2 to some given setpoints while keeping the level of the separator tank L_2 within its hard constraints. Control actions are applied in a zero-order hold fashion with a sampling time of $\delta_t = 1\,\text{min}$. Further, each control action is constrained by hard limits. The numerical values of these hard constraints can be found in Appendix B.4.

6.4.2 Linear Control Theory

Model Identification

The derivation of the nonlinear equations of motion of the system given in Appendix B.4 requires a reasonably detailed understanding of the physics involved in the process, including mass and energy balance equations. It also requires significant understanding of the system to know whether these equations will capture all the salient features of the actual plant. However, once we have this model it will be relatively easy to tune any uncertain parameters, such as physical constants, from observed data.

An alternative to using this first principles nonlinear model is to use techniques from System Identification. The vast majority of these methods focus on the identification of linear systems and address problems such as how to obtain a sufficiently rich training data set from a system operating in closed loop. The simplest example is the *step-test* procedure, in which the tranfer function between each action and state is assumed to be first (or second) order and the parameters are directly estimated from the associated step responses. This makes very strict assumptions on the form of the dynamics but can often provide a model that is good enough for control design. More sophisticated techniques, such as *subspace methods*, learn the parameters of a linear state-space model directly. For an accessible introduction to System Identification we direct the reader to the text by Ljung (1999).

One of the issues with identifying a linear model is how to ensure that the data we use for identification comes from a linear regime of the system. In many applications the nonlinearities are not aggressive enough to cause too much concern. A potential solution is to first fit a general nonlinear model to the data then linearise the model. However, it is generally an easier problem to confront the issues of obtaining a linear data set than to face the ones involved in choosing an appropriate nonlinear model. This issue could be alleviated through the use of a nonparametric modelling method such as Gaussian processes.

If we were to consider the modelling process from the very beginning, a major question is: how do we actually choose an appropriate representation of the state-space that will capture all the relevant features of the system? The choice of X_2, P_2 and L_2 came from the expert knowledge used to derive the first principles model outlined in Appendix B.4. However, this may not be a full enough representation in real life. For example, if we consider the fact that the actions will be applied through some servomechanisms with dynamics of their own it will be necessary to augment the state space to take account of this. Now, assuming we do have a sufficient state-representation and a model we can move onto the design of a control policy.

Control Policy Design

A common approach to controller design in industry is to split the multivariable loop into Single Input Single Output (SISO) loops by assigning each control action to one of the states. There are then many techniques available for tuning each individual feedback loop, for example PID controllers tuned using the Zeigler-Nichols rules. In reality, it may be very difficult to isolate such loops and there will, almost certainly, be coupling between each loop, the effect of which will have to be accounted for in a heuristic manner. Newell & Lee (1989) discuss a method of achieving this for the evaporator. A fairly clear choice of SISO loops for the evaporator would be to control the separator level L_2 with the product flow rate F_2, the evaporator pressure P_2 with the input steam pressure P_{100} and the product composition X_2 with the cooling water flow rate F_{200}. This choice could be obtained by looking at the physical system process in Fig. 6.4. Thus begins an iterative procedure of closing the feedback loop on each pair in turn then returning to the first one and repeating the process.

Applying a multivariable control design technique, such as LQR, circumvents the above issues. Standard LQR provides only proportional feedback terms and therefore cannot guarantee offset free setpoint tracking. To address this, we can explicitly incorporate integral action into the LQ tracking problem using the augmented dynamics trick by defining the state-space system

$$\begin{bmatrix} \mathbf{x}^{\mathbf{x}}_{k+1} \\ \mathbf{x}^{\mathbf{r}}_{k+1} \\ \mathbf{x}^{\mathbf{e}}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^{\mathbf{r}} & \mathbf{0} \\ \mathbf{C} & -\mathbf{C}^{\mathbf{r}} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}^{\mathbf{x}}_{k} \\ \mathbf{x}^{\mathbf{r}}_{k} \\ \mathbf{x}^{\mathbf{e}}_{k} \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \mathbf{u}_k \tag{6.4}$$

where $\mathbf{x}^{\mathbf{x}}$ is the system state, $\mathbf{x}^{\mathbf{r}}$ is the reference state and $\mathbf{x}^{\mathbf{e}}$ is integral of the error $\mathbf{C}\mathbf{x}^{\mathbf{x}} - \mathbf{C}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}$. The integral of the error can be written in discrete-time as

$$\mathbf{x}^{\mathbf{e}}_k = \sum_{i=0}^{k} \mathbf{C}\mathbf{x}^{\mathbf{x}}_i - \mathbf{C}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_i$$

If we consider the tracking of a fixed state setpoint then $\mathbf{A}^{\mathbf{r}} = \mathbf{C}^{\mathbf{r}} = \mathbf{C} = \mathbf{I}$ and the elements of $\mathbf{x}^{\mathbf{r}}$ contain the setpoints of the relevant state variables. We can then solve a standard LQR problem in this augmented state. This approach effectively introduces a new set of integral terms which will help acheive offset-free tracking. Newell & Lee (1989) report a significant performance improvement of this method over the standard SISO PID loops, even with naïvely chosen state and action weighting matrices.

These local control laws may only provide good performance in the region of the state space in which the linearised model was obtained. Therefore we need to address the issue of changing setpoint values that take us into regions of the state space where this model is no longer useful. This issue is traditionally addressed through the use of Gain Scheduling. A number of linear control laws are designed for different setpoint values, then a heuristic smoothing scheme is designed to switch between them. Alternatively, one could use a known nonlinear model online to obtain linearised models based on where we are in the state space. The control actions could then be determined using, for example, an LQR or MPC scheme.

6.4.3 Learning Control

Problem Setup

We now attack this problem from a learning control perspective. Assuming that we have some representation of the state, and have been given a set of control actions we need to design an appropriate prior over system dynamics $p(\mathbf{f})$, stage-cost $c(\cdot)$ and policy structure $\boldsymbol{\pi}(\cdot)$. We shall talk about each one of these considerations in turn. We note that, as in the previous section, a major issue with tackling this problem using minimal expert knowledge is how to actually choose an appropriate representation for the state. In this case, there is no obvious reasoning to choose X_2, P_2 and L_2 without a reasonable understanding of the physics of the problem. Therefore, we shall assume that this state representation is given.

Prior Over Dynamics

We choose a completely general squared-exponential prior over the unknown dynamics. We can achieve reference tracking and controller integral action using the framework of augmented dynamics outlined in Chapter 3 and in a similar spirit to Eq. (6.4). We will use the following sub-dynamics models

$$\begin{aligned}
\mathbf{x}^{\mathbf{x}}_{k+1} &= \mathbf{f}_1\big(\mathbf{x}^{\mathbf{x}}_k, \mathbf{u}_k\big) \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_{\text{SE}}) \\
\mathbf{x}^{\mathbf{r}}_{k+1} &= \mathbf{f}_2\big(\mathbf{x}^{\mathbf{r}}_k\big) = \mathbf{A}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_k \\
\mathbf{x}^{\mathbf{e}}_{k+1} &= \mathbf{f}_3\big(\mathbf{x}^{\mathbf{x}}_k, \mathbf{x}^{\mathbf{r}}_k, \mathbf{x}^{\mathbf{e}}_k\big) = (\mathbf{C}\mathbf{x}^{\mathbf{x}}_k - \mathbf{C}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_k) + \mathbf{x}^{\mathbf{e}}_k
\end{aligned}$$

where we learn the main system dynamics with a Gaussian process, we will define an appropriate linear model for the reference state and the integral of the error is simply another linear relationship. If we consider reference tracking on X_2 and P_2 only then $\mathbf{C} = [\mathbf{I}, \mathbf{0}] \in \mathbb{R}^{2\times 3}$.

We could also encode an approximating linear model for the servomechanisms by adding a fourth sub-dynamics and altering the first as follows

$$\begin{aligned}
\mathbf{x}^{\mathbf{x}}_{k+1} &= \mathbf{f}_1\big(\mathbf{x}^{\mathbf{x}}_k, \mathbf{x}^{\mathbf{u}}_k, \mathbf{u}_k\big) \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_{\text{SE}}) \\
\mathbf{x}^{\mathbf{u}}_{k+1} &= \mathbf{f}_4\big(\mathbf{x}^{\mathbf{u}}_k, \mathbf{u}_k\big) = a\mathbf{x}^{\mathbf{u}}_k + (1-a)\mathbf{u}_k
\end{aligned}$$

where $a = \exp(-\delta_t/\tau_u)$, τ_u is our approximate time lag, $\mathbf{x}^{\mathbf{u}}$ is that action actually applied to the system through the servomechanism and $\mathbf{u}$ is now the input to the servo itself. Clearly as $\tau_u \to 0$ the dependence of $\mathbf{f}_1$ on $\mathbf{x}^{\mathbf{u}}$ will decrease and it will be reasonable to simply ignore its effect.

Stage-Cost Function

We shall split our stage-cost into two parts $c(\mathbf{x}) = c_1(\mathbf{x}) + c_2(\mathbf{x})$. The first penalises deviations from the X_2 and P_2 setpoints. We use the inverted-Gaussian stage-cost from Eq. (3.7) where $\mathbf{Q}$

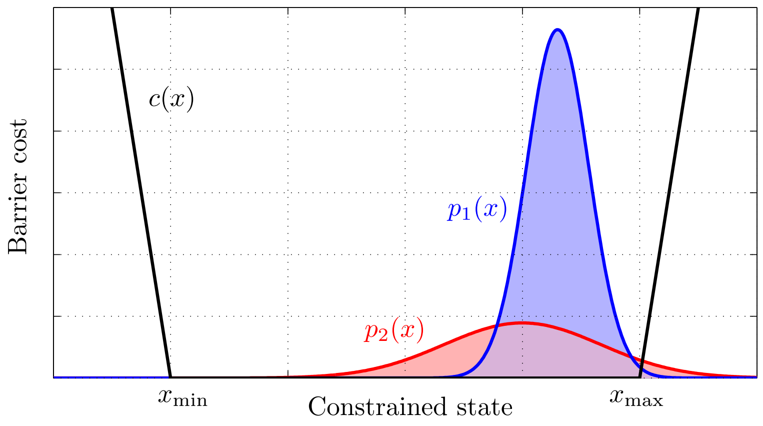


Figure 6.5: Illustration of the intrinsic caution introduced when using barrier functions as soft constraints. Note that $p_2(x)$ will be penalised more heavily than $p_1(x)$ even though its mean is further away from the constraint edge.

is chosen such that the one-standard deviation limits are $\pm 10\%$ on X_2 and ± 20 kPa on P_2, with respect to the given setpoint values. This is because we do not expect to be operating more than $\pm 20\%$ or 40 kPa from a given setpoint and therefore this is an appropriate metric for choosing the "2σ region" of our cost. Deviation from the L_2 setpoint is not explicitly penalised since we only really care that the level stays within its limits.

The second part of the cost accounts for the constraints on P_2 and L_2. For this we use the affine cost given in Eq. (3.62) with slopes of 10 kPa per unit cost on P_2 and 0.1 m per unit cost on L_2 in the constraint violation regions. We are aiming to keep them within $P_2 \in [0, 100]$ kPa and $L_2 \in [0, 4]$ m therefore the corners of the affine regions will be placed at these locations. A more cautious approach would place the corners on the interior of these regions. An example of this cost function is shown in Fig. 6.5 where we can see that the policy should adopt cautious behaviour when acting close to the state boundaries. In other words, it should only operate close to a state boundary if it is certain of the state trajectory that will be taken.

Policy Structure

We shall consider an affine policy structure. This will not have access to a preview horizon of the reference and therefore is a function of $\mathbf{x}^{\mathbf{x}}, \mathbf{C}^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}$ and $\mathbf{x}^{\mathbf{e}}$ to allow for the possibility of integral action. Since we will only be tracking setpoints on X_2 and P_2 we only require a reference and integral action on these dimensions therefore our policy will have to map seven state variables (the three system states, two reference states and two integral states) to the three action variables.

We note that for this application an affine policy may not be flexible enough to track setpoints across a variety of regions in the state space. In the situations where a linear policy is not sufficient we could use a general Radial Basis Function (RBF) policy. However, this may be unnecessarily flexible and requires basis functions spread across the whole state space. In our seven dimensional input space this would require $n_c = 10^7$ basis functions just to provide a grid

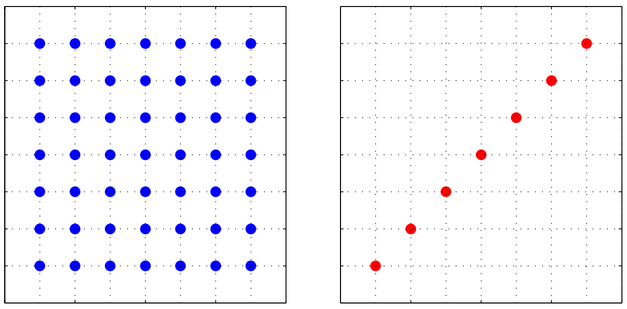

(a) RBF (b) Additive RBF

Figure 6.6: Basis function locations to provide a grid over a two dimensional space for a standard RBF compared with an additive RBF.

of 10 points in each dimension. An alternative class of functions which does not suffer from the same curse of dimensionality is the family of additive RBFs. An example of a single output first order additive Gaussian-RBF is

$$\pi(\mathbf{x}) = \sum_{i=1}^{n_c} w_i \sum_{d=1}^{E} \alpha_d^2 \exp\left(\tfrac{1}{2}\big(x[d] - \mu_i[d]\big)^2 / \lambda_d^2\right) \tag{6.5}$$

with n_c basis functions each with individual weights w_i and centres $\boldsymbol{\mu}_i$. These functions can are of the general form $\boldsymbol{\pi}(\mathbf{x}) = \boldsymbol{\pi}_1\big(x[1]\big) + \cdots + \boldsymbol{\pi}_E\big(x[E]\big)$ and are therefore more flexible than affine policies but more restricted than a general RBF. In this case we only have to provide basis functions along a single dimensional subspace. In our example, the ten dimensional grid could be achieved with only ten basis functions!

Safe Implementation

A further consideration due to the nature of the problem is that during the learning phase we cannot allow the system to "crash"! To address this issue we will assume that there is a working control policy (possibly a human operator) already on the plant from which we can obtain an initial training data set and can hand control back to during the learning phase. We will have to define, in a heuristic manner, boundaries on the states and actions which, if violated, the system will hand control back to the original control policy. Learning can then proceed in a safe manner, provided that the original control policy can react fast enough to recover the system from the boundary of the violation.

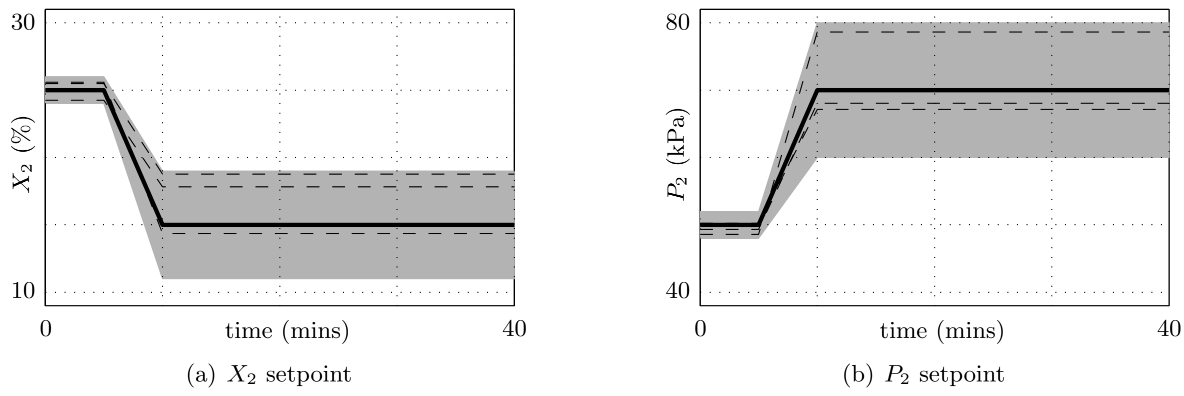


Figure 6.7: Distributions over possible setpoint changes for the states X_2 and P_2. The change takes place over a period of 5 mins using a ramp function. The thick black line depicts the mean of the change, the gray areas denotes the 2σ region and samples from these distributions are given by dashed lines.

6.4.4 Application of Learning Control

Experimental Setup

We now set out to implement learning control on a simulated model of the evaporator. The simulated model was based on the equations defined in Appendix B.4 with the controlled inputs applied through servomechanisms modelled as first-order lags with time constant τ_u. We limited the number of function evaluations in the policy optimisation stage to 50 in order to place a limit on the computation time.

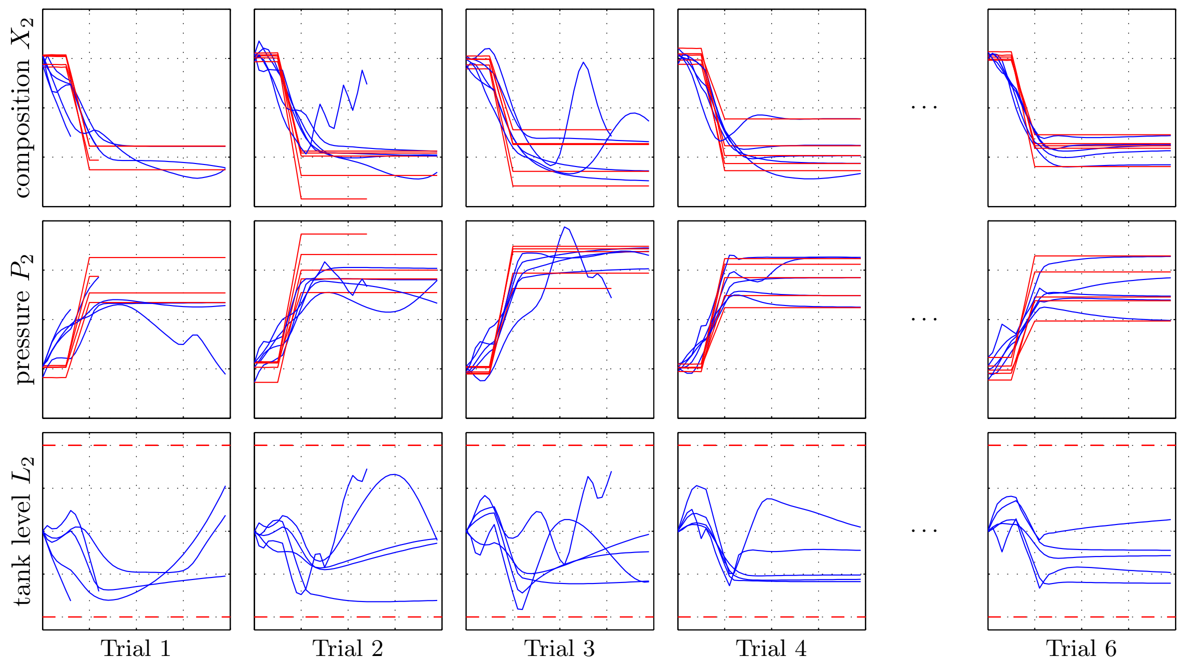
Our task will be to design a control policy to track a number of possible setpoint changes on X_2 and P_2. The setpoint change for X_2 will be from $\mathcal{N}(25, 0.5^2)$ to $\mathcal{N}(15, 2^2)$ each sampled independently. Similarly the change on P_2 will be from $\mathcal{N}(50, 1^2)$ to $\mathcal{N}(70, 5^2)$. The change will take place linearly over a period of 5 mins. These setpoint changes are shown in Fig. 6.7. The reference state space model we use to achieve this, for X_2 for example, is

$$\mathbf{x}^{\mathbf{r}}_{k+1} = \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ 0 & \begin{bmatrix} \mathbf{0} & 1 \end{bmatrix} \end{bmatrix}}_{\mathbf{A}^{\mathbf{r}}} \mathbf{x}^{\mathbf{r}}_k \quad \text{and} \quad X^r_{2,k} = \underbrace{\begin{bmatrix} 1 & \mathbf{0} \end{bmatrix}}_{\mathbf{C}^{\mathbf{r}}} \mathbf{x}^{\mathbf{r}}_k$$

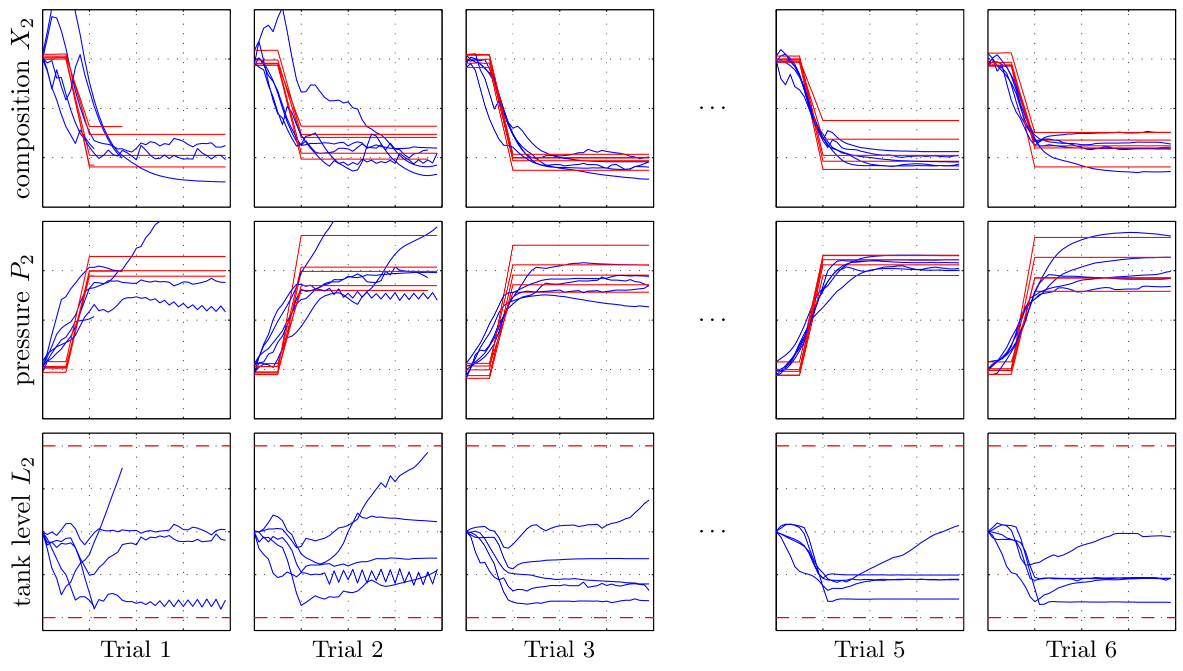
with the following choice of distribution over $\mathbf{x}^{\mathbf{r}}_0$ to obtain the 5 minute transitioning regime shown in Fig. 6.7(a)

$$\mathbf{x}^{\mathbf{r}}_0 \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\alpha} & (\mathbf{1} - \boldsymbol{\alpha}) \end{bmatrix} \begin{bmatrix} 25 \\ 15 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\alpha} & (\mathbf{1} - \boldsymbol{\alpha}) \end{bmatrix} \begin{bmatrix} 0.5^2 & 0 \\ 0 & 2^2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}^\top \\ (\mathbf{1} - \boldsymbol{\alpha})^\top \end{bmatrix} \right) \tag{6.6}$$

where $\boldsymbol{\alpha} = [1, 1, 1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2, 0]^\top \in \mathbb{R}^{11}$. A similar process is then carried out for P_2.



(a) Affine control policy



(b) Additive Gaussian-RBF control policy

Figure 6.8: Trajectories of the state variables of the evaporator over five independent runs of the learning algorithm using different policy structures. The trials are the application of the current best policy after a given simulation phase. The actual state trajectories are given in blue, the setpoints are given in red and the tank constraints are given by the red dashed lines. The x-axes depict a range of $[0, 40]$ mins while the y-axes show $[10, 30]$ %, $[40, 80]$ kPa and $[-0.2, 4.2]$ m for X_2, P_2 and L_2 respectively.

Altering the Policy Structure

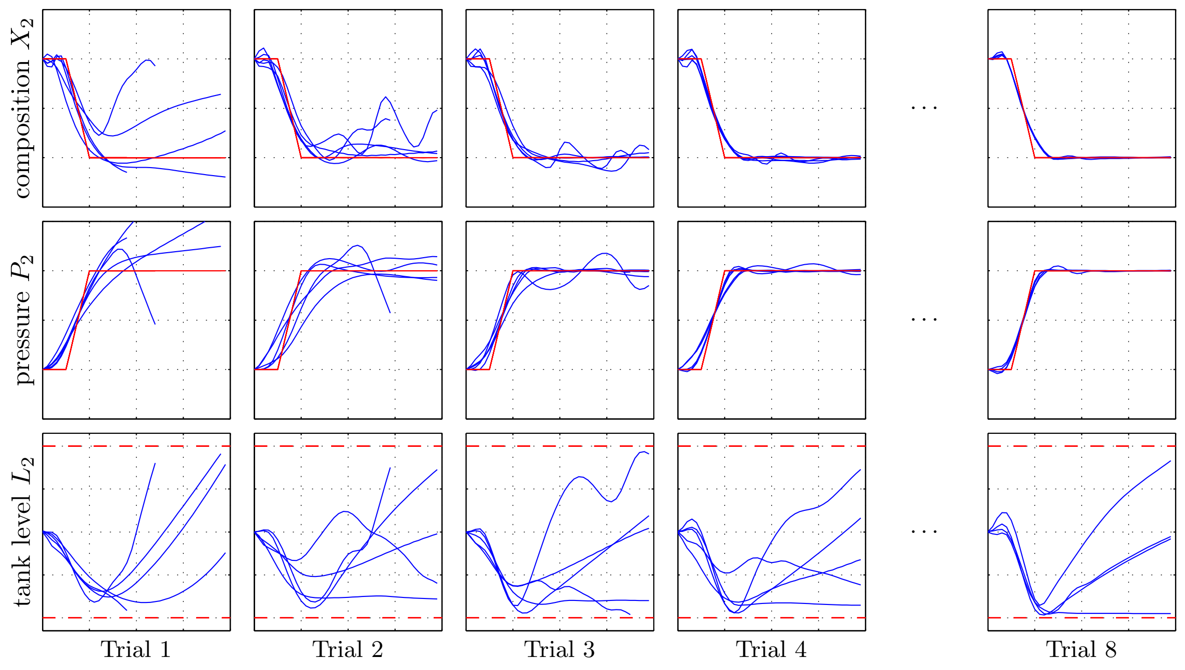
Our initial training data set was generated from a crudely designed LQR control policy for the equilibrium at $X_2 = 25\,\%$ and $P_2 = 50.5\,\text{kPa}$. The training data set consisted of two 20 minute trajectories showing this control policy regulating the system back to its setpoint from an initial perturbation of $X_2 \sim \mathcal{N}(25, 2^2)$ and $P_2 \sim \mathcal{N}(50, 5^2)$. The simulation stage of the algorithm then aimed to minimise a cost function over a horizon of 40 minutes and using the stage cost outlined in the previous section.

The first set of experiments we ran was on a system with servomechanism time constants of $\tau_u = 0.1\,\text{mins}$. In this case we ignore the additional states introduced by the lags and use only the normal system states and control actions for modelling. We compare the results of using an affine feedback policy and an additive Gaussian-RBF policy with a grid of 30 basis functions in the region of operation. The affine policy therefore has a total of 24 free parameters. For the Gaussian-RBF we fix the centres of each basis function but leave the length scales, output scales and weights to be tuned automatically, a total of 132 free parameters. Each policy has access to the integral of the error between the setpoints on X_2 and P_2 and their actual location, therefore are able to achieve integral action.

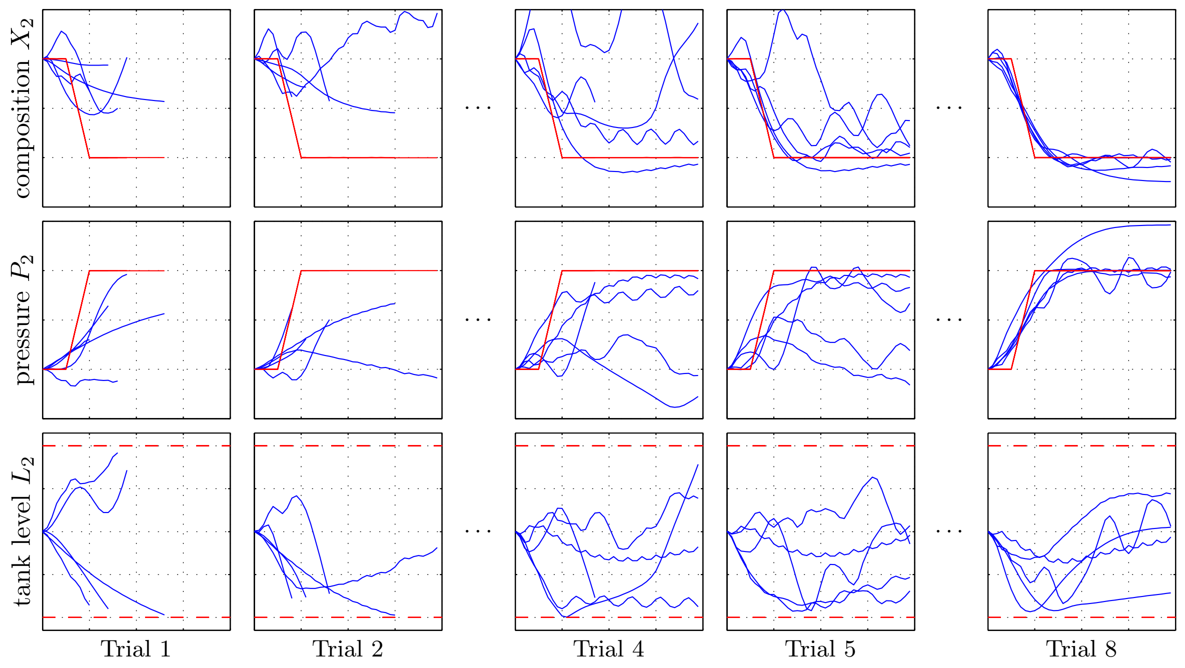
The results of the simulation we have just described are given in Fig. 6.8. Let us first discuss the performance of the learned affine policy shown in Fig. 6.8(a). We can observe in the trial of the first learned policy that some of the runs have found a solution that is stabilising but not yet able to track while others have completely failed and have to be stopped before they drain the separator tank. By the second trial most policies have stabilised the system but it is only by the fourth trial that offset-free tracking is achieved. This has taken a total of 2 hours of interaction (plus the initial training data set) to achieve this performance.

We now turn our attention to the performance of the additive RBF policy given in Fig. 6.8(b). We can see that by the third iteration it has stabilised all the runs, in contrast to the affine policy which took until the fourth. However, it actually takes the policy until the sixth trial to exploit the integral action capability required to achieve offset free tracking as can be clearly observed when comparing the results at the fifth and sixth trials. The reason for this slow learning is most likely the fact that we restrict the policy optimisation to only 50 function evaluations. The decrease in cost when moving from a simply stabilising to a tracking policy is relatively small but the change in policy parameters may be large, therefore the 50 evaluations was not enough to actually reach the optimal solution after a given trial.

We further note that it may appear from the plots in Fig. 6.8 that the policy has access to some preview horizon of the setpoint change as it appears to anticipate the change. This is actually not the case and is simply because neither policy has learned, or is able, to apply offset free tracking at both setpoints.



(a) Assumed servomechanism lag of 0.8 mins



(b) Assumed servomechanism lag of 0.4 mins

Figure 6.9: Trajectories of the state variables of the evaporator over five independent runs of the learning algorithm where wrong approximations of the real servomechanism lag of 1.2 mins are forced upon the internal model of the dynamics. The trials are the application of the current best policy after a given simulation phase. The actual state trajectories are given in blue, the setpoints are given in red and the tank constraints are given by the red dashed lines. The x-axes depict a range of $[0, 40]$ mins while the y-axes show $[10, 30]$ %, $[40, 80]$ kPa and $[-0.2, 4.2]$ m for X_2, P_2 and L_2 respectively.

Incorporating Erroneous Prior Information

The second set of experiments we ran involved learning to track a deterministic setpoint change, given by the means in Fig. 6.7. The use of a deterministic change is simply to make our plots clearer. For these experiments we use the full nonlinear system of equations with control actions applied through servomechanisms with lags of $\tau_u = 1.2$ mins, as used by Maciejowski (2002). The problem we investigate is the effect of incorporating erroneous prior information into the model of the system, in this case: a model for the servomechanisms with an underestimated value of τ_u.

The results for a wrongly modelled lag of $\tau_u = 0.8$ mins (a mismatch of 33 %) are shown in Fig. 6.9(a) and the results for $\tau_u = 0.4$ mins (a mismatch of 66 %) are in Fig. 6.9(b). It is clear from Fig. 6.9(a) that a discrepancy of 33 % causes little disruption to the learning process aside from some unwanted oscillatory behaviour exhibited in the second and third trials. We note that an important issue with finite horizon trajectory optimisation is shown in trial eight. We see that one of the runs is rapidly approaching the upper constraint boundary and therefore incurs no penalty over the horizon. However, this is undesirable since obviously if we run it for any longer the constraint will be violated. This issue could be addressed by adding an additional penalty on L_2 penalising deviations from the 2 m mark. Also, considering distributions over possible setpoint changes tends to deal with this issue.

We now turn our attention to a discrepancy of 66 % as in Fig. 6.9(b). We can see that it has a strongly adverse effect on learning. Even so, by the eighth trial we can observe that three of the runs have managed to achieve the task with the other two getting close. We further note that the learning process exhibits a useful feature. By trial four the algorithm has learned to satisfy the system constraints, which obviously is a more important criteria than the setpoint tracking itself. Once it has achieved this it starts to make improvements on the tracking performance as shown in trial five.

6.5 Summary

We have considered in this chapter various issues associated with control policy design from a classical perspective and from the learning control perspective that has been the subject of most of this thesis. The comparison was carried out in the context of two case studies: the unicycle and the evaporator. One of the main features of the classical control design that we drew out from the case studies was that often we have to deal with achievement of the task, satisfying system constraints and dealing with modelling mismatch all together with the cost function parameters. In other words, the solution to all these issues are coupled into the same design parameters. Methods such as MPC can separate out dealing with constraints but still require model mismatch and task achievement to be dealt with by the same tuning parameters. Conversely, in the learning algorithm, there are separate tuneable parameters associated with each problem and therefore tuning can take place in an arguably more intuitive manner.

Addressing the unicycle balancing problem specifically, we saw that it is actually a difficult problem just to obtain a linearised model appropriate for control design. This process actually requires significant knowledge of the underlying system dynamics. The learning algorithm can bypass this issue by using a learned nonlinear representation with little required expert knowledge. However, when comparing the learned control policy and an LQR control policy we find that the learned solution to the problem, although satisfying the cost criterion, will not be a good control policy in practice. This is because it converges on an orbital trajectory which will fall over once the time of the prediction horizon is over.

Finally, the evaporator control problem threw up some interesting issues. The first was the question of how to actually choose an appropriate state space representation of the system, which is necessary for both the classical and the learning methods. This process requires at least some high level intuition about the problem. We went on to show that the learning control method could be applied in a safe and intuitive manner to the problem with minimal tuning. In particular, we were able to train an additive Gaussian-RBF control policy to obtain greater generality than a standard affine structure but avoid the problem of choosing basis function locations common to standard RBFs. We also demonstrated that in this context the learning approach can handle significant modelling errors forced upon the dynamics prior.

CHAPTER 7

Conclusions and Future Work

7.1 Conclusions

The central objective of this thesis was to provide methods to allow the incorporation of useful prior knowledge when designing a control policy for a system governed by unknown, or partially unknown, dynamics. The specific context which we considered was a probabilistic learning framework in which modelling uncertainty is interpreted as a posterior distribution over a space of dynamics functions. We were able to deal with distributions over spaces of functions using Gaussian processes. In addition to the central objective, we investigated the effect incorporating such information could have on learning and how our resulting learning algorithm compared with a standard control theoretic approach based on two case study examples. In these concluding remarks we shall summarise how these objectives have been achieved.

Prior knowledge about the dynamics of a given system often comes in the form of known, or approximate, relationships between a subset of the state variables. A common example of this is position-velocity dependencies. We outlined a novel and elegant method in which these known relationships could be incorporated into the probabilistic framework while uncertainty was still treated in a rigorous manner. To achieve this we considered how the predictions from multiple dynamics models could be combined to give an overall prediction of a state trajectory when a certain control policy was acting on the system. Experimental results on the pendulum swing-up task showed that including an approximate model for position-velocity relationships could help to avoid problems of finding local minima with some settings, in particular position reconstruction, but seemed to aggravate the problem with other settings, in particular velocity reconstruction. This was a surprising result and prompted us to apply caution before forcing our own assumptions onto a problem even if they seem sensible. Further results were obtained when we applied our algorithm to the problem of learning a control policy to balance a simulated robotic unicycle. Again, this pointed to the need for caution when forcing our beliefs into a model. One of the main advantages we found however was in terms of computational saving in learning where we get a learning speed up linearly proportional to the number of known state relationships we include.

The framework developed for multiple dynamics models, or augmented dynamics, we found could also be applied to the problem of learning a control policy to track a known reference signal or distribution over possible reference signals. Further, if the control policy had access to a preview horizon of the reference, this information could also be encoded. The capability to perform reference tracking was illustrated on a simulated inverted pendulum attached to a moving cart tracking a moving positional setpoint. We then demonstrated it on the simulated unicycle and showed that it was capable of learning how to track a complex spinning manoeuvre.

The second problem we tackled in order to address the central objective was how we could encode the assumption that a discrete-time system was actually a sampled continuous-time system. In particular, how could we exploit any underlying structure in the continuous-time dynamics for prediction in discrete-time given only discrete data. We defined a set of Gaussian process priors over discrete-time dynamics based on a prior over the underlying continuous-time dynamics. The relationship exploited the approximate relationship between continuous and discrete-time systems given by the Runge-Kutta family of numerical integration methods. We demonstrated, on data sets obtained from the simulated pendulum, that our method can pick out structure in the continuous dynamics such as invariance, linear, additive and general nonlinear relationships between states. The resulting posterior distribution can exploit this information for improved predictive performance. A further useful feature of these priors are that we can combine non-uniformly sampled discrete data with continuous-time data, including local linear models.

In our final investigation, we took two case study examples in order to compare how the application of our learning algorithm compared with a standard control theoretic approach. This comparison was in largely in terms of discussing what expert knowledge was required of the user in order to implement either method. In the case of the robotic unicycle we found that significant knowledge of the system was required of the user to even find an appropriate linear model on which to perform design. We noted that performance criteria: constraint satisfaction and dealing with modelling uncertainty often need to be addressed using the same set of design parameters in a standard control approach. When then compared the actual performance of a learned control policy and a policy derived using LQR. We found that, although the learned policy performed well in terms of the cost function over the finite prediction horizon, instead of balancing it spun the unicycle around in a tight orbit of the origin, which would eventually collapse after the prediction horizon had elapsed. This was also true of the evaporator study.

Conversely, the learning approach provides quite a clear structure to separate out these issues leaving some relatively intuitive tuning parameters. One of the issues with applying learning on the evaporator was how to ensure that learning proceeded in a safe manner, this was addressed under the assumption that there was an existing control policy in place, even if that was simply a human operator. In addition we demonstrated the use of an additive RBF policy and showed that the learning approach was able to handle poor modelling assumptions in the form of wrongly estimated servomechanism lags.

7.2 Future Work

There are a number of avenues available to extend the work of this thesis. We begin with the framework of using multiple dynamics models as a way of incorporating prior knowledge into the learning algorithm. There is much scope for further investigation into the effects of incorporating approximate relationships between state variables. This has been touched upon in the setting of position-velocity relationships and lags on the control actions but looking at more general relationships could be fruitful. The overall aim would be to somehow determine when it would be beneficial to force such prior knowledge into the model and when it would be best to let the learning algorithm infer the information for itself.

We now turn to the class of priors over sampled continuous-time systems defined in Chapter 5. The benefit of using such a prior has been clearly demonstrated in terms of model likelihood and predictive performance when there is an underlying continuous-time structure in the system to be exploited. The next stage is to derive the predictive equations when the input is uncertain. We could therefore incorporate these priors into the full learning control algorithm and investigate what the benefits are in terms of learning control policies. The biggest current bottleneck to achieving this is the computational complexity of evaluating and training these priors. This would have to be reduced significantly in order for these priors to be of practical use in the context of learning control.

Finally, on a broader scale, we hope that the work of this thesis will be a step towards a "recipe book" for different types of control problems. Specifically, we would like to determine which control or learning strategies would be most efficacious or suitable for a given control problem. The issue of what forms of prior knowledge would be most appropriate to incorporate into a given problem, as discussed above, would also play a significant role in this classification procedure. This work would help pull together the rich resources available from the fields of Machine Learning and Control in a way that could be useful for the practitioner.

APPENDIX A Mathematical Background

A.1 Gaussian Identities

This section follows closely Appendix A.2 in Rasmussen & Williams (2006). The multivariate Gaussian distribution over the random vector space $\mathbf{x} \in \mathbb{R}^D$ has a probability density given by

$$\mathcal{N}\big(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}\big) = (2\pi)^{-D/2}|\boldsymbol{\Sigma}|^{-1/2} \exp\big(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\big) \tag{A.1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$ is the mean and $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$ is the covariance. Let $\mathbf{x}$ and $\mathbf{y}$ be jointly Gaussian random vectors

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_\mathbf{x}\\ \boldsymbol{\mu}_\mathbf{y}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_\mathbf{x} & \boldsymbol{\Sigma}_\mathbf{xy}\\ \boldsymbol{\Sigma}_\mathbf{xy}^\top & \boldsymbol{\Sigma}_\mathbf{y}\end{bmatrix}\right) \tag{A.2}$$

then the *marginal* distribution of $\mathbf{x}$ and the *conditional* distribution of $\mathbf{x}$ given $\mathbf{y}$ are

$$\mathbf{x} \sim \mathcal{N}\big(\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\Sigma}_\mathbf{x}\big) \tag{A.3}$$

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}\big(\boldsymbol{\mu}_\mathbf{x} + \boldsymbol{\Sigma}_\mathbf{xy}\boldsymbol{\Sigma}_\mathbf{y}^{-1}(\mathbf{y}-\boldsymbol{\mu}_\mathbf{y}), \boldsymbol{\Sigma}_\mathbf{x} - \boldsymbol{\Sigma}_\mathbf{xy}\boldsymbol{\Sigma}_\mathbf{y}^{-1}\boldsymbol{\Sigma}_\mathbf{xy}^\top\big) \tag{A.4}$$

The product of two Gaussians gives another (unnormalised) Gaussian

$$\mathcal{N}(\mathbf{x}|\mathbf{a}, \mathbf{A})\mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{B}) = c^{-1}\mathcal{N}\big(\mathbf{x}|\mathbf{c}, \mathbf{C}\big) \tag{A.5}$$

where the normalising constant is itself a Gaussian $c^{-1} = \mathcal{N}(\mathbf{a}|\mathbf{b}, \mathbf{A}+\mathbf{B})$ and the mean and covariance are given by

$$\begin{aligned}
\mathbf{c} &= (\mathbf{A}^{-1}+\mathbf{B}^{-1})^{-1}(\mathbf{A}^{-1}\mathbf{a}+\mathbf{B}^{-1}\mathbf{b}) &&= \mathbf{B}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{a}+\mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{b} \\
\mathbf{C} &= (\mathbf{A}^{-1}+\mathbf{B}^{-1})^{-1} &&= \mathbf{B}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{A}
\end{aligned}$$

To generate samples from a multivariate Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ using a scalar Gaussian generator, first compute the Cholesky decomposition $\mathbf{L}$ of the covariance matrix $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$ where $\mathbf{L}$ is lower triangular. Next, generate a random vector $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ using independent scalar samples. Compute $\mathbf{x} = \boldsymbol{\mu} + \mathbf{L}\mathbf{u}$ which has the desired distribution with expectation $\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} + \mathbf{L}\mathbb{E}[\mathbf{u}] = \boldsymbol{\mu}$ and covariance $\text{cov}[\mathbf{x}] = \mathbf{L}\text{cov}[\mathbf{u}]\mathbf{L}^\top = \mathbf{L}\mathbf{L}^\top = \boldsymbol{\Sigma}$.

A.2 Kullback-Leibler Divergence

This section is taken from Appendix A.5 in Rasmussen & Williams (2006). The Kullback-Leibler (KL) divergence $\text{KL}(p||q)$ of some distribution $p(\mathbf{x})$ with respect to $q(\mathbf{x})$ is defined as

$$\text{KL}(p||q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \mathrm{d}\mathbf{x} \tag{A.6}$$

The KL divergence of a distribution $p(\mathbf{x})$ on $\mathbb{R}^D$ with respect to some Gaussian distribution $q(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is

$$\text{KL}(p||q) = \int \Big(\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) + \log\big(p(\mathbf{x})\big)\Big)p(\mathbf{x})\mathrm{d}\mathbf{x} + \tfrac{1}{2}\log|\boldsymbol{\Sigma}| + \tfrac{D}{2}\log 2\pi \tag{A.7}$$

This can be minimised with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ by differentiating with respect to these parameters and setting the resulting expression to zero. It turns out the the optimal q is the one with first moment $\int \mathbf{x}p(\mathbf{x})\mathrm{d}\mathbf{x}$ and second moment $\int \mathbf{x}\mathbf{x}^\top p(\mathbf{x})\mathrm{d}\mathbf{x}$ i.e. the moment matched solution.

A.3 Matrix Calculus

The convention for dealing with derivatives of matrices with respect to matrices is taken from Brookes (2005) which makes use of the vectorisation operation to ensure that all algebra remains in terms of standard matrix operations. Consider the matrices $\mathbf{X} \in \mathbb{R}^{m\times n}$ and $\mathbf{Y} \in \mathbb{R}^{p\times q}$ then the derivative of $\mathbf{Y}$ with respect to $\mathbf{X}$ could be viewed in terms of a fourth-order tensor $\partial\mathbf{Y}/\partial\mathbf{X} \in \mathbb{R}^{p\times q\times m\times n}$. However, these are awkward to deal with computationally so define the derivative as follows

$$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} := \frac{\partial \text{vec}(\mathbf{Y})}{\partial \text{vec}(\mathbf{X})} = \begin{bmatrix} \frac{\partial \mathbf{Y}[:,1]}{\partial \mathbf{X}[:,1]} & \cdots & \frac{\partial \mathbf{Y}[:,1]}{\partial \mathbf{X}[:,n]} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{Y}[:,q]}{\partial \mathbf{X}[:,1]} & \cdots & \frac{\partial \mathbf{Y}[:,q]}{\partial \mathbf{X}[:,n]} \end{bmatrix} \in \mathbb{R}^{pq\times mn} \tag{A.8}$$

consisting of blocks $\partial \mathbf{Y}[:, i]/\partial \mathbf{X}[:, j] \in \mathbb{R}^{p\times m}$. This convention will also hold for vectors and scalars (for which the vec operation has no effect). For example, for scalars y and x and vectors $\mathbf{y} \in \mathbb{R}^p$ and $\mathbf{x} \in \mathbb{R}^m$ the dimensions of the associated derivatives are as follows with elements determined by the definition in Eq. (A.8)

$$\begin{array}{lll}
\frac{\partial y}{\partial x} \in \mathbb{R} & \frac{\partial y}{\partial \mathbf{x}} \in \mathbb{R}^{1\times m} & \frac{\partial y}{\partial \mathbf{X}} \in \mathbb{R}^{1\times mn} \\
\frac{\partial \mathbf{y}}{\partial x} \in \mathbb{R}^{p} & \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{p\times m} & \frac{\partial \mathbf{y}}{\partial \mathbf{X}} \in \mathbb{R}^{p\times mn} \\
\frac{\partial \mathbf{Y}}{\partial x} \in \mathbb{R}^{pq} & \frac{\partial \mathbf{Y}}{\partial \mathbf{x}} \in \mathbb{R}^{pq\times m} & \frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \in \mathbb{R}^{pq\times mn}
\end{array}$$

Now consider a third matrix $\mathbf{Z} \in \mathbb{R}^{q\times r}$ which is dependent on $\mathbf{X}$ through $\mathbf{Y}$. The chain rule and product rule can be written as

$$\textit{Chain Rule:} \qquad \frac{\partial \mathbf{Z}}{\partial \mathbf{X}} = \frac{\partial \mathbf{Z}}{\partial \mathbf{Y}}\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \tag{A.9}$$

$$\textit{Product Rule:} \qquad \frac{\partial (\mathbf{YZ})}{\partial \mathbf{X}} = (\mathbf{I}\otimes \mathbf{Y})\frac{\partial \mathbf{Z}}{\partial \mathbf{X}} + (\mathbf{Z}^\top \otimes \mathbf{I})\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \tag{A.10}$$

Some further useful results where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are matrices independent of $\mathbf{X}$ and of appropriate dimensions are

$$\frac{\partial}{\partial \mathbf{X}}\big(\mathbf{X}^\top\big) = \mathbf{T}_{m,n} \tag{A.11}$$

$$\frac{\partial}{\partial \mathbf{X}}\big(\mathbf{A}^\top\mathbf{X}\mathbf{B}\big) = (\mathbf{B}\otimes\mathbf{A})^\top \tag{A.12}$$

$$\frac{\partial}{\partial \mathbf{X}}\big(\mathbf{X}^\top\mathbf{C}\mathbf{X}\big) = (\mathbf{I}\otimes\mathbf{X}^\top\mathbf{C}) + (\mathbf{X}^\top\mathbf{C}^\top\otimes\mathbf{I})\mathbf{T}_{m,n} \tag{A.13}$$

The matrix $\mathbf{T}_{m,n} \in \mathbb{R}^{mn\times mn}$ is the *vectorised transpose matrix* whose $(i, j)^{\text{th}}$ element is equal to one if $j = 1 + m(i-1) - (mn-1)\lfloor (i-1)/n\rfloor$ and zero otherwise.

APPENDIX B

Equations of Motion

B.1 Pendulum

The pendulum system used in this thesis is shown in Fig. B.1, along with the forces and moments that act on it. Resolving the rate of change of angular momentum around the pivot point gives

$$\begin{aligned} I\ddot{\theta} &= u - T_\mathrm{f} - \tfrac{1}{2}l(W\sin\theta) \\ \tfrac{1}{3}ml^2\ddot{\theta} &= u - b\dot{\theta} - \tfrac{1}{2}mlg\sin\theta \end{aligned} \tag{B.1}$$

where we consider the pole to be of uniform density. The variables I, T_f and W refer to the moment of inertia (about the pivot point), friction torque and weight respectively. The constants we used were mass $m = 1\,\mathrm{kg}$, length $l = 1\,\mathrm{m}$, friction $b = 0.1\,\mathrm{N\,s^{-1}}$ and gravitational acceleration $g = 9.81\,\mathrm{m\,s^{-2}}$. This system then has states $\mathbf{x} = [\theta; \dot{\theta}]^\top$ and we constrain the input torque such that $u \in [-3, 3]\,\mathrm{N\,m}$.

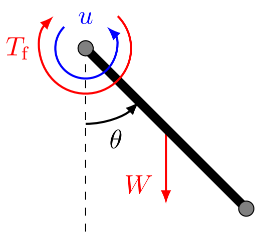
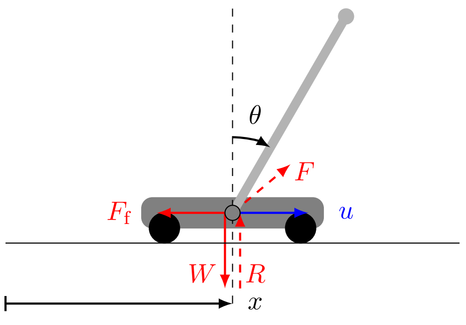
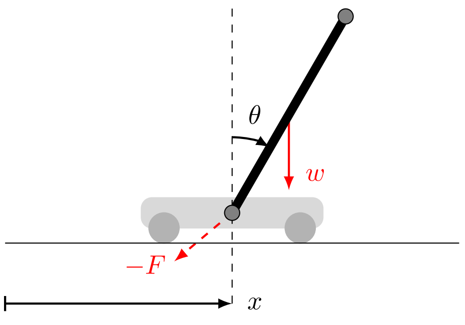


Figure B.1: Torque-limited pendulum with angle from the down position θ, input torque u, weight W and friction torque T_f.



(a) Cart free body diagram



(b) Pole free body diagram

Figure B.2: Cart-pole free body diagrams with position x, pole angle θ, action force u, weight of the pole w, weight of the cart W, friction force F_f, internal force F and reaction force R.

B.2 Cart-Pole

The cart-pole system used in this thesis is shown by the free body diagrams in Fig. B.2, along with the forces and moments that act upon it. The following derivation is based on that of Florian (2007). We shall begin by considering the free body diagram of the cart alone, shown in Fig. B.2(b), and balancing the forces and accelerations in the lateral direction. Doing this we get

$$M\ddot{x} = u - F_\mathrm{f} + F_x \tag{B.2}$$

where x is the lateral position of the cart, θ is the angle of the pole, M is the mass of the cart, F_x is the lateral component of the internal force between cart and pole, u is the driving force and F_f is the friction force. Now we turn to the free body diagram of the pole, shown in Fig. B.2(a), and balance the forces and accelerations along the lateral and vertical directions. This yields

$$m\big(\ddot{x} + \tfrac{1}{2}l\ddot{\theta}\cos\theta - \tfrac{1}{2}l\dot{\theta}^2\sin\theta\big) = -F_x \tag{B.3}$$
$$m\big(\tfrac{1}{2}l\ddot{\theta}\sin\theta + \tfrac{1}{2}l\dot{\theta}^2\cos\theta\big) = mg + F_y \tag{B.4}$$

respectively, where m is the mass of the pole and l is its length. The $\frac{1}{2}ml\dot{\theta}^2$ and $\frac{1}{2}ml\ddot{\theta}$ terms are the accelerations due to the angular acceleration of the pole, or the “fictitious” forces. Finally, if we balance the torques and angular accelarations around the joint we get

$$I\ddot{\theta} + \tfrac{1}{2}m\ddot{x}l\cos\theta = \tfrac{1}{2}mgl\sin\theta \tag{B.5}$$

with moment of intertia $I = \frac{1}{3}ml^2$ for a pole of uniform density. We can now eliminate the components of the internal force, F_x and F_y, to find the two equations of motion. Substitute Eq. (B.2) into Eq. (B.3) and rearrange Eq. (B.5) to yield the relationships

$$(M+m)\ddot{x} = \tfrac{1}{2}ml(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta) + u - b\dot{x} \tag{B.6}$$
$$l\ddot{\theta} = \tfrac{3}{2}g\sin\theta - \tfrac{3}{2}\ddot{x}\cos\theta \tag{B.7}$$

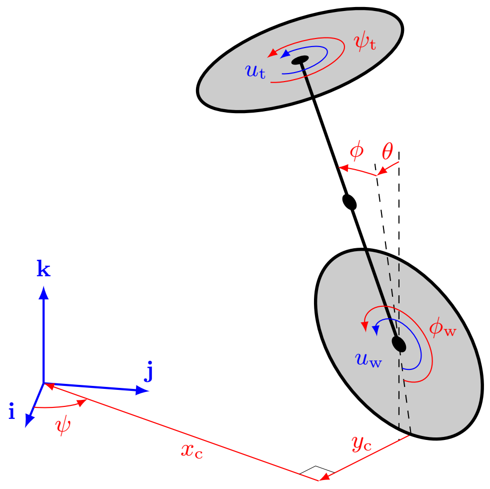


Figure B.3: The robotic unicycle with state variables: pitch angle ϕ, roll angle θ, yaw angle ψ, wheel angle ϕ_w, turntable angle ψ_t, the associated angular velocities and the location of the global origin (x_c, y_c). The actions are the wheel motor torque u_w and the turntable motor torque u_t. The global coordinate system is defined by the vectors $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$.

where we employ a linear friction model $F_f = b\dot{x}$. Eqs. (B.6)–(B.7) can then be rearranged to isolate the terms $\ddot{x}$ and $\ddot{\theta}$ and give the equations of motion for this system. The action force was constrained to $u \in [-10, 10]$ N and the physical constants we used were: length $l = 1$ m, mass of cart $M = 0.5$ kg, mass of pole $m = 0.5$ kg and coefficient of friction $b = 0.1\,\mathrm{N\,s\,m^{-1}}$.

B.3 Unicycle

B.3.1 Method

The robotic unicycle is shown in Fig. B.3 with global coordinate system defined by the orthonormal vectors $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$. The spatial position of the unicycle is fully defined by the pitch angle ϕ, roll angle θ, yaw angle ψ, wheel angle ϕ_w, turntable angle ψ_t and location of the global origin with respect to the body-centred coordinate system (x_c, y_c). We chose the state vector to be $\mathbf{x} = [\phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}, \dot{\phi}_w, \dot{\psi}_t, x_c, y_c]^\top \in \mathbb{R}^{10}$ where we exclude ϕ_w and ψ_t since they clearly have no effect on the dynamics. The action vector $\mathbf{u}$ is made up of a wheel motor torque u_w and a turntable motor torque u_t. The equations of motion that govern the unicycle were derived by Forster (2009). We shall provide a sketch of the full derivation here, in which we follow the steps taken by Forster in Section 3.3 of his thesis.

Let us start with the coordinates (x_c, y_c). These are centred on the point of contact with the floor and define the location of the global origin. The coordinate x_c lies parallel to the current direction of travel and y_c is orthogonal to it. These coordinates evolve according to

$$\dot{x}_c = r_w \dot{\phi}_w \cos\psi \tag{B.8}$$

$$\dot{y}_c = r_w \dot{\phi}_w \sin\psi \tag{B.9}$$

Constant	Description	Value	Units
m_w	Wheel mass	1.0	kg
r_w	Wheel radius	0.225	m
A_w	Moment of inertia of wheel around $\mathbf{i}_\mathrm{w}$	0.0242	$\mathrm{kg\,m^2}$
C_w	Moment of inertia of wheel around $\mathbf{k}_\mathrm{w}$	0.0484	$\mathrm{kg\,m^2}$
m_f	Frame mass	23.5	kg
r_f	Frame centre of mass to wheel	0.54	m
A_f	Moment of inertia of frame around $\mathbf{i}_\mathrm{f}$	0.4248	$\mathrm{kg\,m^2}$
B_f	Moment of inertia of frame around $\mathbf{j}_\mathrm{f}$	0.4608	$\mathrm{kg\,m^2}$
C_f	Moment of inertia of frame around $\mathbf{k}_\mathrm{f}$	0.8292	$\mathrm{kg\,m^2}$
m_t	Turntable mass	10.0	kg
r_t	Frame centre of mass to turntable	0.27	m
A_t	Moment of inertia of turntable around $\mathbf{i}_\mathrm{t}$	1.3	$\mathrm{kg\,m^2}$
C_t	Moment of inertia of turntable around $\mathbf{k}_\mathrm{t}$	0.2	$\mathrm{kg\,m^2}$
g	Gravitational acceleration	9.81	$\mathrm{m\,s^{-2}}$

Table B.1: Physical constants used for the simulated robotic unicycle. The coordinate systems defined by the $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ vectors are shown in Fig. B.4.

where r_w is the wheel radius. The full unicycle model was obtained by analysing the wheel, frame and turntable as individual Free Body Diagrams (FBDs), as depicted in Fig. B.4. Linear momentum and moment of momentum for each FBD were then resolved to yield six scalar equations for each free body. The internal forces were then eliminated to yield five independent scalar equations which govern the evolution of the angular states. A description of the physical constants of the system along with the values we use in this thesis are given in Table B.1.

B.3.2 Wheel FBD

The wheel coordinate system is defined by the orthonormal vectors $\mathbf{i}_\mathrm{w}, \mathbf{j}_\mathrm{w}$ and $\mathbf{k}_\mathrm{w}$ as shown in Fig. B.4(a). We begin by noting that the angular velocity of the wheel coordinate system is $\Omega_\mathrm{w} = \dot{\psi}\mathbf{k} + \dot{\theta}\mathbf{j}_\mathrm{w}$. Now noting that the angular velocity of the wheel only differs in the $\mathbf{k}_\mathrm{w}$ direction and assuming no slip between wheel and floor we have expressions for the velocity and angular velocity of the wheel

$$\mathbf{v}_\mathrm{w} = -(\boldsymbol{\omega}_\mathrm{w} \times r_\mathrm{w}\mathbf{i}_\mathrm{w})$$
$$\boldsymbol{\omega}_\mathrm{w} = \Omega_\mathrm{w} + \dot{\phi}_\mathrm{w}\mathbf{k}_\mathrm{w}$$

From these expressions we can derive the acceleration of the wheel $\dot{\mathbf{v}}_{\mathrm{w}}$ and the rate of change of angular momentum $\dot{\mathbf{h}}_{\mathrm{w}}$ as

$$\begin{aligned}
\dot{\mathbf{v}}_{\mathrm{w}} &= \frac{\partial \mathbf{v}_{\mathrm{w}}}{\partial t} + (\Omega_{\mathrm{w}} \times \mathbf{v}_{\mathrm{w}}) \\
\dot{\mathbf{h}}_{\mathrm{w}} &= A_{\mathrm{w}}\frac{\partial \omega_{\mathrm{w}}[1]}{\partial t}\mathbf{i}_{\mathrm{w}} + A_{\mathrm{w}}\frac{\partial \omega_{\mathrm{w}}[2]}{\partial t}\mathbf{j}_{\mathrm{w}} + C_{\mathrm{w}}\frac{\partial \omega_{\mathrm{w}}[3]}{\partial t}\mathbf{k}_{\mathrm{w}} + (\Omega_{\mathrm{w}} \times \mathbf{h}_{\mathrm{w}})
\end{aligned}$$

where angular momentum in the wheel frame of reference is $\mathbf{h}_{\mathrm{w}} = [A_{\mathrm{w}}; A_{\mathrm{w}}; C_{\mathrm{w}}] \circ \boldsymbol{\omega}_{\mathrm{w}}$. Now we consider the forces acting on the wheel free body. These are given by the unknown quantities: axle force F_{w}, axle torque Q_{w} & reaction force R and the known quantities: wheel weight W_{w} & friction torque T. These forces and moments are shown in the right-hand plot of Fig. B.4(a). Note that we actually know the component of the axle torque Q_{w} in the $\mathbf{k}_{\mathrm{w}}$ direction as it is given by the reaction of the wheel motor on the wheel itself u_{w}. Resolving the rate of change of linear momentum and the rate of change of angular momentum around the centre of mass leads to

$$m_{\mathrm{w}}\dot{\mathbf{v}}_{\mathrm{w}} = R + F_{\mathrm{w}} + W_{\mathrm{w}} \tag{B.10}$$

$$\dot{\mathbf{h}}_{\mathrm{w}} = (r_{\mathrm{w}}\mathbf{i}_{\mathrm{w}} \times R) + Q_{\mathrm{w}} + T \tag{B.11}$$

B.3.3 Frame FBD

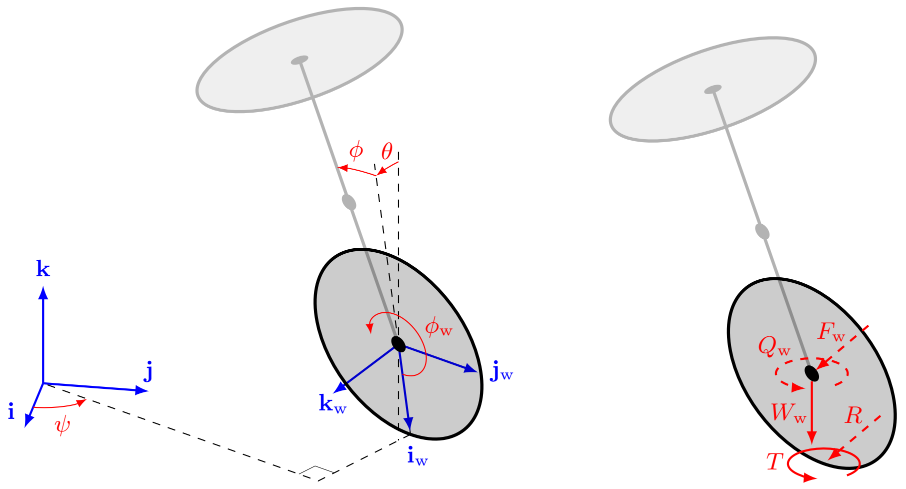
The frame coordinate system is defined by the orthonormal vectors $\mathbf{i}_{\mathrm{f}}, \mathbf{j}_{\mathrm{f}}$ and $\mathbf{k}_{\mathrm{f}} = \mathbf{k}_{\mathrm{w}}$ as shown in Fig. B.4(b). In this case, the angular velocity of the frame $\boldsymbol{\omega}_{\mathrm{f}}$ is given by the angular velocity of the wheel plus an additional spin about the wheel axis and the velocity of the frame $\mathbf{v}_{\mathrm{f}}$ is given by the velocity of the wheel plus an additional rotation about the wheel centre

$$\begin{aligned}
\mathbf{v}_{\mathrm{f}} &= \mathbf{v}_{\mathrm{w}} - (\boldsymbol{\omega}_{\mathrm{f}} \times r_{\mathrm{f}}\mathbf{i}_{\mathrm{f}}) \\
\boldsymbol{\omega}_{\mathrm{f}} &= \Omega_{\mathrm{w}} + \dot{\phi}\mathbf{k}_{\mathrm{f}}
\end{aligned}$$

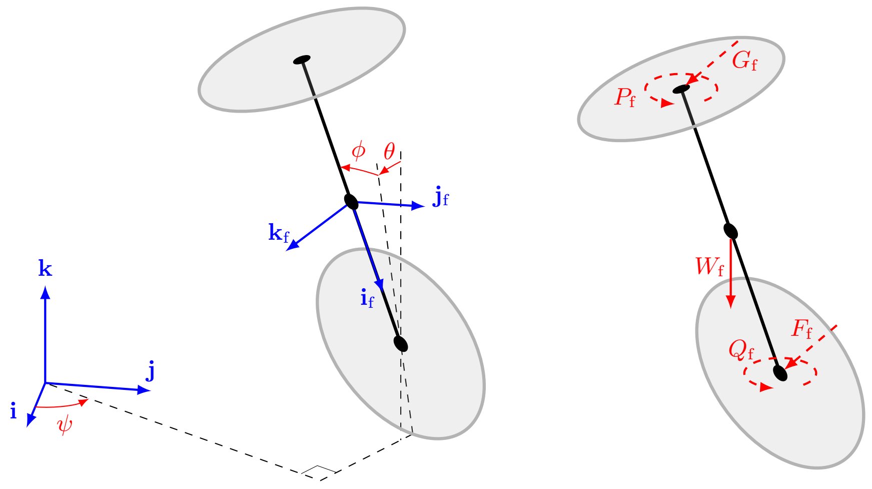
As before, we can now derive the acceleration of the frame $\dot{\mathbf{v}}_{\mathrm{f}}$ and the rate of change of angular momentum $\dot{\mathbf{h}}_{\mathrm{f}}$ as

$$\begin{aligned}
\dot{\mathbf{v}}_{\mathrm{f}} &= \frac{\partial \mathbf{v}_{\mathrm{f}}}{\partial t} + (\Omega_{\mathrm{f}} \times \mathbf{v}_{\mathrm{f}}) \\
\dot{\mathbf{h}}_{\mathrm{f}} &= A_{\mathrm{f}}\frac{\partial \omega_{\mathrm{f}}[1]}{\partial t}\mathbf{i}_{\mathrm{f}} + B_{\mathrm{f}}\frac{\partial \omega_{\mathrm{f}}[2]}{\partial t}\mathbf{j}_{\mathrm{f}} + C_{\mathrm{f}}\frac{\partial \omega_{\mathrm{f}}[3]}{\partial t}\mathbf{k}_{\mathrm{f}} + (\Omega_{\mathrm{f}} \times \mathbf{h}_{\mathrm{f}})
\end{aligned}$$

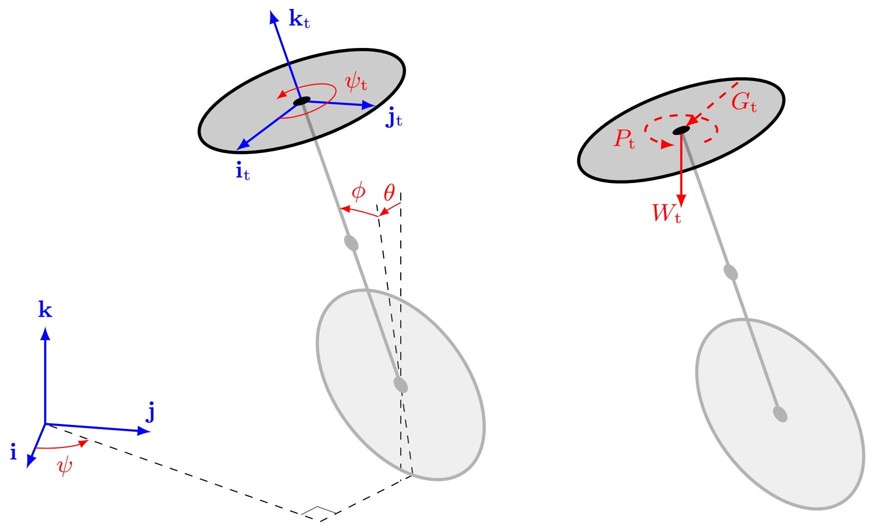
where angular momentum of the frame in this frame of reference is $\mathbf{h}_{\mathrm{f}} = [A_{\mathrm{f}}; B_{\mathrm{f}}; C_{\mathrm{f}}] \circ \boldsymbol{\omega}_{\mathrm{f}}$. The forces and moments acting on the frame are shown on the right in Fig. B.4(b). They consist of the known frame weight W_{f} and the unknown: wheel axle force $F_{\mathrm{f}} = -F_{\mathrm{w}}$, wheel axle torque $Q_{\mathrm{f}} = -Q_{\mathrm{w}}$, turntable axle force G_{f} & turntable axle torque P_{f}. But again we know that the dimension of P_{f} acting along $\mathbf{i}_{\mathrm{f}}$ is given by the reaction of the frame to the turntable motor u_{t}.



(a) The wheel as a free body diagram and coordinate system defined by $\mathbf{i}_w, \mathbf{j}_w$ and $\mathbf{k}_w$



(b) The frame as a free body diagram and coordinate system defined by $\mathbf{i}_f, \mathbf{j}_f$ and $\mathbf{k}_f$



(c) The turntable as a free body diagram and coordinate system defined by $\mathbf{i}_t, \mathbf{j}_t$ and $\mathbf{k}_t$

Figure B.4: Free body diagrams of the wheel, frame and turntable of the unicycle. The model has the angular state variables pitch ϕ, roll θ, yaw ψ, wheel angle ϕ_w and turntable angle ψ_t. The vectors $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ define the global fixed frame of reference. The centres of mass for each component are shown by the black dots. Forces and moments are shown on the right, with unknown quantities as dashed lines.

So resolving the rate of change of linear momentum and the rate of change of angular momentum around the centre of mass gives us

$$m_\mathrm{f}\dot{\mathbf{v}}_\mathrm{f} = F_\mathrm{f} + G_\mathrm{f} + W_\mathrm{f} \tag{B.12}$$

$$\dot{\mathbf{h}}_\mathrm{f} = (r_\mathrm{f}\mathbf{i}_\mathrm{f} \times F_\mathrm{f}) - (r_\mathrm{t}\mathbf{i}_\mathrm{f} \times G_\mathrm{f}) + Q_\mathrm{f} + P_\mathrm{f} \tag{B.13}$$

B.3.4 Turntable FBD

Finally, the turntable coordinate system is defined by the orthonormal vectors $\mathbf{i}_\mathrm{t} = \mathbf{k}_\mathrm{f}, \mathbf{j}_\mathrm{t} = \mathbf{j}_\mathrm{f}$ and $\mathbf{k}_\mathrm{t} = -\mathbf{i}_\mathrm{f}$ as shown in Fig. B.4(c). The velocity of the turntable centre $\mathbf{v}_\mathrm{t}$ is equal to the velocity of the wheel plus an additiona lterm caused by rotation about the wheel centre, while the angular velocity $\boldsymbol{\omega}_\mathrm{t}$ differs from $\Omega_\mathrm{t} = \Omega_\mathrm{f}$ only along $\mathbf{k}_\mathrm{t}$

$$\mathbf{v}_\mathrm{t} = \mathbf{v}_\mathrm{w} + (\Omega_\mathrm{t} \times r\mathbf{k}_\mathrm{t})$$
$$\boldsymbol{\omega}_\mathrm{t} = \Omega_\mathrm{t} + \dot{\psi}_\mathrm{t}\mathbf{k}_\mathrm{t}$$

Again, we derive the acceleration of the frame $\dot{\mathbf{v}}_\mathrm{t}$ and the rate of change of angular momentum $\dot{\mathbf{h}}_\mathrm{t}$ as

$$\dot{\mathbf{v}}_\mathrm{t} = \frac{\partial \mathbf{v}_\mathrm{f}}{\partial t} + (\Omega_\mathrm{f} \times \mathbf{v}_\mathrm{f})$$
$$\dot{\mathbf{h}}_\mathrm{t} = A_\mathrm{t}\frac{\partial \omega_\mathrm{t}[1]}{\partial t}\mathbf{i}_\mathrm{t} + A_\mathrm{t}\frac{\partial \omega_\mathrm{t}[2]}{\partial t}\mathbf{j}_\mathrm{t} + C_\mathrm{t}\frac{\partial \omega_\mathrm{t}[3]}{\partial t}\mathbf{k}_\mathrm{t} + (\Omega_\mathrm{t} \times \mathbf{h}_\mathrm{t})$$

where $\mathbf{h}_\mathrm{t} = [A_\mathrm{t}; A_\mathrm{t}; C_\mathrm{t}] \circ \boldsymbol{\omega}_\mathrm{t}$. The forces and moments acting on the turntable lead to the last of our equations

$$m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t} = G_\mathrm{t} + W_\mathrm{t} \tag{B.14}$$

$$\dot{\mathbf{h}}_\mathrm{t} = P_\mathrm{t} \tag{B.15}$$

B.3.5 Eliminating Internal Forces

We now have 18 kinematic relationships given by Eqs. (B.10)–(B.15) which govern the dynamics of the unicycle. These can be reduced to five expressions by eliminating the 13 scalar unknowns found in the unknown internal forces, F and G, the unknown reaction force R and the partially unkown torques Q and P. The first can be obtained from Eq. (B.15) and noting that the component of P_f about $\mathbf{k}_\mathrm{t}$ is the reaction to the motor torque u_t

$$\dot{\mathbf{h}}_\mathrm{t}^\top \mathbf{k}_\mathrm{t} = u_\mathrm{t} \tag{B.16}$$

The second can be obtained by first making use of the relationships $G_\mathrm{f} = -G_\mathrm{t}$ & $P_\mathrm{f} = -P_\mathrm{t}$ and then rearranging Eq. (B.12), Eq. (B.14) and Eq. (B.15) to get

$$\begin{aligned} F_\mathrm{f} &= m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t} + m_\mathrm{f}\dot{\mathbf{v}}_\mathrm{f} - W_\mathrm{t} - W_\mathrm{f} \\ G_\mathrm{f} &= W_\mathrm{t} - m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t} \\ P_\mathrm{f} &= -\dot{\mathbf{h}}_\mathrm{t} \end{aligned}$$

Substituting these into Eq. (B.13) and noting that $Q_\mathrm{w} = -Q_\mathrm{f}$ gives us

$$Q_\mathrm{w} = -\dot{\mathbf{h}}_\mathrm{f} - \dot{\mathbf{h}}_\mathrm{t} - \Big(r_\mathrm{f}\mathbf{i}_\mathrm{f} \times (W_\mathrm{f} + W_\mathrm{t} - m_\mathrm{f}\dot{\mathbf{v}}_\mathrm{f} - m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t})\Big) - \Big(r_\mathrm{t}\mathbf{i}_\mathrm{f} \times (W_\mathrm{t} - m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t})\Big)$$

Once again, the component of Q_w about the wheel axis is equal to the torque provided by the wheel motor u_w, therefore $Q_\mathrm{w}^\top \mathbf{k}_\mathrm{w} = u_\mathrm{w}$ and we have our second expression

$$-\bigg(\dot{\mathbf{h}}_\mathrm{f} + \dot{\mathbf{h}}_\mathrm{t} + \Big(r_\mathrm{f}\mathbf{i}_\mathrm{f} \times (W_\mathrm{f} + W_\mathrm{t} - m_\mathrm{f}\dot{\mathbf{v}}_\mathrm{f} - m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t})\Big) + \Big(r_\mathrm{t}\mathbf{i}_\mathrm{f} \times (W_\mathrm{t} - m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t})\Big)\bigg)^\top \mathbf{k}_\mathrm{w} = u_\mathrm{w} \quad \text{(B.17)}$$

Finally, Eq. (B.10) can be combined with our expression for $F_\mathrm{f} = -F_\mathrm{w}$ to find the reaction force at the base

$$R = m_\mathrm{w}\dot{\mathbf{v}}_\mathrm{w} + m_\mathrm{f}\dot{\mathbf{v}}_\mathrm{f} + m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t} - W_\mathrm{w} - W_\mathrm{f} - W_\mathrm{t}$$

and can be substituted into Eq. (B.11) to obtain the following three relationships

$$\begin{aligned} \dot{\mathbf{h}}_\mathrm{w} = {} & \Big(r_\mathrm{w}\mathbf{i}_\mathrm{w} \times (m_\mathrm{w}\dot{\mathbf{v}}_\mathrm{w} + m_\mathrm{f}\dot{\mathbf{v}}_\mathrm{f} + m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t} - W_\mathrm{w} - W_\mathrm{f} - W_\mathrm{t})\Big) \qquad \text{(B.18)} \\ & - \bigg(\dot{\mathbf{h}}_\mathrm{f} + \dot{\mathbf{h}}_\mathrm{t} + \Big(r_\mathrm{f}\mathbf{i}_\mathrm{f} \times (W_\mathrm{f} + W_\mathrm{t} - m_\mathrm{f}\dot{\mathbf{v}}_\mathrm{f} - m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t})\Big) + \Big(r_\mathrm{t}\mathbf{i}_\mathrm{f} \times (W_\mathrm{t} - m_\mathrm{t}\dot{\mathbf{v}}_\mathrm{t})\Big)\bigg) + T \end{aligned}$$

The five expressions contained in Eqs. (B.16)–(B.18) form the foundation for the model we use. The only unknowns remaining in these relationships are the states: $\phi, \dot{\phi}, \theta, \dot{\theta}, \dot{\psi}, \dot{\phi}_\mathrm{w}, \dot{\psi}_\mathrm{t}$, the torque actions: $u_\mathrm{w}, u_\mathrm{t}$ and the accelerations: $\ddot{\phi}, \ddot{\theta}, \ddot{\psi}, \ddot{\phi}_\mathrm{w}, \ddot{\psi}_\mathrm{t}$. The equations were then rearranged by Forster, through use of the Symbolic Toolbox in MATLAB, to isolate the five acceleration terms. We use this mapping along with Eqs. (B.8)–(B.9) to perform simulations of the unicycle.

B.4 Evaporator

The derivation of the equations that govern the evaporator can be found in the case study carried out by Newell & Lee (1989). They are displayed in Table B.2. The variables associated with this model are given in Table B.3 while the values for the constants are given in Table B.4. It is clear from Table B.2 that the state of the evaporator could be defined as $\mathbf{x} = [X_2, P_2, L_2]^\top$ since they appear as time derivatives. The actions are defined to be $\mathbf{u} = [F_2, P_{100}, F_{200}]^\top$ which leaves five disturbance variables F_1, F_3, X_1, T_1 and T_{200} which are set to the steady state values given in Table B.3.

Process Element	Equations
Mass Balance	$\dot{X}_2 = (F_1X_1 - F_2X_2)/M$
	$\dot{P}_2 = (F_4 - F_5)/C$
	$\dot{L}_2 = (F_1 - F_4 - F_2)/(\rho A)$
Energy Balance	$T_2 = 0.5616P_2 + 0.3126X_2 + 48.43$
	$T_3 = 0.507P_2 + 55.0$
	$F_4 = \big(Q_{100} - F_1C_P(T_2 - T_1)\big)/\lambda$
Heater Steam Jacket	$T_{100} = 0.1538P_{100} + 90.0$
	$Q_{100} = 0.16(F_1 + F_3)(T_{100} - T_2)$
	$F_{100} = Q_{100}/\lambda_s$
Condenser	$Q_{200} = \big(UA_2(T_3 - T_{200})\big)/\big(1 + UA_2/(2C_PF_{200})\big)$
	$T_{201} = T_{200} + Q_{200}/(F_{200}C_P)$
	$F_5 = Q_{200}/\lambda$

Table B.2: Evaporator equations as derived in Chapter 2.2 of Newell & Lee (1989). The evaporator state can be given by X_2, P_2 and L_2 since they appear as time derivatives.

Variable	Description	Value	Units	Constraints
F_1	feed flowrate	10.0	$\mathrm{kg\,min^{-1}}$	[0, 20.0]
F_2	product flowrate	2.0	$\mathrm{kg\,min^{-1}}$	[0, 4.0]
F_3	circulating flowrate	50.0	$\mathrm{kg\,min^{-1}}$	[0, 100.0]
F_4	vapour flowrate	8.0	$\mathrm{kg\,min^{-1}}$	[0, 16.0]
F_5	condensate flowrate	8.0	$\mathrm{kg\,min^{-1}}$	[0, 16.0]
X_1	feed composition	5.0	%	
X_2	product composition	25.0	%	
T_1	feed temperature	40.0	°C	
T_2	product temperature	84.6	°C	
T_3	vapour temperature	80.6	°C	
L_2	separator level	1.0	m	[0, 2.0]
P_2	operating pressure	50.5	kPa	[0, 100.0]
F_{100}	steam flowrate	9.3	$\mathrm{kg\,min^{-1}}$	[0, 20.0]
T_{100}	steam temperature	119.9	°C	
P_{100}	steam pressure	194.7	kPa	[0, 400.0]
Q_{100}	heater duty	339.0	kW	
F_{200}	cooling water flowrate	208.0	$\mathrm{kg\,min^{-1}}$	[0, 400.0]
T_{200}	cooling water inlet temperature	25.0	°C	
T_{201}	cooling water outlet temperature	46.1	°C	
Q_{200}	condenser duty	307.9	kW	

Table B.3: Evaporator variables, values at a steady state solution and constraints. Given in Table 2.1 in Newell & Lee (1989). Flow constraints are physical limitations of the relevant actuators whereas constraints on the level and pressures are recommended to avoid damage to the system.

Constant	**Description**	**Value**	**Units**
M	mass of liquid in evaporator	20.0	kg
C	converts mass of vapour into equivalent pressure	4.0	$\mathrm{kg\,kPa^{-1}}$
ρA	liquid density $\times$ cross-sectional area of separator	20.0	$\mathrm{kg\,m^{-1}}$
C_P	heat capacity of the liquor and cooling water	0.07	$\mathrm{kW\,min\,K^{-1}kg^{-1}}$
λ	latent heat of vapourisation of the liquor	38.5	$\mathrm{kW\,min\,kg^{-1}}$
λ_s	latent heat of steam at the saturated conditions	36.6	$\mathrm{kW\,min\,kg^{-1}}$
UA_2	overall heat transfer coefficient $\times$ heat transfer area	6.84	$\mathrm{kW\,K^{-1}}$

Table B.4: Physical constants in the evaporator model.

APPENDIX C MATLAB Code

C.1 Computing the Expected Cost

Here we provide the basic MATLAB code we used to find the expected cost of a trajectory, defined in Eq. (3.4), and the associated derivatives with respect to policy parameters given a probabilistic dynamics model and a control policy. The derivatives are computed using the recursive equations provided in Eqs. (3.10)–(3.13) and implemented in the function `value` given below. This piece of code is a collaborative effort with the author's main contribution being the matrix calculus convention for computing derivatives.

The input and output arguments to this function are clearly explained in the code comments and we note that we make use of structures to pass information around. In particular, the structure `dynmodel` contains all the information pertaining to the probabilistic dynamics model, including all sub-dynamics if we are using multiple dynamics models.

```
function [f, df] = value(p, m, s, dynmodel, policy, plant, cost, H)
% Compute expected (discounted) cost for a given initial state distribution.
%
% INPUTS:
% p               policy parameters chosen by minimize
% m               initial state mean
% s               initial state covariance matrix
% dynmodel        dynamics model structure
% policy          policy structure
%   policy.fcn    function which implements the policy
%   policy.p      parameters passed to the policy
% plant         plant structure
% cost          cost function structure
%   cost.fcn      function implementing cost
%   cost.gamma    discount factor
% H             length of prediction horizon
%
% OUTPUTS:
% f             expected cumulative (discounted) cost
% df            derivative structure of f wrt policy
%
% Copyright (C) 2008-2012 Marc Deisenroth & Carl Edward Rasmussen, 2012-06-25.
% Edited by Joe Hall 2012-10-02.

policy.p = p;            % overwrite policy.p with new parameters from minimize
p = unwrap(policy.p); dp = 0*p; L = zeros(1,H);

if nargout < 2 % -------------------------------------- no derivatives required

  for t = 1:H                                    % for all time steps in horizon
    [m, s] = propagate(m, s, plant, dynmodel, policy);         % get next state
    L(t) = cost.gamma^t.*cost.fcn(cost, m, s);      % expected discounted cost
  end

else % ---------------------------------------------- otherwise, get derivatives

  dmodp = zeros([size(m,1), length(p)]);
  dsodp = zeros([size(m,1)*size(m,1), length(p)]);

  for t = 1:H                                    % for all time steps in horizon
    [m, s, dmdmo, dsdmo, dmdso, dsdso, dmdp, dsdp] = ...
                    propagated(m, s, plant, dynmodel, policy); % get next state

    dmdp = dmdmo*dmodp + dmdso*dsodp + dmdp;
    dsdp = dSdmo*dmodp + dsdso*dsodp + dsdp;

    [L(t), dLdm, dLds] = cost.fcn(cost, m, s);             % predictive cost
    L(t) = cost.gamma^t*L(t);                                     % discount
    dp = dp + cost.gamma^t*( dLdm*dmdp + dLds*dsdp )';

    dmodp = dmdp; dsodp = dsdp;                                 % bookkeeping
  end

end

f = sum(L); df = rewrap(policy.p, dp);
```

The actual propagation of uncertainty from the current state to the next state, along with the associated derivatives, is carried out in `propagated`. The author's main contribution to the code was the multiple dynamics models framework, used in lines 64–76 and the `sliceModel` function in lines 92–104, and the convention for the calculation of the derivatives used throughout.

In order to implement and combine predictions from multiple dynamics models we use multiple structures `sub{i}` within the overall dynamics structure `dynmodel`. Each `sub` structure contains local indices to show which elements of the current state (`dyni`), current action (`dynu`) and outputs from previous sub-dynamics (`dynj`) are used as inputs to a particular sub-dynamics with outputs indexed by (`dyno`).

```
 1 function [Mo, So, dMdm, dSdm, dMds, dSds, dMdp, dSdp] = ...
 2                                       propagated(m, s, plant, dynmodel, policy)
 3 % Propagate the state distribution one time step forward with derivatives.
 4 %
 5 % INPUTS:
 6 % m        input mean                                              [ D        ]
 7 % s        input covariance                                        [ D  by  D ]
 8 % plant    plant structure containing useful indices
 9 % dynmodel dynamics model structure
10 %   dynmodel.inputs   training data inputs
11 %   dynmodel.target   training data targets
12 %   dynmodel.sub{i}   (optional) the ith sub-dynamics structure
13 % policy   control policy structure
14 %
15 % OUTPUTS:
16 % Mo       output mean                                             [ E        ]
17 % So       output covariance                                       [ E  by  E ]
18 % dMdm     output mean wrt input mean                              [ E  by  D ]
19 % dMds     output mean wrt input covariance                        [ E  by D*D]
20 % dSdm     output covariance wrt input mean                        [E*E by  D ]
21 % dSds     output covariance wrt input covariance                  [E*E by D*D]
22 % dMdp     output mean wrt policy parameters                       [ E  by  P ]
23 % dSdp     output covariance wrt policy parameters                 [E*E by  P ]
24 %
25 % where P is the number of policy parameters.
26 %
27 % Copyright (C) 2008-2012 by Marc Deisenroth, Carl Edward Rasmussen, Henrik
28 % Ohlsson, Andrew McHutchon and Joe Hall 2012-06-25.
29
30 if nargout < 3                                                 % no derivatives
31   [Mo, So] = propagate(m, s, plant, dynmodel, policy);
32   return
33 end
34
35 poli = plant.poli;               % state variables passed to the control policy
36 dyni = plant.dyni;            % state variables passed to the dynamics model(s)
37 difi = plant.difi;                   % state variables predicted as differences
38 angi = plant.angi;   % angle variables which are given a sin/cos representation
39
```

```matlab
D0 = length(m);                                           % size of the input mean
D1 = D0 + 2*length(angi);           % length after mapping all angles to sin/cos
D2 = D1 + length(policy.maxU);              % length after computing ctrl signal
D3 = D2 + D0;                                         % length after predicting

M = zeros(D3,1); M(1:D0) = m; S = zeros(D3); S(1:D0,1:D0) = s;      % initialise
Mdm = [eye(D0); zeros(D3-D0,D0)]; Sdm = zeros(D3*D3,D0);
Mds = zeros(D3,D0*D0); Sds = kron(Mdm,Mdm);
X = reshape(1:D3*D3,[D3 D3]); XT = X'; Sds = (Sds + Sds(XT(:),:))/2;
X = reshape(1:D0*D0,[D0 D0]); XT = X'; Sds = (Sds + Sds(:,XT(:)))/2;

% 1) Augment state distribution with trigonometric functions ------------------
i = 1:D0; j = 1:D0; k = D0+1:D1;
[M(k) S(k,k) C mdm sdm Cdm mds sds Cds] = gTrig(M(i), S(i,i), angi);

[S Mdm Mds Sdm Sds] = ...
      fillIn(S,C,mdm,sdm,Cdm,mds,sds,Cds,Mdm,Sdm,Mds,Sds,[ ],[ ],[ ],i,j,k,D3);

mm=zeros(D1,1); mm(i)=M(i); ss(i,i)=S(i,i)+diag(exp(2*dynmodel.hyp(end,:))/2);
[mm(k), ss(k,k) C] = gTrig(mm(i), ss(i,i), angi);     % noisy state measurement
q = ss(j,i)*C; ss(j,k) = q; ss(k,j) = q';

% 2) Compute distribution of the control signal -------------------------------
i = poli; j = 1:D1; k = D1+1:D2;
[M(k) S(k,k) C mdm sdm Cdm mds sds Cds Mdp Sdp Cdp] = ...
                                             policy.fcn(policy, mm(i), ss(i,i));

[S Mdm Mds Sdm Sds Mdp Sdp] = ...
      fillIn(S,C,mdm,sdm,Cdm,mds,sds,Cds,Mdm,Sdm,Mds,Sds,Mdp,Sdp,Cdp,i,j,k,D3);

% 3) Compute distribution of the change in state ------------------------------
ii = [dyni D1+1:D2]; j = 1:D2;
if isfield(dynmodel,'sub'), Nf = length(dynmodel.sub); else Nf = 1; end
for n=1:Nf                                 % potentially multiple dynamics models
  [dyn i k] = sliceModel(dynmodel,n,ii,D1,D2,D3); j = setdiff(j,k);

  [M(k) S(k,k) C mdm sdm Cdm mds sds Cds] = dyn.fcn(dyn, M(i), S(i,i));

  [S Mdm Mds Sdm Sds Mdp Sdp] = ...
      fillIn(S,C,mdm,sdm,Cdm,mds,sds,Cds,Mdm,Sdm,Mds,Sds,Mdp,Sdp,[ ],i,j,k,D3);

  j = [j k];                                   % update 'previous' state vector
end

% 4) Compute distribution of the next state -----------------------------------
P = [zeros(D0,D2) eye(D0)]; P(difi,difi) = eye(length(difi));  P = sparse(P);
Mo = P*M; So = P*S*P'; So = (So+So')/2;

PP = kron(P,P);
dMdm =  P*Mdm; dMds =  P*Mds; dMdp =  P*Mdp;
```

```
dSdm = PP*Sdm; dSds = PP*Sds; dSdp = PP*Sdp;

X = reshape(1:D0*D0,[D0 D0]); XT = X';                      % ensure symmetrical S
dSdm = (dSdm + dSdm(XT(:),:))/2; dMds = (dMds + dMds(:,XT(:)))/2;
dSds = (dSds + dSds(XT(:),:))/2; dSds = (dSds + dSds(:,XT(:)))/2;
dSdp = (dSdp + dSdp(XT(:),:))/2;


% A1) Separate multiple dynamics models ------------------------------------------------
function [dyn i k] = sliceModel(dynmodel,n,ii,D1,D2,D3) % separate sub-dynamics
if isfield(dynmodel,'sub')
  dyn = dynmodel.sub{n}; do = dyn.dyno; D = length(ii)+D1-D2;
  if isfield(dyn,'dyni'), di=dyn.dyni; else di=[]; end
  if isfield(dyn,'dynu'), du=dyn.dynu; else du=[]; end
  if isfield(dyn,'dynj'), dj=dyn.dynj; else dj=[]; end
  i = [ii(di) D1+du D2+dj]; k = D2+do;
  dyn.inputs = [dynmodel.inputs(:,[di D+du]) dynmodel.target(:,dj)];   % inputs
  dyn.target = dynmodel.target(:,do);                                  % targets
else
  dyn = dynmodel; k = D2+1:D3; i = ii;
end

% A2) Apply chain rule and fill out cross covariance terms ---------------------
function [S Mdm Mds Sdm Sds Mdp Sdp] = ...
       fillIn(S,C,mdm,sdm,Cdm,mds,sds,Cds,Mdm,Sdm,Mds,Sds,Mdp,Sdp,dCdp,i,j,k,D)

if isempty(k), return; end

X = reshape(1:D*D,[D D]); XT = X';                          % vectorised indices
I=0*X; I(i,i)=1; ii=X(I==1)'; I=0*X; I(k,k)=1; kk=X(I==1)';
I=0*X; I(j,i)=1; ji=X(I==1)'; I=0*X; I(j,k)=1; jk=X(I==1)'; kj=XT(I==1)';

Mdm(k,:)  = mdm*Mdm(i,:) + mds*Sdm(ii,:);                             % chainrule
Mds(k,:)  = mdm*Mds(i,:) + mds*Sds(ii,:);
Sdm(kk,:) = sdm*Mdm(i,:) + sds*Sdm(ii,:);
Sds(kk,:) = sdm*Mds(i,:) + sds*Sds(ii,:);
dCdm      = Cdm*Mdm(i,:) + Cds*Sdm(ii,:);
dCds      = Cdm*Mds(i,:) + Cds*Sds(ii,:);
if isempty(dCdp) && nargout > 5
  Mdp(k,:)  = mdm*Mdp(i,:) + mds*Sdp(ii,:);
  Sdp(kk,:) = sdm*Mdp(i,:) + sds*Sdp(ii,:);
  dCdp      = Cdm*Mdp(i,:) + Cds*Sdp(ii,:);
elseif nargout > 5
  aa = length(k); bb = aa^2; cc = numel(C);
  mdp = zeros(D,size(Mdp,2)); sdp = zeros(D*D,size(Mdp,2));
  mdp(k,:)  = reshape(Mdp,aa,[]); Mdp = mdp;
  sdp(kk,:) = reshape(Sdp,bb,[]); Sdp = sdp;
  Cdp       = reshape(dCdp,cc,[]); dCdp = Cdp;
end

```

```
q = S(j,i)*C; S(j,k) = q; S(k,j) = q';                          % off−diagonal
SS = kron(eye(length(k)),S(j,i)); CC = kron(C',eye(length(j)));
Sdm(jk,:) = SS*dCdm + CC*Sdm(ji,:); Sdm(kj,:) = Sdm(jk,:);
Sds(jk,:) = SS*dCds + CC*Sds(ji,:); Sds(kj,:) = Sds(jk,:);
if nargout > 5
  Sdp(jk,:) = SS*dCdp + CC*Sdp(ji,:); Sdp(kj,:) = Sdp(jk,:);
end
```

C.2 Prior Over Sampled Systems

Here we include the code used to evaluate the general sampled prior based on the Runge-Kutta family of methods and outlined in Sec. 5.4. In particular, for a base continuous-time prior of the form $\mathcal{GP}(\mathbf{m}_{\text{lin}}, \mathbf{K}_{\text{add}} + \mathbf{K}_{\text{full}})$. The code is well commented and should be straightforward to follow.

```

function [Mo, Ko] = prior(lh, rkp, Z)
% A state space prior for discrete−time predictions with priors over the
% continuous−time dynamics but discrete data. The mapping from continuous to
% discrete−time is approximated using a Runge−Kutta (RK) method of the form:
%
%   x(k+1) = x(k) + rkp.dt * sum{i=1:R}   bi   * f(pi)
%       pi = x(k) + rkp.dt * sum{c=1:i−1} aic * f(pk)
%
% where "x" [E] is the state, "rkp.dt" is the discrete timestep, "f" [D to E]
% is the continuous−time dynamics (assumed to be stationary). The parameters of
% the RK method are the lower triangular RK matrix "rkp.a" [R by R] and
% weights "rkp.b" [R].
%
% The mean and the covariance that are produced are made up of the blocks
%
%   Mo_m  = dt   * sum{i=1:R}            bi      *   E[f(pmi)]
%   Ko_mn = dt^2 * sum{i=1:R} sum{j=1:R} bi * bj * cov[f(pmi),f(pnj)]
%
% where m and n are in [1..data set size]. The continuous−time structure is
% assumed to take the form of a linear part plus an additive part plus a
% nonlinear part
%
%   xdot = f(z) = V'*z + sum{i=1:D} fi(z_i) + ff(z)
%
% The continuous−time kernel is therefore made up of a linear part, an
% additive squared exponential and a full squared exponential part. Therefore
% the log hyper−parameters are given by
%
%               |V11 .. V1E|
%   lh.lin =    | :   ..  : | in [D by E]
%                |VD1 .. VDE|
%
%                |L11 .. L1E|
%                | :   ..  : |
%                |LD1 .. LDE|
%   lh.add = log|a11 .. a1E| in [2D by E]
%                | :   ..  : |
%                |aD1 .. aDE|
%
%                |L11 .. L1E|
%                | :   ..  : |
%   lh.squ = log|LD1 .. LDE| in [D+1 by E]
%                |aa1 .. aaE|
%
% Training data is given in the form:
%
%          |x1' u1'|
%     Z = |    :    |
%          |xn' un'|
%
% and the input "u" is assumed to be applied in a zero-order hold fashion.
% Written by Joe Hall 2012-12-15.

dt = rkp.dt;                                               % discrete timestep
R = length(rkp.b);                                          % order of RK scheme
[n,D] = size(Z);           % size of data set and dimension of state-action space
E = size(lh,2); F = D-E;              % dimension of state space and action space
nD = n*D; nDR = nD*R; nE = n*E; nER = nE*R; nF = n*F;

M = zeros(nER,1); Mp = zeros(nDR,1); Mp(1:nD) = Z(:);             % initialise
S = zeros(nER);    Sp = zeros(nDR);

Pa = sparse( kron(rkp.a,[eye(nE);zeros(nF,nE)])*dt );          % RK matrix * dt
Pb = sparse( kron(rkp.b,eye(nE))*dt );                  % expanded RK weights * dt
Q  = sparse( kron(eye(E),kR([D 1],eye(n))) );               % trace-type operator

VV = lh(1:D,:); V = VV(:);                                      % linear weights


% First time round its just the normal kernels
% 0a) Update the Mean E[f(P1)] -----------------------------------------------
ix = 1:nE;
M(ix) = Q'*( kR([E 1],Z(:)) .* kL(V,[n 1]) );

% 0b) Update the Covariance cov[f(P1),f(P1)] ---------------------------------
SS = zeros(E);
for e=1:E                                             % loop over state dimensions
  ee = (e-1)*n + (1:n);

  % SE Part
```

```
    iL = diag(exp(-lh(4*D+(1:D),e))); alp2 = exp(2*lh(5*D+1,e));
    K = sq_dist(iL*Z');
    K = alp2*exp(-K/2);
    SS(ee,ee) = SS(ee,ee) + K;

    % Additive Part
    for d=1:D
      iL = diag(exp(-lh(2*D+d,e))); alp2 = exp(2*lh(3*D+d,e));
      K = sq_dist(iL*Z(:,d)');
      K = alp2*exp(-K/2);
      SS(ee,ee) = SS(ee,ee) + K;
    end
  end % e
  S(ix,ix) = SS;

  % Now with propagation of uncertainty
  for ii=2:R                                         % loop over order of RK method
    % 1) Update the Mean E[Pi] and Covariance cov[Pi,Pj] ------------------------
    i = nD*(ii-1)+(1:nD);
    Mp(i) = Z(:) + Pa(i,:)*M;
    Sp(i,:) = Pa(i,:)*S*Pa'; Sp(:,i) = Sp(i,:)';

    % 2a) Update the Mean E[f(Pi)] ----------------------------------------------
    ix = nE*(ii-1)+(1:nE);
    M(ix) = Q'*( kR([E 1],Mp(i)) .* kL(V,[n 1]) );

    % 2b) Update the Covariance cov[f(Pi),f(Pj)] --------------------------------
    for jj=1:ii
      j  = nD*(jj-1)+(1:nD);
      jx = nE*(jj-1)+(1:nE);
      % Linear Part
      SS = Q'*( kR([E E],Sp(i,j)+Mp(i)*Mp(j)') .* kL(V*V',[n n]) )*Q ...
                                                            - M(ix)*M(jx)';

      % SE Part
      MMp = Mp([i j]); SSp = Sp([i j],[i j]);
      for e=1:E                                        % loop over state dimensions
        iL = diag(exp(-lh(4*D+(1:D),e)));
        iLL = [iL -iL; -iL iL]/sqrt(2);
        alp2 = exp(2*lh(5*D+1,e));

        iLa = diag(exp(-lh(2*D+(1:D),e)));
        iLLa = [iLa -iLa; -iLa iLa]/sqrt(2);
        alp2a = exp(2*lh(3*D+(1:D),e));
        for a=1:n                                        % loop over training data
          for b=1:n                                                 % ...and again
            aa = (e-1)*n+a; bb = (e-1)*n+b;
            mmp = MMp([a:n:nD (b:n:nD)+nD]);
            ssp = SSp([a:n:nD (b:n:nD)+nD],[a:n:nD (b:n:nD)+nD]);

            BB = iLL*ssp*iLL + eye(2*D);
            iR = iLL*(BB\iLL);
            SE = alp2/sqrt(det(BB)) * exp( -mmp'*iR*mmp/2 );

            SS(aa,bb) = SS(aa,bb) + SE;

      % Additive Part
            for d=1:D
              dd = [d d+D]; iLLad = iLLa(dd,dd);
              mmpd = mmp(dd); sspd = ssp(dd,dd);

              BB = iLLad*sspd*iLLad + eye(2);
              iR = iLLad*(BB\iLLad);
              aSE = alp2a(d)/sqrt(det(BB)) * exp( -mmpd'*iR*mmpd/2 );

              SS(aa,bb) = SS(aa,bb) + aSE;
            end % d
          end % bb
        end % aa
      end % e

      S(ix,jx) = SS; S(jx,ix) = SS';
    end % jj
  end % ii

  % Return prior mean and covariance
  Mo = full(Pb*M);
  Ko = full(Pb*S*Pb'); Ko = (Ko+Ko')/2;


  function C = kR(dim,A)
  % Implement the kronecker product: ones(dim) ox A
  C = kron(ones(dim),A);

  function C = kL(A,dim)
  % Implement the kronecker product: A ox ones(dim)
  C = kron(A,ones(dim));
```

Bibliography

Abbeel, P., Coates, A., Quigley, M. & Ng, A.Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19*.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, **10**, 251–276.

Anderson, B.D.O. & Moore, J.B. (1979). *Optimal Filtering*. Prentice Hall.

Ariyur, K.B. & Krstić, M. (2003). *Real-Time Optimization by Extremum-Seeking Control*. Wiley-Interscience.

Åström, K.J. & Wittenmark, B. (1994). *Adaptive Control*. Prentice Hall, 2nd edn.

Ažman, K. & Kocijan, J. (2008). Non-linear model predictive control for models with local information and uncertainties. *Transactions of the Institute of Measurement and Control*, **30**, 371–396.

Ažman, K. & Kocijan, J. (2011). Dynamical systems identification using Gaussian process models with incorporated local models. *Engineering Applications of Artificial Intelligence*, **24**, 398–408.

Bastin, G., Nešić, D., Tan, Y. & Mareels, I. (2009). On extremum seeking in bioprocesses with multivalued cost functions. *American Institute of Chemical Engineers: Biotechnology Progress*, **25**, 683–689.

Baxter, J. & Bartlett, P.L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, **15**, 319–350.

Baxter, J., Tridgell, A. & Weaver, L. (2001). Learning to play chess using temporal differences. *Machine Learning*, **40**, 243–263.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

Bertsekas, D.P. & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bitmead, R.R., Gevers, M. & Wertz, V. (1990). *Adaptive Optimal Control: The Thinking Man's GPC*. Prentice Hall International Series in Systems and Control Engineering, Prentice Hall.

Boyd, S. & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Brookes, M. (2005). The matrix reference manual: http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/.

Buchli, J., Stulp, F., Theodorou, E. & Schaal, S. (2011). Learning variable impedance control. *International Journal of Robotics Research*.

Choi, J.Y., Krstić, M., Ariyur, K.B. & Lee, J.S. (2002). Extremum seeking control for discrete-time systems. *IEEE Transactions on Automatic Control*, **47**, 318–323.

Deisenroth, M.P. (2009). *Efficient Reinforcement Learning using Gaussian Processes*. Ph.D. thesis, Cambridge University.

Deisenroth, M.P. & Rasmussen, C.E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, Bellevue, WA.

Deisenroth, M.P., Hennig, P. & Rasmussen, C.E. (2011). Robotic unicycle loss function, Internal report, Cambridge University.

Deisenroth, M.P., Turner, R., Huber, M.F., Haneback, U.D. & Rasmussen, C.E. (2012). Robust filtering and smoothing with Gaussian processes. *IEEE Transactions on Automatic Control*.

Duvenaud, D., Nickisch, H. & Rasmussen, C.E. (2011). Additive Gaussian processes. In *Advances in Neural Information Processing Systems 25*, 1–8.

Engel, Y., Mannor, S. & Meir, R. (2003). Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning*, vol. 20, 154–161, Washington DC, USA.

Engel, Y., Mannor, S. & Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, vol. 22 of *ACM International Conference Proceeding Series*, 201–208, ACM, Bonn, Germany.

Ernst, D., Geurts, P. & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, **6**, 503–556.

Ernst, D., Glavic, M., Capitanescu, F. & Wehenkel, L. (2009). Reinforcement learning versus model predictive control: A comparison on a power system problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **39**, 517–529.

Fabri, S. & Kadirkamanathan, V. (1998). Dual adaptive control of nonlinear stochastic systems using neural networks. *Automatica*, **34**, 245–253.

Fel'dbaum, A.A. (1960–61). Dual control theory, parts I-IV. *Automation and Remote Control*, **21,22**, 874–880, 1033–1039, 1–12, 109–121.

Florian, R.V. (2007). Correct equations for the dynamics of the cart-pole system, Center for Cognitive and Neural Studies (Coneural), Romania.

Forster, D. (2009). *Robotic unicycle*. Master's thesis, University of Cambridge.

Geist, M. & Pietquin, O. (2010a). A brief survey of parametric value function approximation. Tech. rep., Supélec.

Geist, M. & Pietquin, O. (2010b). Kalman temporal differences. *Journal of Artificial Intelligence Research*, **39**, 483–532.

Geist, M., Pietquin, O. & Fricout, G. (2009). From supervised to reinforcement learning: A kernel-based Bayesian filtering framework. *International Journal On Advance in Software*, **2**, 101–116.

Girard, A., Rasmussen, C.E., Quiñonerno-Candela, J. & Murray-Smith, R. (2003). Gaussian process priors with uncertain inputs - Application to multiple-step ahead time series forecasting. In S. Becker, S. Thrun & K. Obermayer, eds., *Advances in Neural Information Processing Systems 15*, 529–536, MIT Press, Cambridge, MA.

Hall, J., Rasmussen, C.E. & Maciejowksi, J.M. (2011). Reinforcement learning with reference tracking control in continuous state spaces. In *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference*.

Hall, J., Rasmussen, C.E. & Maciejowksi, J.M. (2012). Modelling and control of nonlinear systems using Gaussian processes with partial model information. In *Proceedings of the 51st IEEE Conference on Decision and Control*.

Hastie, T.J. & Tibshirani, R.J. (1990). *Generalized Additive Models*. Chapman & Hall/CRC.

Hjalmarsson, H. (2002). Iterative feedback tuning - An overview. *International Journal of Adaptive Control and Signal Processing*, **16**, 373–395.

Hjalmarsson, H., Gunnarsson, S. & Gevers, M. (1994). A convergent iterative restricted complexity control design scheme. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, 1735–1740.

Hjalmarsson, H., Gevers, M., Gunnarsson, S. & Lequin, O. (1998). Iterative feedback tuning: Theory and applications. *IEEE Control Systems Magazine*, **18**, 26–41.

Ioannou, P. & Fidan, B. (2006). *Adaptive Control Tutorial*. Advances in Design and Control, SIAM (Society for Industrial and Applied Mathematics).

Jacobson, D.H. & Mayne, D.Q. (1970). *Differential Dynamic Programming*. American Elsevier, New York, NY.

Julier, S.J. & Uhlmann, J.K. (1997). A new extension of the kalman filter to nonlinear systems. In *In International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, vol. 3068, 182–193.

Kaelbling, L.P., Littman, M.L. & Moore, A.W. (1996). Reinforcement learning: A survey. *Artificial Intelligence Research*, **4**, 237–285.

Kakade, S. (2002). A natural policy gradient. In *Advances in Neural Information Processing Systems 14*.

Kalman, R.E. (1960). Contributions to the theory of optimal control. *Bull. Soc. Math. Mex*, **5**, 102–119.

Krstić, M., Kanellakopoulos, I. & Kokotović, P.V. (1995). *Nonlinear and Adaptive Control Design*. Adaptive and Learning Systems for Signal Processing, Communications and Control, Wiley-Interscience.

Ljung, L. (1999). *System Identification: Theory for the User*. Prentice Hall, 2nd edn.

Maciejowski, J.M. (2002). *Predictive Control with Constraints*. Pearson Education Limited.

Marbach, P. & Tsitsiklis, J.N. (2001). Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control*, **46**, 191–209.

Maybeck, P.S. (1982). *Stochastic Models, Estimation and Control*. Academic Press.

McHutchon, A. & Rasmussen, C.E. (2011). Gaussian process training with input noise. In *Advances in Neural Information Processing Systems 25*.

Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A*, **209**, 441–458.

Mitrovic, D., Klanke, S., Osu, R., Kawato, M. & Vijayakumar, S. (2010a). A computational model of limb impedance control based on principles of internal model uncertainty. *PLoS ONE*, **5**.

Mitrovic, D., Klanke, S. & Vijayakumar, S. (2010b). *Motor Learning to Interactive Learning in Robotics*, chap. Adaptive Optimal Feedback Control with Learned Internal Dynamics Model, 65–84. Springer-Verlag Berlin Heidelberg.

Morimoto, J., Zeglin, G. & Atkeson, C.G. (2003). Minimax differential dynamic programming: Application to a biped walking robot. In *Proceedings of the IEEE/RSJ International Conterence on Intelligent Robots and Systems*, vol. 2, 1927–1932.

Nešić, D. (2009). Extremum seeking control: Convergence analysis. In *Proceedings of the 10th European Control Conference*, 1702–1715, Budapest, Hungary.

Newell, R.B. & Lee, P.L. (1989). *Applied Process Control - A Case Study*. Prentice Hall, New York, NY.

Ng, A.Y. & Jordan, M.I. (2000). Pegasus: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*.

Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E. & Liang, E. (2004a). Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*.

Ng, A.Y., Kim, J.H., Jordan, M.I. & Sastry, S. (2004b). Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16*.

Peters, J. & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, **21**, 682–697.

Peters, J., Vijayakumar, S. & Schaal, S. (2005). Natural actor-critic. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 280–291.

Quiñonerno Candela, J. & Rasmussen, C.E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, **6**, 1939–1959.

Rasmussen, C.E. & Williams, C.K.I. (2006). *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, MA.

Särkkä, S. (2011). Linear operators and stochastic partial differential equations in Gaussian process regression. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, Espoo, Finland.

Simpkins, A., de Callafon, R. & Todorov, E. (2008). Optimal trade-off between exploration and exploitation. In *Proceedings of the American Control Conference*, Seattle, Washington.

Snelson, E. & Gharamani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In B. Weiss, B. Schölkopf & J. Platt, eds., *Advances in Neural Information Processing Systems 18*, 1259–1266.

Solak, E., Murray-Smith, R., Leithead, W.E., Leith, D.J. & Rasmussen, C.E. (2003). Derivative observations in Gaussian process models of dynamic systems. In S. Thrun, S. Becker & K. Obermayer, eds., *Advances in Neural Information Processing Systems 15*, 1033–1040, Vancouver, Canada.

Sutton, R.S. & Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA.

Sutton, R.S., Barto, A.G. & Williams, R.J. (1992). Reinforcement learning is direct adaptive optimal control. In *Proceedings of the American Control Conference*, 2143–2146.

Sutton, R.S., MacAllester, D., Singh, S. & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, 1057–1063, MIT Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, **8**, 257–277.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, **38**.

Theodorou, E., Buchli, J. & Schaal, S. (2010a). A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, **11**, 3153–2197.

Theodorou, E., Buchli, J. & Schaal, S. (2010b). Reinforcement learning of motor skills in high dimensions: A path integral approach. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2397–2403, Anchorage, Alaska.

Ting, J.A., Vijayakumar, S. & Schaal, S. (2010). Locally weighted regression for control. In C. Sammut & G.I. Webb, eds., *Encyclopedia of Machine Learning*, 613–624, Springer.

Todorov, E. & Li, W. (2005). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the American Control Conference*.

Todorov, E. & Tassa, Y. (2009). Iterative local dynamic programming. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 90–95, Nashville, TN.

Tse, E. & Bar-Shalom, Y. (1972). An actively adaptive control for linear systems with random parameters via the dual control approach. In *Proceedings of the IEEE Conference on Decision and Control and 11th Symposium on Adaptive Processes*, 623 – 627.

Tse, E. & Bar-Shalom, Y. (1975). Generalized certainty equivalence and dual effect in stochastic control. *IEEE Transactions on Automatic Control*, **20**, 817–819.

Tsitsiklis, J.N. & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, **42**, 674–690.

Unbehauen, H. (2000). Adaptive dual control: A survey. In *Proceedings of the Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 171–180, Lake Louise, Alta.

Vijayakumar, S., D'Souza, A. & Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation*, **17**, 2602–2634.

Watkins, C.J.C.H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, University of Cambridge, Cambridge, UK.

Watkins, C.J.C.H. & Dayan, P. (1992). $\mathcal{Q}$-Learning. *Machine Learning*, **8**, 279–292.

Williams, R.J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, **8**, 229–256.

Wittenmark, B. (1995). Adaptive dual control methods: An overview. In *Proceedings of the 5th IFAC Symposium on Adaptive Systems in Control and Signal Processing*, 67–72.

Wittenmark, B. (2000). Adaptive dual control. Draft for EOLSS.