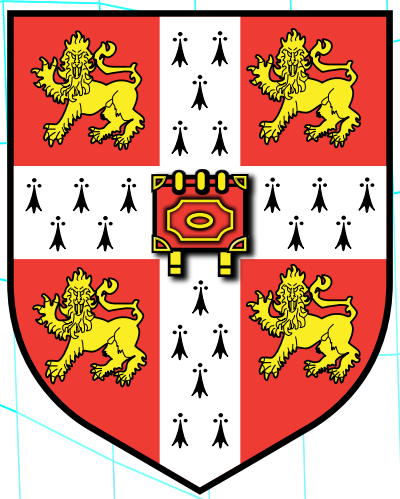
Nonlinear Modelling and Control using Gaussian Processes

Andrew McHutchon
Churchill College
August 2014

A thesis presented for the degree of
Doctor of Philosophy



Department of Engineering
University of Cambridge

This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements

This dissertation contains 55,487 words and 79 figures

Abstract

In many scientific disciplines it is often required to make predictions about how a system will behave or to deduce the correct control values to elicit a particular desired response. Efficiently solving both of these tasks relies on the construction of a model capturing the system's operation. In the most interesting situations, the model needs to capture strongly nonlinear effects and deal with the presence of uncertainty and noise. Building models for such systems purely based on a theoretical understanding of underlying physical principles can be infeasibly complex and require a large number of simplifying assumptions. An alternative is to use a data-driven approach, which builds a model directly from observations. A powerful and principled approach to doing this is to use a Gaussian Process (GP).

In this thesis we start by discussing how GPs can be applied to data sets which have noise affecting their inputs. We present the 'Noisy Input GP', which uses a simple local-linearisation to refer the input noise into heteroscedastic output noise, and compare it to other methods both theoretically and empirically. We show that this technique leads to a effective model for nonlinear functions with input and output noise. We then consider the broad topic of GP state space models for application to dynamical systems. We discuss a very wide variety of approaches for using GPs in state space models, including introducing a new method based on moment-matching, which consistently gave the best performance. We analyse the methods in some detail including providing a systematic comparison between approximate-analytic and particle methods. To our knowledge such a comparison has not been provided before in this area. Finally, we investigate an automatic control learning framework, which uses Gaussian Processes to model a system for which we wish to design a controller. Controller design for complex systems is a difficult task and thus a framework which allows an automatic design directly from data promises to be extremely useful. We demonstrate that the previously published framework cannot cope with the presence of observation noise but that the introduction of a state space model dramatically improves its performance. This contribution, along with some other suggested improvements opens the door for this framework to be used in real-world applications.

Acknowledgements

I could not have completed this work in isolation and so before we launch into the technical matter I must first take the time to acknowledge and thank a number of my colleagues, friends, and family.

Firstly, I owe a debt of gratitude to my supervisor, Carl, whom I've worked with for five years now: through both my MEng and my PhD. It's been great, Carl, I've really enjoyed working with you and I really hope the collaboration continues in the future! Secondly, many thanks to Ulrich for the inordinate amount of time he spent finding typos and mistakes in my draft, and for being a great colleague, friend, and brother in Christ (4 a.m. in Beijing!). As always, thanks to my wife, Ruth, for her support and for reminding me that no matter how tough my day had been hers was worse because at least no one died during my day. I also want to thank all the faculty, post-docs, and my fellow students in the CBL for providing such a stimulating and friendly environment in which to work. In particular, thanks to Roger and David for the many hours of discussion and help. Also thanks to our administrator Diane, without whom the lab could not run! I must also thank all my friends at Barney's for their support and friendship over the years, Jo, Blaise, Emma, all the Jons, Amineh, the Hannahs, Beers, Tom, the list goes on! Cheers guys. Thanks also to Lucy for her typo spotting skills! Thanks to my family for their support and for imparting the enjoyment in your work and drive to get it done that are a prerequisite for completing a PhD. Finally, but most importantly, thank you Jesus for your extraordinary patience and perseverance with me and the countless gifts (and challenges!) you've given to me.

Contents

Chapter 1

Introduction

1.1 Motivation

Control theory, as we know the field today, has been studied for over 150 years and yet it only ever becomes more important as we seek to automate control of ever more advanced systems. Many of these systems, such as cars, have a direct, observable impact on our lives and thus the challenges control engineers must overcome are very much in the spotlight. These control challenges are often beyond stabilisation of simple systems around a set point and are focused on complex planning and decision operations or control of systems for which there is no simple mathematical model from which we may derive a control law. For this reason many authors are expanding the borders of traditional control theory to include more data-driven approaches from the field of machine learning. These approaches are able to design and adapt controllers directly from data measurements rather than requiring, or being limited by, a previously created mathematical model. This potentially allows us to design controllers for complex systems, which can adapt to their environments and are not limited by the knowledge of the control designer. It is this idea which is the driving motivation for this thesis.

We investigate and improve one particular machine learning/control framework, known as 'Pilco', which uses Gaussian Processes to build a model of the system dynamics directly from data. Unfortunately, observed data is usually corrupted by noise and may not contain all the information we would like. Our investigations into this control learning framework, which can be found in chapter 4, demonstrate that observation noise is a serious problem for the algorithm. As observation noise is extremely common this is a fatal flaw in the framework and one which we, partly, overcome in this thesis.

As building models in the face of uncertain or noisy data is an important challenge in and of itself, we will develop and analyse the theory separately from the control application; this is the subject of chapters 2 and 3. We return to the control learning framework, armed with the results of the proceeding chapters, in chapter 4.

1.2 Thesis Overview

In this chapter we introduce and motivate Gaussian Processes and some of the the key concepts, which will be referred to in later chapters. In chapter 2, we discuss modelling with Gaussian Processes in the presence of input noise — a common situation which can cause the classic GP formulation to perform poorly. In chapter 3 we provide a detailed investigation into modelling dynamical systems with GPs. Dynamical systems, those which change their state over time, can be found in nearly every area of our lives and are becoming even more important as we seek further automation of common tasks (for example, consider the recent investment in driverless cars). Thus modelling in this area is vitally important and is often a prerequisite for the design of control strategies, which is the topic of chapter 4. There, we consider an automatic control learning framework, which can design a controller to solve a task based on observed data from the real system. This framework has been shown to be very powerful on noise-free tasks. We provide some of the necessary steps to apply the framework to real-world data, that is, in the presence of non-negligible noise. Finally, we present our conclusions, and some directions for future research, in chapter 5.

1.3 Background

A common task in data modelling applications is to estimate a function $f(\boldsymbol{x})$ given some noisy observations $\boldsymbol{y}$ at particular input locations $\boldsymbol{x}$. For example, consider the data shown in figure 1.1 and imagine we believe the data is generated from some hidden, underlying process which we want to model. A simple approach to doing this is to pick a particular function class to use as our model, say second order polynomials, and then fit the model to the data by, for example, minimising the squared error between the predicted values and the observed values. This gives us a so called function of 'best-fit'. However, there are two fundamental flaws with this approach. Firstly, how do we pick the class of functions to use as our model? As figure 1.1 shows, there are multiple possible functions, which all could be valid fits to the observed data, depending on how much noise there is: is the underlying function simple and the observations just very noisy, or should we be trying to fit the observations exactly? The quadratic distance metric will favour the more complex models which fit the data exactly, but this will lead to overfitting in the presence of noise. We therefore need to decide how to trade-off model fit with model complexity, typically done via the use of a regularisation term. Even if we enforce that the function should pass exactly through the observed points, there are still an infinite number of possibilities (e.g. all polynomials of order greater than four). A common approach to try and avoid this model selection problem is to fit multiple models using different function classes and then use cross-validation on a held-out test set to pick the model with the best generalisation performance. However, this still restricts our model to belonging to one of a pre-determined set of classes. Also this approach typically becomes unwieldy if we include too many different classes — whilst fitting polynomials might be straightforward, it is a lot more complicated once we start including radial basis functions with different numbers of components and different choices for basis function.

The second fundamental flaw with picking a single function of 'best-fit' is that we have no way to represent the uncertainty in our model. Clearly we should be very sceptical about our model's prediction at a location far away from the data, but as figure 1.1 shows, there can also be significant uncertainty about the function's behaviour even within the range of the data. If we pick just one of

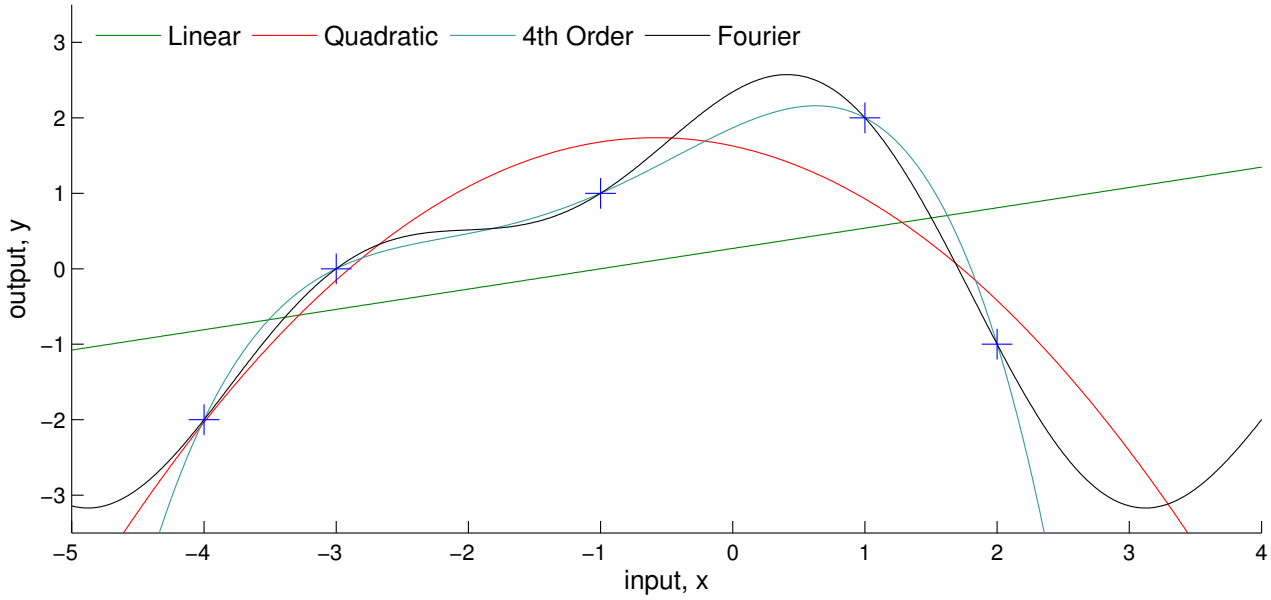


Figure 1.1: Four example function fits to observed data (blue crosses) out of an infinite number of possibilities

the example functions shown in the figure and proceed to make predictions with it we should expect very poor modelling performance: the predictions will frequently be wrong and the model will give no indication of how much you can trust its output. Our two modelling requirements are therefore to

1. Find a model class which allows us to easily consider a very wide variety of candidate functions

2. Find a model which can quantify and express the uncertainty in its fit

The solution to both of these requirements can be found in the field of Bayesian non-parameterics. In Bayesian reasoning a probability distribution is used to represent our belief over the value of a particular variable, for example the function value f at an input location $\boldsymbol{x}$. We encapsulate our current knowledge about the unknown variable f in the prior distribution $p(f)$ and specify how an observed output y at input $\boldsymbol{x}$ is related to the function variable f, via the likelihood $p(y \mid f, \boldsymbol{x})$. Bayes' rule then tells us how to update our belief over the uncertain function variable given the observations we have,

$$p(f \mid \boldsymbol{x}, y) \;=\; \frac{p(y \mid f, \boldsymbol{x})\, p(f)}{p(y \mid \boldsymbol{x})} \tag{1.1}$$

where $p(f \mid \boldsymbol{x}, y)$ is called the posterior and where $p(y \mid \boldsymbol{x})$ ensures that the posterior normalises (integrates to 1). By taking this approach we end up with a probability distribution over the function value at a particular input $f(\boldsymbol{x})$, conditioned on the observed data. This is in stark contrast to the 'best-fit' approach which only provided a single value for $f(\boldsymbol{x})$. This provides a solution for the second modelling requirement as listed above. We must now consider which model class to use; in the Bayesian setting this can be seen as choosing the prior over functions. For this we turn to the Gaussian Process.

1.4 Gaussian Processes

1.4.1 Introduction to Gaussian Processes

Gaussian Processes (GPs) (Rasmussen and Williams, 2006; O'Hagan and Kingman, 1978) are a flexible and principled method for modelling functions from data. Rather than providing a single 'best-fit' to the observed data, GPs take a Bayesian approach and provide a complete posterior distribution over possible functions. As mentioned above, in Bayesian reasoning, a probability distribution describes our belief over the value of a finite-dimensional variable. Roughly speaking a stochastic process extends this concept to a distribution over functions (or an infinite collection of variables). As its name suggests, a Gaussian Process is the extension of the Gaussian distribution to functions. Figure 1.2 shows the familiar Gaussian distribution in one, two, and three dimensions. A Gaussian Process can be though of as a distribution on the space of functions where the function values (or outputs) at any and every point are jointly Gaussian. A Gaussian distribution is fully determined by its mean and variance, as is a Gaussian Process, except that here they are generalised into a mean function, $m(\boldsymbol{x})$, and (co)variance function, $k(\boldsymbol{x})$. These allow the marginal mean and variance of the GP to vary across the input space, as is shown in figure 1.3

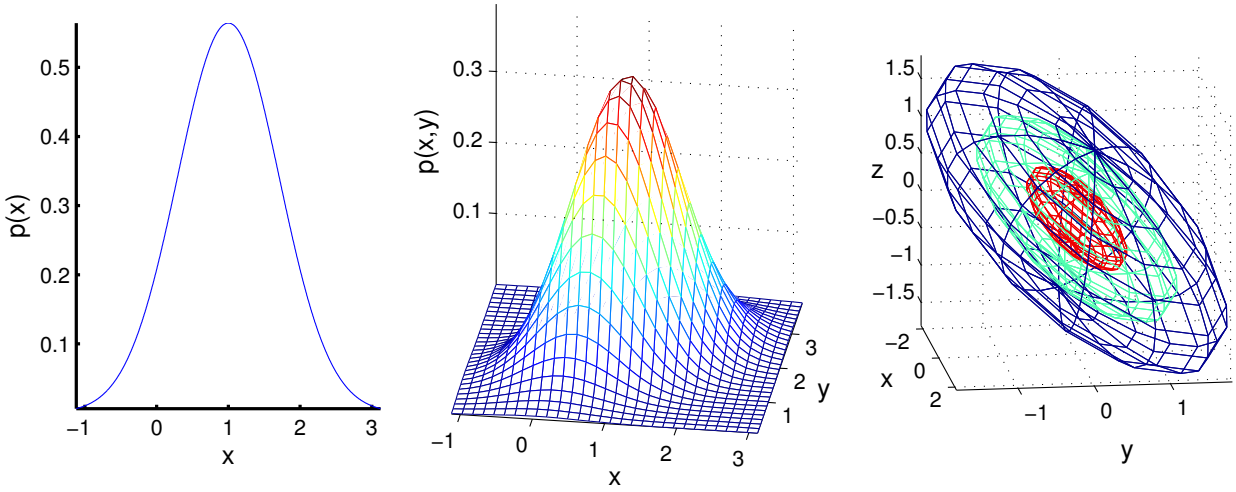


Figure 1.2: A Gaussian distribution in 1D (left), 2D (middle), and 3D (right). The first two plots show the probability density function over the input space, the third plot shows contours of iso-probabilities, where red indicates a contour of high probability.

Figure 1.3 shows how the mean and covariance function of a GP can specify a distribution on a collection of function values. With this particular covariance function there are areas of high uncertainty where the distribution is very broad and areas of very low uncertainty, where the distribution collapses around a particular function value. However, the most important role of the covariance function is not to specify the *marginal* variance of a function value but rather to determine how two function values *covary*. The blue crosses in figure 1.3 show sampled function values, drawing each sample independently from the distribution at that particular value of $\boldsymbol{x}$. As there is no covariance specified between function values they appear just like Gaussian noise, albeit with a varying mean and variance, which is clearly not a very satisfactory distribution on functions. Furthermore, if we were to observe a particular value of the function at an input location then, without covariance, this would have no

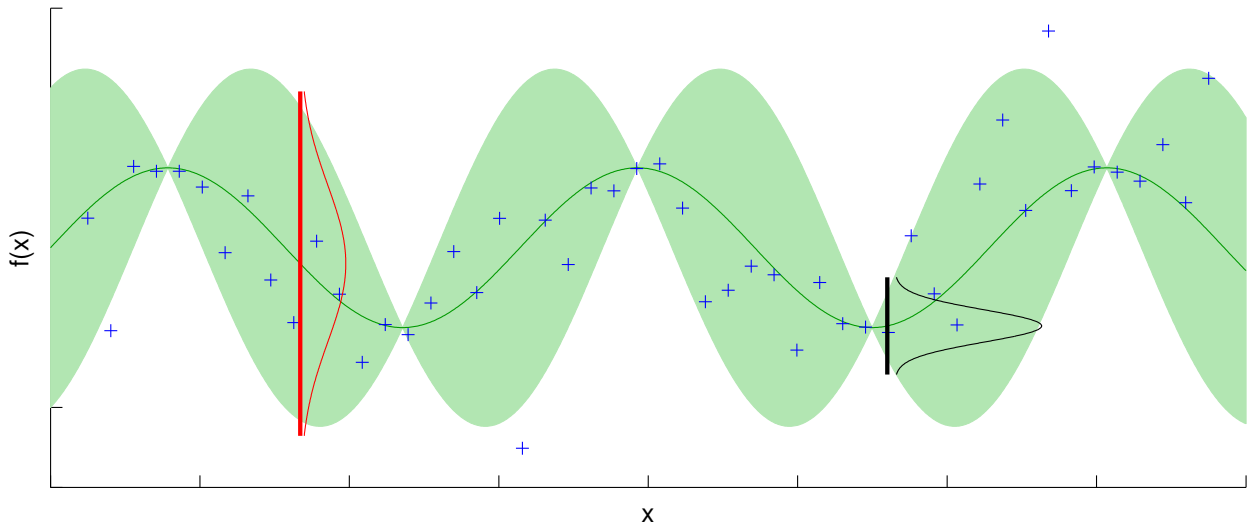


Figure 1.3: A Gaussian distribution on function values, with a mean function of $\sin(\boldsymbol{x})$ and a marginal variance function of $\cos^2(\boldsymbol{x})$. The solid green line represents the mean and the shaded green area shows two standard deviations either side of the mean. The marginal distribution over the function value at a particular value of $\boldsymbol{x}$ is Gaussian, as illustrated by the red and black vertical slices. The blue crosses are independently sampled function values from this distribution.

effect at all on any of the other function values. In other words, without covariance we cannot learn anything from any observed data. Thus specifying how function values covary is vitally important. The covariance function therefore takes two arguments, $k(\boldsymbol{x}_1, \boldsymbol{x}_2)$, and returns the covariance between their corresponding function values,

$$\mathbb{C}[f(\boldsymbol{x}_1),\ f(\boldsymbol{x}_2)] \ = \ k(\boldsymbol{x}_1, \boldsymbol{x}_2) \tag{1.2}$$

Notice that the covariance between function values is fully specified by the corresponding input locations, and not at all by the actual values of the function; this is a property of the Gaussian Process. We can now formulate the statement we made earlier, that in a GP all the function values are jointly Gaussian, by writing,

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \ \sim \ \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x}_1) \\ \vdots \\ m(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ & \ddots & \\ k(\boldsymbol{x}_1, \boldsymbol{x}_N) & & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right) \tag{1.3}$$

where $\boldsymbol{x}_1$ to $\boldsymbol{x}_N$ are a set of input locations. We will mostly use the shorthand,

$$\boldsymbol{f}(X) \ \sim \ \mathcal{N}(\boldsymbol{m}(X),\ K(X, X)) \tag{1.4}$$

or $K \ = \ K(X, X)$.

The most common choices for the covariance function k exhibit the stationarity property, that is the covariance between two function values $f(\boldsymbol{x}_1)$ and $f(\boldsymbol{x}_2)$ is purely dependent on the distance between their inputs and not the actual values of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. In these covariance functions it is usual for the covariance between two function values to increase the closer their corresponding inputs get to each other. This captures a simple intuition: if the inputs are close then the function outputs are also likely

to be close. This is clearly a sensible assumption for continuous functions. If we extend this argument then it is not hard to see how the shape of the covariance function can affect the smoothness of the resulting sampled functions — if the covariance decays slowly as a function of the distance between the input points, as it does in the 'squared exponential' covariance function, then the resulting functions are much smoother than if the covariance decays rapidly, as for example in the Matérn $\frac{3}{2}$ covariance function. We can also use the covariance function to encode properties such as periodicity by specifying that a function value covaries strongly with points which are separated by any multiple of the period. Examples of these three covariance functions are shown in figure 1.4, along with an example sampled function. These sampled functions can be drawn in the same way as one samples from a Gaussian distribution. That is, for an ith sample,

$$\boldsymbol{f}^i(X) \;=\; \boldsymbol{m}(X) \;+\; L\,\boldsymbol{r}^i \tag{1.5}$$

$$\text{where} \qquad L\,L^T \;=\; K(X,\,X), \qquad\qquad \boldsymbol{r}^i \;\sim\; \mathcal{N}(\boldsymbol{0},\; I) \tag{1.6}$$

Note that all three sampled functions in figure 1.4 use the same set of random values $\boldsymbol{r}$, the only difference is in the covariance between each value. The different covariance functions lead to very different functions: the 'squared exponential' (SE) covariance function gives rise to very smooth functions, whereas the Matérn produces very rough functions. Unsurprisingly, the periodic covariance function results in functions which are themselves periodic. Thus the choice of covariance function encodes high level properties into the distribution over functions, such as smoothness and periodicity, but does not have to constrict the functions to a particular class, e.g. polynomials. Indeed the model class is very broad indeed: for example a GP with squared exponential covariance function places some probability mass on every infinitely-smooth function. By way of comparison, this modelling power is equivalent to a Gaussian radial basis function (RBF) with an infinite number of basis functions. As figure 1.5 shows, this flexibility leads to an extremely broad range of the potential functions. The use of a GP within the Bayesian framework thus satisfies both the modelling requirements listed above.

1.4.2 Modelling data with Gaussian Processes

Figure 1.5 shows a GP prior on functions with the squared exponential covariance function. In this section we look at how we can incorporate observations and find the posterior distribution on function values. This can be thought of as finding the functions from the GP prior which agree 'most' with the observed data. Suppose that there is some unknown function $f(\boldsymbol{x})$, which maps a D-dimensional input to a scalar output value f. We have a set of N noisy observations $\{y_i\}_{i=1}^N$ corresponding to a set of N, D-dimensional inputs, $\{\boldsymbol{x}_i\}_{i=1}^N$. We shall denote these collections of variables as the $N \times D$ input matrix X and the $N \times 1$ output vector $\boldsymbol{y}$. Our goal is to find the posterior distribution on the unknown function values given the observed data, $p(\boldsymbol{f} \mid X, \boldsymbol{y})$ and using a GP prior on the function values,

$$p(\boldsymbol{f} \mid X) \;=\; \mathcal{N}\big(m(X),\; k(X,X)\big) \tag{1.7}$$

To do this we must first quantify the relationship between the unknown function value $f(\boldsymbol{x})$ and the observation y at the same input location. This relationship is usually problem-specific, but in this

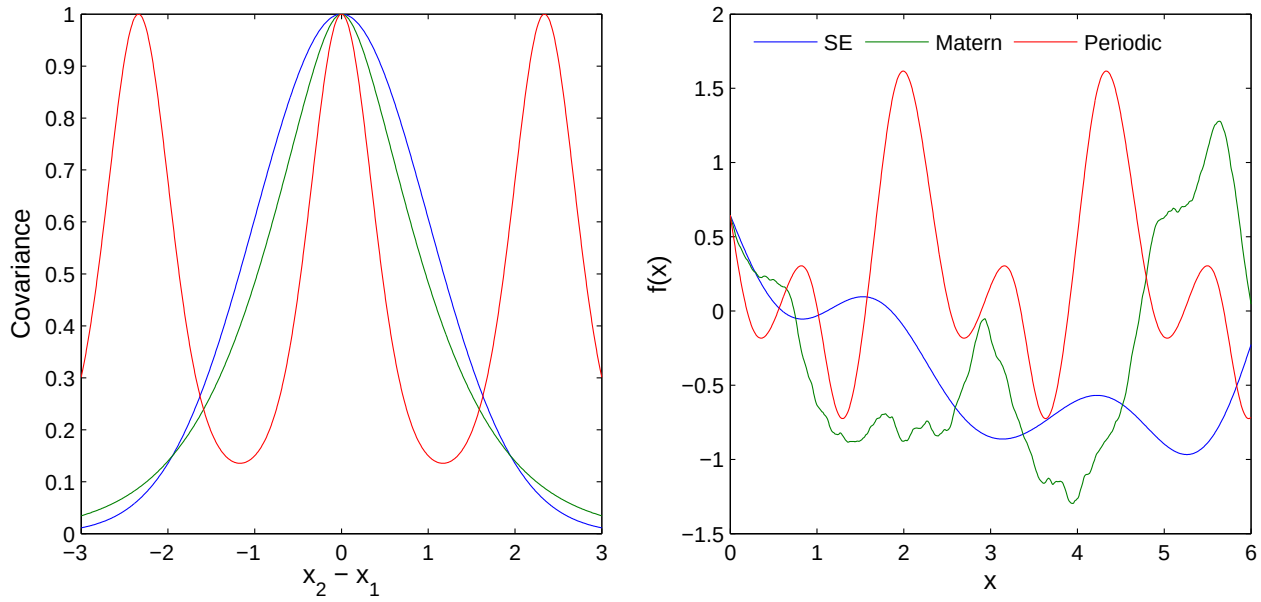


Figure 1.4: The left plot shows three examples of stationary covariance functions, the squared exponential (SE), Matérn 3/2, and the periodic covariance function. As they are stationary they only depend on the distance between the two points they are evaluated on, and this distance is used on the x-axis. A GP induces a distribution over function values and the right hand plot shows a set of sampled values for each of covariance functions.

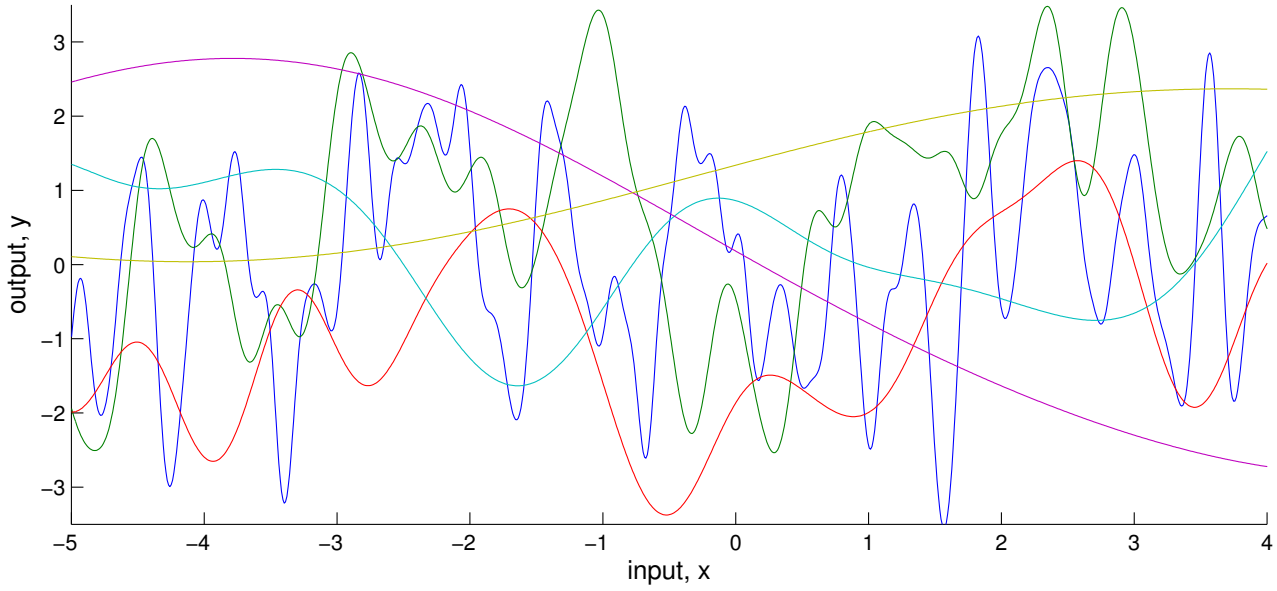


Figure 1.5: Example functions drawn from a GP prior with a squared exponential covariance function

thesis we will limit ourselves to the case of additive noise,

$$y \;=\; f(\boldsymbol{x}) \;+\; \epsilon \tag{1.8}$$

where ϵ is a noise variable. We will usually further assume (for reasons of tractabilitiy) that the noise is Gaussian and independent on each data point, for example,

$$\epsilon \;\sim\; \mathcal{N}(0, \sigma_\epsilon^2) \tag{1.9}$$

where Σ_ϵ is the unknown noise variance. With these modelling assumptions in place we can write the likelihood as,

$$p(\boldsymbol{y} \mid \boldsymbol{f}, X) \;=\; \prod_{i=1}^{N} \mathcal{N}(y_i;\; f_i,\; \sigma_\epsilon^2) \;=\; \mathcal{N}(\boldsymbol{y};\; \boldsymbol{f},\; \sigma_\epsilon^2\, I_N) \tag{1.10}$$

We can combine the prior (equation 1.7) and the likelihood (equation 1.10) to write the joint probability between $\boldsymbol{f}$ and $\boldsymbol{y}$ given X,

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{y} \end{bmatrix} \;\sim\; \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X) \end{bmatrix}, \begin{bmatrix} K & K \\ K & K \;+\; \sigma_\epsilon^2\, I_N \end{bmatrix} \right) \tag{1.11}$$

We can now apply the standard Gaussian conditioning equations to find that

$$p(\boldsymbol{f} \mid \boldsymbol{y},\, X) \;=\; \mathcal{N}\Big(m(X) + K\,[K \;+\; \sigma_\epsilon^2\, I_N]^{-1}\,(\boldsymbol{y} - m(X)), \quad K - K\,[K \;+\; \sigma_\epsilon^2\, I_N]^{-1} K \Big) \tag{1.12}$$

Of course we are not just interested in finding the posterior distribution on the function values at the points X where we have observed data, but also at a set of test points X^*. This is simple extension as, from the definition of a GP, the function values $\boldsymbol{f}^*$ at the test points X^* are also jointly Gaussian with the observations $\boldsymbol{y}$,

$$\begin{bmatrix} \boldsymbol{f}^* \\ \boldsymbol{y} \end{bmatrix} \;\sim\; \mathcal{N}\left(\begin{bmatrix} m(X^*) \\ m(X) \end{bmatrix}, \begin{bmatrix} K(X^*, X^*) & K(X^*,\, X) \\ K(X,\, X^*) & K \;+\; \sigma_\epsilon^2\, I_N \end{bmatrix} \right) \tag{1.13}$$

which leads to the standard GP predictive equations,

$$p(\boldsymbol{f}^* \mid X^*,\, X,\, \boldsymbol{y}) \;=\; \mathcal{N}(\mathfrak{m},\; \mathfrak{s}) \tag{1.14}$$

$$\mathfrak{m} \;=\; m(X^*) \;+\; \boldsymbol{k}(X^*,\, X)\left[K(X,\, X) \;+\; \sigma_\epsilon^2\, I_N\right]^{-1} \left(\boldsymbol{y} \;-\; m(X)\right) \tag{1.15}$$

$$\mathfrak{s} \;=\; k(X^*,\, X^*) \;-\; K(X^*,\, X)\left[K(X,\, X) \;+\; \sigma_\epsilon^2\, I_N\right]^{-1} K(X,\, X^*) \tag{1.16}$$

We will often abbreviate the inverse covariance matrix multiplied by the observations with $\boldsymbol{\beta}$,

$$\boldsymbol{\beta} \;=\; \left[K(X,\, X) \;+\; \sigma_\epsilon^2\, I_N\right]^{-1} \left(\boldsymbol{y} \;-\; m(X)\right) \tag{1.17}$$

Figure 1.6 shows a GP posterior on the same set of points as we looked at in figure 1.1. We can see how the uncertainty in the function value, represented by the shaded green area, shrinks as we approach an observation and grows as we move away from the data. Thus we have a model which can quantify and report its uncertainty.

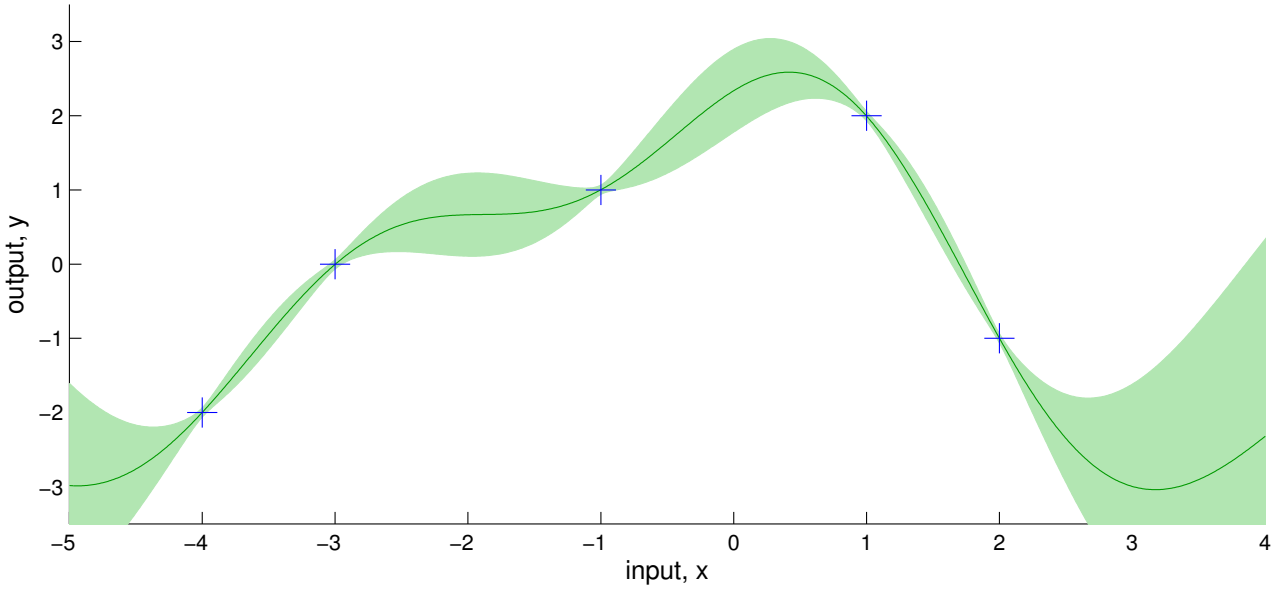


Figure 1.6: Example GP posterior using the same data as in figure 1.1 and using a squared exponential covariance function. The green line shows the mean of the posterior distribution and the green shaded area shows two standard deviations either side of the mean. Note how the uncertainty collapses around the data points and expands as we move away from the observations.

In figure 1.6 we specified that the mean function m is zero ($m(X) = 0$), in other words, before we observe any data points we expect the function to move symmetrically around zero. Figure 1.6 shows that in the region of the data the GP posterior moves significantly away from zero — the data is overriding the prior. However, as we move away from the data we fall back on the prior, which is why the posterior mean is returning to zero at the edges of the figure. Thus the GP mean function only has a strong effect away from the range of the data. Because of this, it is common to use the zero mean function unless we have reason to believe *a-priori* that the true function exhibits a certain parametric form, for example it might have a general linear rise, in which case we can use a linear mean function.

1.4.3 Learning in Gaussian Processes

In the previous section we discussed 'fitting' the GP to observed data. However, we didn't actually do any 'fitting' — no parameters were optimised. This is the Bayesian methodology: we specify our belief over any unknown variables by means of a prior and then we use the rules of probability to update our beliefs in the presence of observed data. The advantage of this approach is that there is no fear of overfitting, as we haven't optimised anything. The disadvantage is that the equations we need to solve to find the posterior are rarely analytically solvable. The great strength of the Gaussian Process is that we can find the posterior on the function values analytically (for the case of a Gaussian likelihood, as in equation 1.10). However, there are some additional unknown variables, which we have not yet discussed. These lie in the covariance, and potentially mean, functions.

The covariance functions used in figure 1.4 are the 'squared exponential' (SE) (a.k.a. the Gaussian or

'exponentiated quadratic' covariance function),

$$k_{SE}(\boldsymbol{x}_1, \boldsymbol{x}_2) \;=\; \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)\right) \tag{1.18}$$

the Matérn $\frac{3}{2}$,

$$k_{M32}(\boldsymbol{x}_1, \boldsymbol{x}_2) \;=\; \sigma_f^2 \,(1 + r) \exp(-r) \tag{1.19}$$

$$r \;=\; \sqrt{3(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)} \tag{1.20}$$

and the periodic covariance function, which first embeds the data points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ into a trigonometric feature space, and then applies a base covariance function k_0, which can be any of the other stationary covariance functions,

$$k_{per}(\boldsymbol{x}_1, \boldsymbol{x}_2) \;=\; k_0\big(u(\boldsymbol{x}_1), u(\boldsymbol{x}_2)\big) \tag{1.21}$$

$$u(\boldsymbol{x}) \;=\; \begin{bmatrix} \sin(\boldsymbol{a}) \\ \cos(\boldsymbol{a}) \end{bmatrix}, \qquad a_i \;=\; \frac{\pi x_i}{p_i} \quad \text{for } i \;=\; 1 \ldots D \tag{1.22}$$

The covariance functions listed above have a number of parameters, including a scale term σ_f^2, often referred to as the signal variance, and a set of lengthscales Λ, which determine the relevant scales in the input domain. The periodic function also has a period parameter p. These parameters are commonly referred to as 'hyperparameters' as they parameterise the distribution over functions rather than the functions themselves. We would like to treat these hyperparameters in a Bayesian manner — place a prior over them and then find the posterior distribution on the function values $\boldsymbol{f}$ whilst averaging over the hyperparameters $\boldsymbol{\theta}$,

$$p(\boldsymbol{f} \mid X, \boldsymbol{y}) \;=\; \int p(\boldsymbol{f} \mid X, \boldsymbol{y}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\; d\boldsymbol{\theta} \tag{1.23}$$

Unfortunately, due to the complex way in which these hyperparameters affect the distribution over $\boldsymbol{f}$, the integral in equation 1.23 is rarely solvable. We can either use an approximate inference method such as Markov Chain Monte Carlo or variational inference, or we can proceed via a non-Bayesian approach and use maximum (marginal) likelihood. In standard maximum likelihood we optimise the parameters to maximise the probability of the data under the model. This is often the easiest and quickest approach to fit a model in a probabilistic framework. However, it can suffer from problems with overfitting as, once again, we are only considering the 'best-fit' setting of the parameters and ignoring any uncertainty. In the GP hyperparameter setting overfitting isn't so much of a problem (although it can still occur) because by optimising the hyperparameters we are adapting the distribution over the functions, rather than picking the single best-fitting function as we would if we were optimising parameters. Recall that the likelihood is $p(\boldsymbol{y} \mid \boldsymbol{f}, X, \boldsymbol{\theta})$, which gives the probability of the observations $\boldsymbol{y}$ at input locations X given the function values $\boldsymbol{f}$. As we have just described, we don't want to pick a single set of function values $\boldsymbol{f}$, we want to average over the distribution on functions,

$$p(\boldsymbol{y} \mid X, \boldsymbol{\theta}) \;=\; \int p(\boldsymbol{y} \mid \boldsymbol{f}, X, \boldsymbol{\theta})\, p(\boldsymbol{f} \mid X, \boldsymbol{\theta})\; d\boldsymbol{f} \tag{1.24}$$

The distribution on $\boldsymbol{f}$ is Gaussian, from the definition of a GP, and thus if we also use a Gaussian likelihood, which we did here by assuming Gaussian noise (equation 1.10), then equation 1.24 is

analytically solvable. This allows us to write that the log marginal likelihood is,

$$\log p(\boldsymbol{y} \mid X, \boldsymbol{\theta}) \;=\; -\frac{1}{2}\left(\boldsymbol{y} - \boldsymbol{m}(X)\right)^T \left(K \;+\; \sigma_\epsilon^2\, I_N\right)^{-1} \left(\boldsymbol{y} - \boldsymbol{m}(X)\right) \;-\; \frac{1}{2}\left|K \;+\; \sigma_\epsilon^2\, I_N\right| \;-\; \frac{N}{2}\log 2\pi \tag{1.25}$$

We can easily take derivatives of the equation 1.25, w.r.t. the hyperparameters $\boldsymbol{\theta}$, which lie within the covariance matrix K and the mean vector $\boldsymbol{m}(X)$. In addition we can also differentiate w.r.t. the noise variance σ_ϵ^2, which allows us to optimise all these hyperparameters at the same time. For non-Gaussian likelihoods, equation 1.24 is not tractable and so approximate inference methods are often used, such as expectation propagation (Minka, 2001) or Laplace's method.

1.4.4 Predictions at uncertain inputs

The GP predictive moments in equations 1.15 and 1.16 give the mean and variance of the GP posterior at the known point $\boldsymbol{x}^*$. However, in this thesis we will often be interested in making predictions at uncertain inputs. This amounts to finding the average predictive distribution over the uncertainty in the input point,

$$p(f^* \mid \boldsymbol{y}, X) \;=\; \int p(f^* \mid \boldsymbol{x}^*, \boldsymbol{y}, X)\, p(\boldsymbol{x}^*)\; d\boldsymbol{x}^* \tag{1.26}$$

If the uncertain test point is Gaussian,

$$p(\boldsymbol{x}^*) \;=\; \mathcal{N}\left(\boldsymbol{\mu}^*,\; \Sigma^*\right) \tag{1.27}$$

then, although the predictive distribution is non-Gaussian (see figure 1.7), we can still compute its moments exactly for certain covariance functions. These have been derived in Girard et al. (2003) and Deisenroth (2009) and we summarise the results here. The predictive mean can be found by the rule of iterated expectations,

$$\begin{aligned}
\mathbb{E}\left[f^* \mid \boldsymbol{y}, X\right] \;&=\; \mathbb{E}_{\boldsymbol{x}^* \sim p(\boldsymbol{x}^*)}\left[\mathbb{E}_{f^* \sim p(f^* \mid \boldsymbol{x}^*,\, \boldsymbol{y},\, X)}\left[f^*\right]\right] \\
&=\; \mathbb{E}_{x^*}\left[m(\boldsymbol{x}^*)\right] \;+\; \mathbb{E}_{x^*}\left[k(\boldsymbol{x}^*, X)\right]\boldsymbol{\beta}
\end{aligned} \tag{1.28}$$

where $\boldsymbol{\beta}$ is defined in equation 1.17. The expectation over the covariance function can be solved for covariance functions such as the linear kernel and the squared exponential. We present the results for the squared exponential in Appendix A. The predictive variance can be found by use of the rule of total variance,

$$\begin{aligned}
\mathbb{V}\left[f^* \mid \boldsymbol{y}, X\right] \;&=\; \mathbb{V}_{\boldsymbol{x}^* \sim p(\boldsymbol{x}^*)}\left[\mathbb{E}_{f^* \sim p(f^* \mid \boldsymbol{x}^*,\, \boldsymbol{y},\, X)}\left[f^*\right]\right] \;+\; \mathbb{E}_{\boldsymbol{x}^* \sim p(\boldsymbol{x}^*)}\left[\mathbb{V}_{f^* \sim p(f^* \mid \boldsymbol{x}^*,\, \boldsymbol{y},\, X)}\left[f^*\right]\right] \\
&=\; V_{x^*}\left[m(\boldsymbol{x}^*)\right] \;+\; 2\,\mathbb{C}\left[m(\boldsymbol{x}^*), \boldsymbol{k}(\boldsymbol{x}^*, X)\right]\boldsymbol{\beta} \;+\; \boldsymbol{\beta}^T\, \mathbb{V}_{x^*}\left[k(\boldsymbol{x}^*, X)\right]\boldsymbol{\beta} \\
&\qquad +\; \mathbb{E}\left[k(\boldsymbol{x}^*, \boldsymbol{x}^*)\right] \;-\; \mathbb{E}\left[\boldsymbol{k}(\boldsymbol{x}^*, X)\left[K \;+\; \sigma_\epsilon^2\, I_N\right]^{-1} \boldsymbol{k}(X, \boldsymbol{x}^*)\right]
\end{aligned} \tag{1.29}$$

We can also find the covariance between the Gaussian test point $\boldsymbol{x}^*$ and the GP function value

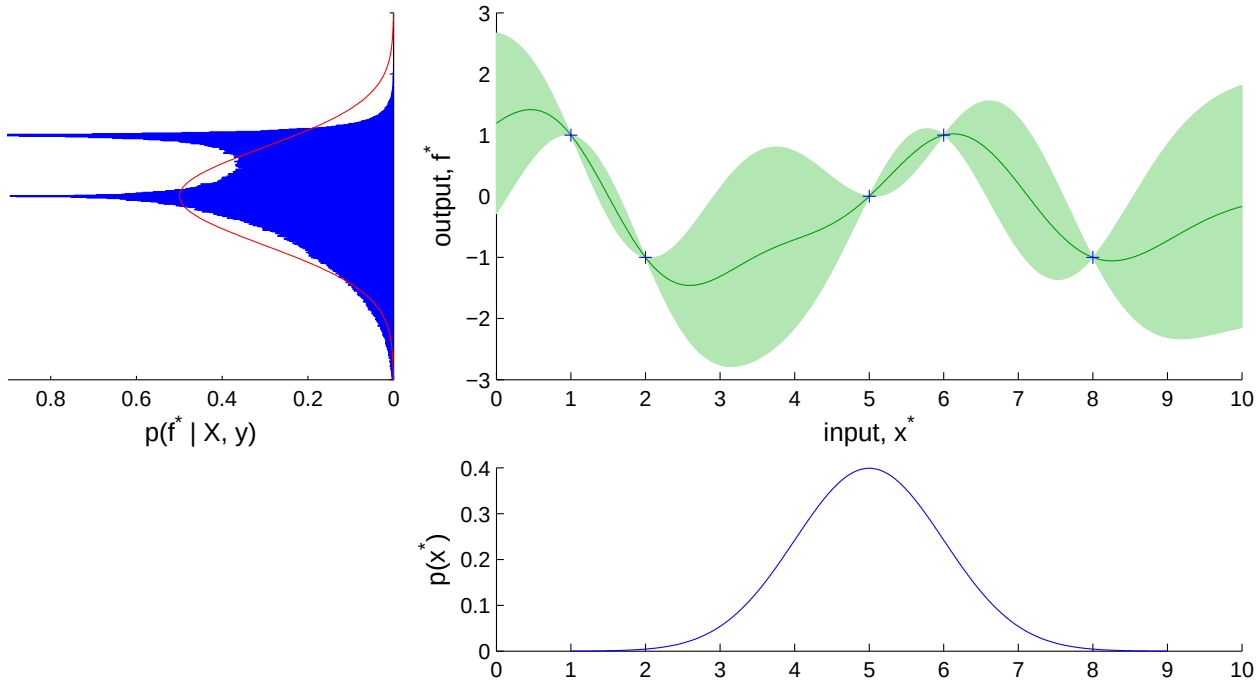


Figure 1.7: Making a GP prediction at a Gaussian test input. The top right plot shows a GP posterior based on the input-output data X, $\boldsymbol{y}$. We wish to make a prediction at the uncertain test input $\boldsymbol{x}^*$, distributed according to the Gaussian shown in the bottom plot. The blue histogram shows the true predictive distribution whilst the red Gaussian shows a moment-matched approximation.

$f^* = f(\boldsymbol{x}^*)$,

$$\begin{aligned} \mathbb{C}_{\boldsymbol{x}^*, f^*}\left[\boldsymbol{x}^*,\, f(\boldsymbol{x}^*)\right] &= \mathbb{E}_{f,x^*}\left[\boldsymbol{x}^*\, f(\boldsymbol{x}^*)\right] - \mathbb{E}_{x^*}\left[\boldsymbol{x}^*\right] \mathbb{E}_{f,x^*}\left[f(\boldsymbol{x}^*)\right] \\ &= \mathbb{E}_{x^*}\left[\boldsymbol{x}^*\, \mathbb{E}_f\left[f(\boldsymbol{x}^*)\right]\right] - \boldsymbol{\mu}^*\, \mathbb{E}_{f,x^*}\left[f(\boldsymbol{x}^*)\right] \\ &= \mathbb{E}_{x^*}\left[\boldsymbol{x}^*\, \boldsymbol{k}(\boldsymbol{x}^*, X)\right] \boldsymbol{\beta} - \boldsymbol{\mu}^*\, \mathbb{E}_{f,x^*}\left[f(\boldsymbol{x}^*)\right] \end{aligned} \tag{1.30}$$

where $\mathbb{E}_{x^*}\left[\boldsymbol{x}^*\, \boldsymbol{k}(\boldsymbol{x}^*, X)\right]$ is computed for the SE kernel in the appendix and $\mathbb{E}_{f,x^*}\left[f(\boldsymbol{x}^*)\right]$ is the term we computed above in equation 1.28. Equation 1.30 results in a $D \times 1$ vector.

1.4.5 Comparison to Bayesian feature-space regression

Related to Gaussian Processes is Bayesian feature-space regression: which is Bayesian linear regression where we first transform the input $\boldsymbol{x}$ into a feature vector $\boldsymbol{\phi}(\boldsymbol{x})$, via some (nonlinear) mapping. Here we provide a brief comparison. Whilst linear regression is very limited in the variety of functions it can fit, it can be made vastly more powerful by the simple extension of mapping the inputs to a feature-space. We then perform linear regression in this expanded space,

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} w_j\, \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{1.31}$$

where $\boldsymbol{w}$ is a vector of weights and $\phi_j(\boldsymbol{x})$ is the mapping from the input $\boldsymbol{x}$ to the jth feature. A popular choice for the features is to use a Gaussian radial basis function,

$$\phi_j(\boldsymbol{x}) \;=\; \exp\left(-\frac{1}{2}(\boldsymbol{x} \,-\, \boldsymbol{c}_j)^T \Lambda_j^{-1} (\boldsymbol{x} \,-\, \boldsymbol{c}_j)\right) \tag{1.32}$$

where the basis function has centre $\boldsymbol{c}_j$, and width Λ_j. Inference in functions of the form of equation 1.31 is simple if we only treat the weights in a Bayesian manner and optimise any parameters of the feature space mapping. For example,

$$p(w_j) \;=\; \mathcal{N}(0,\; \sigma_w^2) \tag{1.33}$$

$$\Rightarrow\; p(f \mid \boldsymbol{x},\, \boldsymbol{\theta}) \;=\; \mathcal{N}(0,\, \sigma_w^2\, \boldsymbol{\phi}^T \boldsymbol{\phi}) \tag{1.34}$$

If we have a set of training inputs X and outputs $\boldsymbol{y}$ then, by applying a similar set of steps to the GP derivations in section 1.4.2, we can find the predictive posterior distribution at a test point $\boldsymbol{x}^*$ to be,

$$p(f^* \mid \boldsymbol{x}^*,\, X,\, \boldsymbol{y}) \;=\; \mathcal{N}\left(\sigma_\epsilon^{-2}\, \boldsymbol{\phi}(\boldsymbol{x}^*)^T [\sigma_\epsilon^{-2}\, \Phi^T \Phi \,+\, \sigma_w^2\, I_M]^{-1} \Phi^T\, \boldsymbol{y},\quad \phi(\boldsymbol{x}^*)^T [\sigma_\epsilon^{-2}\, \Phi^T \Phi \,+\, \sigma_w^2\, I_M]^{-1} \phi(\boldsymbol{x}^*)\right) \tag{1.35}$$

with $\Phi_{ij} \;=\; \phi_j(\boldsymbol{x}_i)$, a $N \times M$ matrix. Figure 1.8 shows an example fit to the same five points as in figures 1.1 and 1.6 using a Bayesian Gaussin RBF with ten fixed basis functions, as shown in red. If we compare this plot to the GP posterior in figure 1.6 then we see that within the range of the basis functions the posteriors look very similar. However, outside this range the variance of the RBF drops away to zero, which is the exact opposite of what we would desire. There are two ways in which this problem can be fixed, either the input space must be completely tiled with basis functions, or we must also treat the centres and widths of the basis functions probabilistically, with appropriate priors. Taking the second approach causes the inference to no longer be analytically tractable, although Barber and Schottky (1998) discuss some suitable approximate inference methods. Tiling the complete input space with basis functions sounds completely farcical. Surprisingly, however, this is in effect exactly what the Gaussian Process does, by means of the 'kernel trick'. Gaussian Processes can, therefore, be viewed as the extension of the Bayesian feature-space regression to the case where we have infinitely many features. For a more in depth discussion of these concepts, see MacKay (1998) and Rasmussen and Williams (2006).

There are further difficulties with Bayesian feature-space regression, namely in how to choose the number and form of the features. A RBF format is popular, due to the universal approximation property, but this still leaves the question of which type of basis functions to use and how many should be included. Gaussian Processes remove the need to choose the number of RBFs and elevate the choice of basis function type to a choice of covariance functions, which are often easier to interpret. Due to their nonparametric nature, GPs adjust their complexity to the number of observed data points and are not limited to a predetermined complexity as RBFs are. The price paid for these advantages is in terms of computational time, as GPs can scale more poorly with the data than an RBF. Although this is only true for the case where the number of data points is greater than the number of basis functions. If we need a very large number of basis functions so as to tile the space with fine enough precision then GPs may even be faster. We believe that these many advantages strongly motivate the use of Gaussian Processes in modelling tasks.

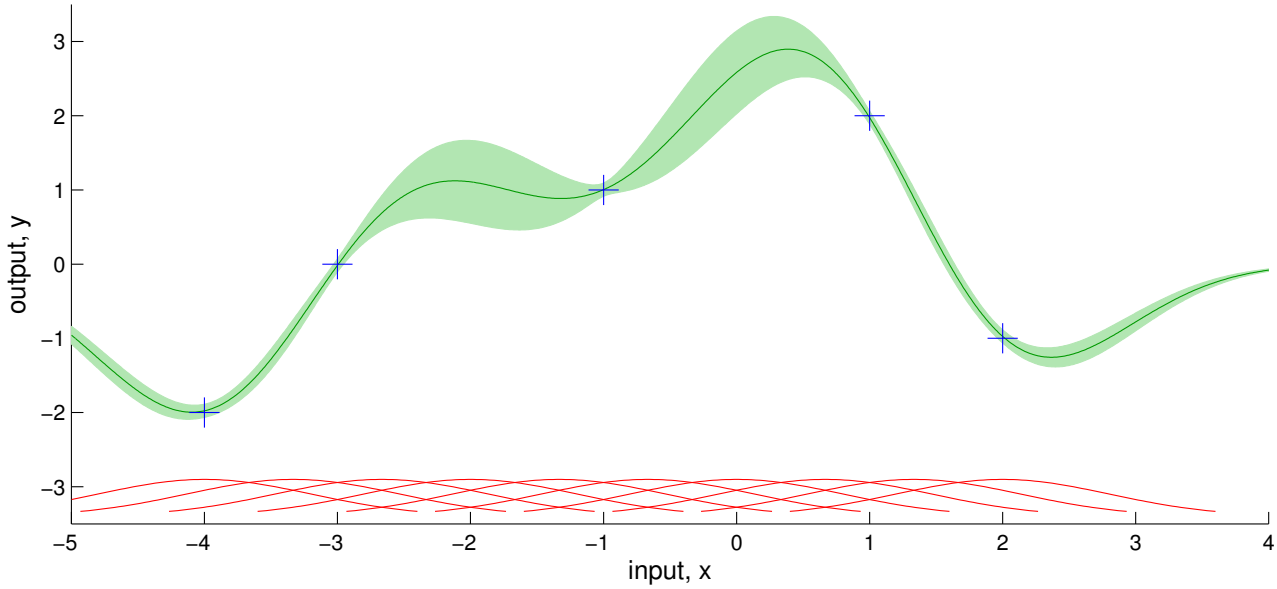


Figure 1.8: Illustration of a Bayesian RBF fitted to the data shown previously. Ten Gaussian basis functions where used as indicated by the red lines at the bottom of the plot. The basis function centres and widths were fixed and the weights were integrated out. The resulting posterior is shown in green.

1.5 Variational Bayesian Inference

We will use variational methods (Attias, 1999; Jordan et al., 1999) for approximate Bayesian inference in chapters 2 and 3. Therefore, we provide a brief introduction to variational methods here. Using Bayes' rule we can write the marginal likelihood of our observed data $\boldsymbol{y}$ in terms of our latent variables $\boldsymbol{x}$,

$$p(\boldsymbol{y}) \;=\; \frac{p(\boldsymbol{y} \mid \boldsymbol{x})\, p(\boldsymbol{x})}{p(\boldsymbol{x} \mid \boldsymbol{y})} \tag{1.36}$$

We can then multiply and divide by an arbitrary distribution $q(\boldsymbol{x})$ and take logarithms,

$$p(\boldsymbol{y}) \;=\; \frac{p(\boldsymbol{y} \mid \boldsymbol{x})\, p(\boldsymbol{x})}{q(\boldsymbol{x})} \frac{q(\boldsymbol{x})}{p(\boldsymbol{x} \mid \boldsymbol{y})} \tag{1.37}$$

$$\Rightarrow \log p(\boldsymbol{y}) \;=\; \log \frac{p(\boldsymbol{y} \mid \boldsymbol{x})\, p(\boldsymbol{x})}{q(\boldsymbol{x})} \;+\; \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x} \mid \boldsymbol{y})} \tag{1.38}$$

Finally we can take the expectation of both sides w.r.t. $q(\boldsymbol{x})$. As the left hand side has no dependence on $\boldsymbol{x}$ this step only has an effect on the right hand side of the equation,

$$\log p(\boldsymbol{y}) \;=\; \int q(\boldsymbol{x}) \log \frac{p(\boldsymbol{y} \mid \boldsymbol{x})\, p(\boldsymbol{x})}{q(\boldsymbol{x})}\, d\boldsymbol{x} \;+\; \int q(\boldsymbol{x}) \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x} \mid \boldsymbol{y})}\, d\boldsymbol{x} \tag{1.39}$$

We can recognise that the last term in equation 1.39 is the KL divergence between the arbitrary distribution $q(\boldsymbol{x})$ and the true posterior on the latent variables, $p(\boldsymbol{x} \mid \boldsymbol{y})$. As this term is always positive the log marginal likelihood must be lower bounded by first integral term,

$$\log p(\boldsymbol{y}) \;\geq\; \int q(\boldsymbol{x}) \log \frac{p(\boldsymbol{y} \mid \boldsymbol{x})\, p(\boldsymbol{x})}{q(\boldsymbol{x})}\, d\boldsymbol{x} \;\triangleq\; \mathcal{L}\big(q(\boldsymbol{x})\big) \tag{1.40}$$

The inequality in equation 1.40 is true for any distribution $q(x)$ although clearly if $q(\boldsymbol{x})$ is very far from the true posterior $p(\boldsymbol{x} \mid \boldsymbol{y})$ the KL divergence term will be large and the lower bound will be very loose. Equation 1.40 will reach equality if and only if $q(\boldsymbol{x}) = p(\boldsymbol{x} \mid \boldsymbol{y})$, however we assume that we cannot solve the equation for this situation. Thus the variational approach involves finding a $q(\boldsymbol{x})$ that makes equation 1.40 tractable whilst also minimising the KL divergence between $q(\boldsymbol{x})$ and $p(\boldsymbol{x} \mid \boldsymbol{y})$. This is usually achieved by assuming a particular form and/or independence structure for $q(\boldsymbol{x})$ and then using calculus of variations (hence the name) to find the optimal $q(\boldsymbol{x})$ subject to those restrictions.

1.6 General Notation

This section contains some notation which will be standard across all chapters.

Scalars, vectors, and matrices

x	Scalars are usually represented by a lower-case character
$\boldsymbol{x}$	Vectors are usually represented by a bold lower-case character
X	Matrices are usually represented by an upper-case character

Common terms

$\boldsymbol{x}$	a D-dimensional column vector, usually representing a noise-free state
$\boldsymbol{y}$	a column vector of observations
X	a collection of N $\boldsymbol{x}$ variables, arranged into a $N \times D$ matrix
$k(\boldsymbol{x}_1, \boldsymbol{x}_2)$	the covariance function k evaluated between points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$
$\boldsymbol{k}(X, \boldsymbol{x}^*)$ or $\boldsymbol{k}(\boldsymbol{x})$	a N-dimensional column vector, where the ith entry is $k(\boldsymbol{x}_i, \boldsymbol{x}^*)$
$\boldsymbol{k}(\boldsymbol{x}^*, X)$	a N-dimensional row vector, where the ith entry is $k(\boldsymbol{x}^*, \boldsymbol{x}_i)$
$K(X, X)$ or K	a $N \times N$ matrix, where the i, jth entry is $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
$\boldsymbol{\beta}$	a N-dimensional column vector defined in equation 1.17
$\mathfrak{m}$	the GP predictive mean, equation 1.15
$\mathfrak{s}$	the GP predictive variance, equation 1.16

Probability

$p(\boldsymbol{x})$	the probability density function on $\boldsymbol{x}$
$q(\boldsymbol{x})$	usually implies an approximate probability density function on $\boldsymbol{x}$
$\boldsymbol{x} \sim p(\boldsymbol{x})$	Implies that $\boldsymbol{x}$ is distributed according to $p(\boldsymbol{x})$
$\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[\boldsymbol{x}]$	The expectation of $\boldsymbol{x}$ over the distribution $p(\boldsymbol{x})$
$\mathbb{V}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[\boldsymbol{x}]$	The variance of $\boldsymbol{x}$ over the distribution $p(\boldsymbol{x})$
$\mathbb{C}[\boldsymbol{x}, \boldsymbol{y}]$	The covariance between $\boldsymbol{x}$ and $\boldsymbol{y}$
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$	The Gaussian (normal) probability density function, with mean $\boldsymbol{\mu}$ and variance Σ

1.7 Test systems

In this thesis we are particularly focussed on the modelling of dynamical systems. These systems are pervasive in our lives, and thus modelling them, which is a prerequisite for making predictions (chapter 3) and designing controllers (chapter 4), is an incredibly important area for research. Throughout this

thesis, we shall test various modelling strategies on two benchmark dynamical systems: a cart & pendulum, and a robotic unicycle. We describe these two systems next.

1.7.1 Cart and Pendulum System

The first dynamical system we introduce is the four dimensional cart and pendulum, as shown in figure 1.9. This consists of a cart, which is free to move side-to-side under the action of a force. An unactuated, free-to-rotate pendulum is attached to the centre of the cart, as shown in the figure. The system state vector is,

$$\boldsymbol{x} \;=\; \left[x,\; \dot{x},\; \theta,\; \dot{\theta}\right]^T \tag{1.41}$$

and there is a single control variable: the force applied to the cart. The (idealised) equations of motion can be derived fairly simply from Newtonian mechanics, e.g. see Hall (2013). There are two control tasks which we will consider on the cart and pendulum:

- **The balancing task** In this task the pendulum starts in the inverted position and the goal is to maintain it at this unstable equilibrium point. This is an extremely well studied problem and it can be straightforwardly solved using linearised dynamics.
- **The swing-up task** In this task the pendulum starts hanging downwards and the controller must move the cart so as to swing the pendulum up into the inverted position and then hold it there. This is a considerably harder problem than the balancing task. It cannot be solved with a linear model as the pendulum will move through at least 180 degrees. Because of this, the swing-up task has been used to test various nonlinear control strategies (Åström and Furuta, 2000; Furuta et al., 1991).

1.7.2 Unicycle System

The ten dimensional unicycle system is considerably more complex than the cart and pendulum. It consists of a body mounted above a single wheel, and with a flywheel set horizontally above it, as shown in figure 1.10. The wheel and flywheel can be driven by a motor allowing the unicycle to move in the forwards and backwards directions, and to rotate on the spot. We represent the unicycle by the ten dimensional state,

$$\boldsymbol{x} \;=\; \left[\dot{\phi},\; \dot{\theta},\; \dot{\psi},\; \dot{\phi}_w,\; \dot{\phi}_t,\; x_c,\; y_c,\; \phi,\; \theta,\; \psi\right] \tag{1.42}$$

where we have ignored the angle of the wheel and the flywheel as these are assumed to have negligible effect on the dynamics. The task is to balance the unicycle in the upright position, whilst minimising its distance from the origin. If the unicycle starts to fall in pitch it can be corrected by moving forwards or backwards as required. However, if the unicycle starts to fall in the roll direction there is no direct control action it can take to remedy this situation: it must first use the flywheel to turn into the fall, replacing and angle in roll with an angle in pitch, which it can then correct by moving forwards or backwards. The task would be made much simpler if the flywheel where to be mounted vertically, that is rotated ninety degrees in pitch (as defined in figure 1.10). In that situation a torque could be generated to directly counteract a fall in roll by spinning the flywheel (this is similar to a human rider

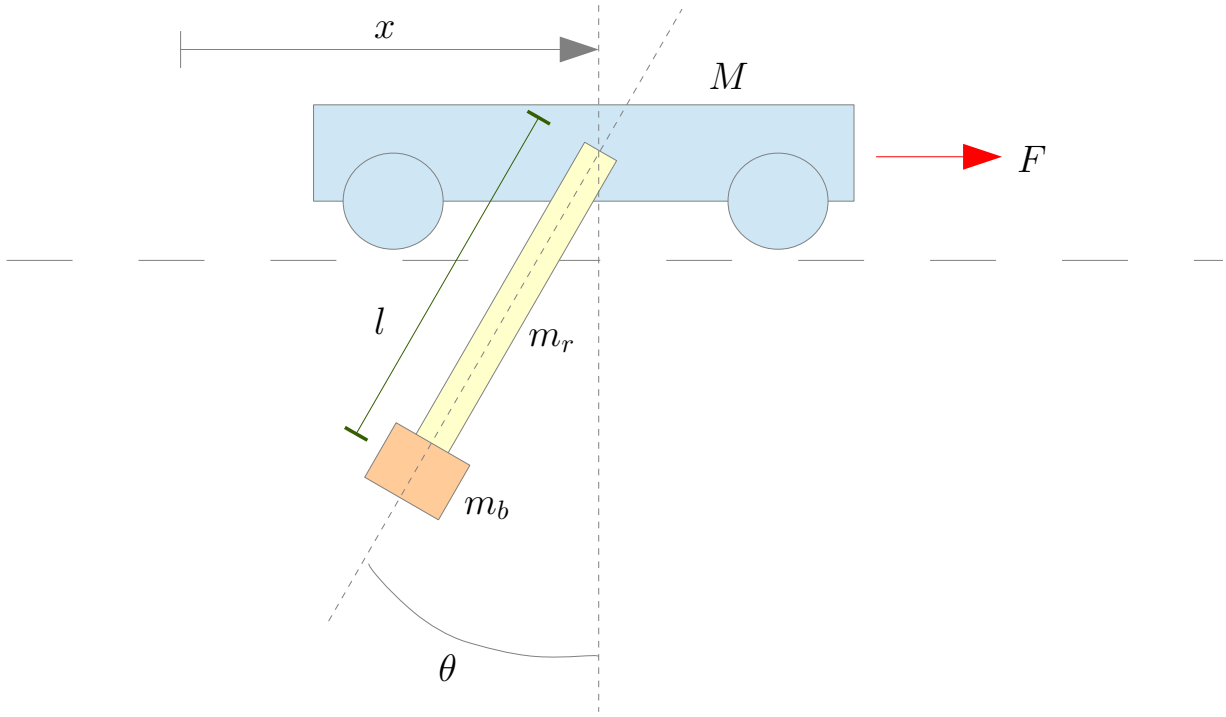


Figure 1.9: The cart and pendulum system. A cart of mass M is free to move in one dimension under the action of a force F. Attached to the centre of the cart is a pendulum, consisting of a rod of mass m_r and length l, and a bob of mass m_b. The pendulum is unactuated and free to rotate about its attachment. The distance of the cart from the origin in denoted by x and the angle of the pendulum by θ.

leaning to one side). Here we will concentrate on the much more challenging task of stabilising the unicycle with a horizontally mounted flywheel.

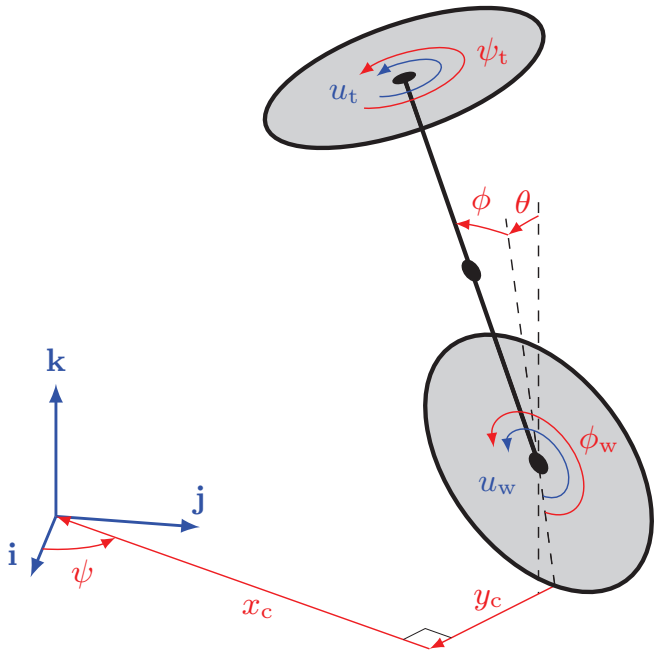


Figure 1.10: Idealised diagram of the unicycle. The state is described by the pitch ϕ, roll θ, and yaw ψ angles, the angles of the wheel ϕ_w and flywheel ψ_t, and their corresponding time derivatives. In addition there are two position variables, which map the position of the origin w.r.t. the unicycle (self-centred coordinates) x_c and y_c. Figure reproduced from Hall (2013).

Chapter 2

Gaussian Processes with Input Noise

2.1 Chapter Overview

In this chapter we look at the problem of regression using Gaussian Processes for the case where the input points are corrupted with noise (a situation sometimes termed *errors-in-the-variables*). In such cases Gaussian Process regression is no longer tractable and so a number of approximate methods have been developed. In section 2.2 we introduce the problem and outline some of the approaches taken to perform GP regression with input noise. In section 2.3 we demonstrate how to apply the work of Titsias and Lawrence (2010) to find a variational approach to the problem; then in section 2.4 we present a fast approach, Noisy Input Gaussian Process (NIGP) (McHutchon and Rasmussen, 2011), based on the Taylor series expansion of the GP posterior. Finally we compare the various methods and present our conclusions. We see that a variational approach tends to underfit but that NIGP produces strong results. As an appendical section, we present mathematical results on the derivatives of Gaussian Processes in section 2.7, which are needed by the NIGP model.

The main contribution of this chapter lies in the NIGP model published in NIPS 2011. We also apply the variational approach proposed by Titsias and Lawrence (2010) to GP regression with input noise and compare the results.

2.2 Introduction

Over the last decade the use of Gaussian Processes (GPs) as non-parametric regression models has grown significantly. They have been successfully used to learn mappings between inputs and outputs in a wide variety of tasks. However, many authors have highlighted a limitation in the way GPs handle noisy measurements. Standard GP regression (Rasmussen and Williams, 2006) makes two assumptions about the noise in datasets: firstly that measurements of input points, $\boldsymbol{y}_{\text{in}}$, are noise-free, and, secondly, that output points, y_{out}, are corrupted by constant-variance noise. For some datasets this makes intuitive sense: for example, an application in Rasmussen and Williams (2006) is that of modelling CO_2 concentration in the atmosphere over the last forty years. One can viably assume that the date is available noise-free and the CO_2 sensors are affected by signal-independent sensor noise. However, in many datasets, either or both of these assumptions are not valid and proceeding as if they are leads to poor modelling performance. In this chapter we look at datasets where the input

measurements $\boldsymbol{y}_{\text{in}}$, as well as the output y_{out}, are corrupted by noise. As figure 2.1 illustrates, input and output noise can have very different effects on observed values from a regression point of view, where we are interested in the outputs as a function of the inputs; hence input noise needs to be specifically accounted for by regression methodologies.

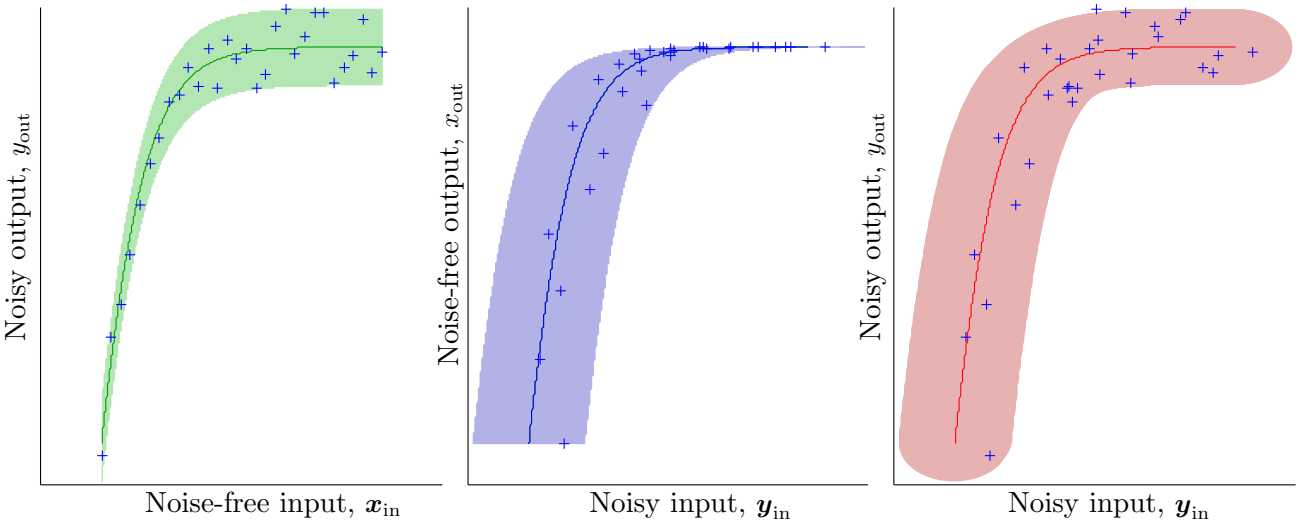


Figure 2.1: Illustration of the effects of output (left) and input (middle) noise in regression. In each of the three plots the solid line shows the true function from input (x-axis) to output (y-axis). The shaded regions show two standard deviations of noise (here Gaussian), indicating where observed points may lie; the blue crosses show sample observations. The left plot shows pure output noise, the middle pure input noise, and the right plot shows both input and output noise. In all cases the noise has constant variance across the input space.

Figure 2.2 shows the graphical model for regression with input and output noise. We assume that there exists a mapping f from the hidden, noise-free inputs, $\boldsymbol{x}_{\text{in}}$ to the noise-free outputs x_{out}. In regression we attempt to model this mapping function f by using a set of N paired input-output observations, $\{\boldsymbol{y}_{\text{in}}^i,\ y_{\text{out}}^i\}_{i=1}^N$. As we have just stated, the standard Gaussian Process regression approach is to assume that the input noise, $\boldsymbol{\epsilon}_{\text{in}}$ is zero. The result of this assumption is that the GP attempts to model the function g in figure 2.2, rather than the function f. However, if $\boldsymbol{\epsilon}_{\text{in}}$ is actually non-zero, the presence of random noise in $\boldsymbol{y}_{\text{in}}$ means that there is, most likely, no function g which satisfies the mapping $\boldsymbol{y}_{\text{in}}$ to x_{out}. It is not surprising therefore, that a Gaussian Process struggles to model a system with input noise. Figure 2.3 shows some input-output pairings taken from a simple quadratic function with additive, constant-variance noise on both inputs and outputs. If we take the GP approach and assume that there is only output noise then we will greatly underestimate the noise on some points and overestimate it on others. If we are using a Gaussian noise model for tractability then the light-tails nature of a Gaussian distribution (there is relatively little probability mass in the tails compared to the central part of the distribution) commonly results in the GP greatly overestimating the total amount of noise in the system. This is because the points where noise is overestimated have a much stronger effect than the points where there is an underestimate of the noise.

We now attempt to derive the GP posterior for a system with input noise. Let X_{in} be a $N \times D$ matrix of noise-free training inputs, X_{out} a $N \times 1$ vector of noise-free training targets, and Y_{in}, Y_{out} be the

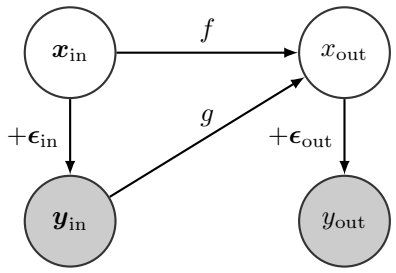


Figure 2.2: Graphical model of regression with input and output noise.

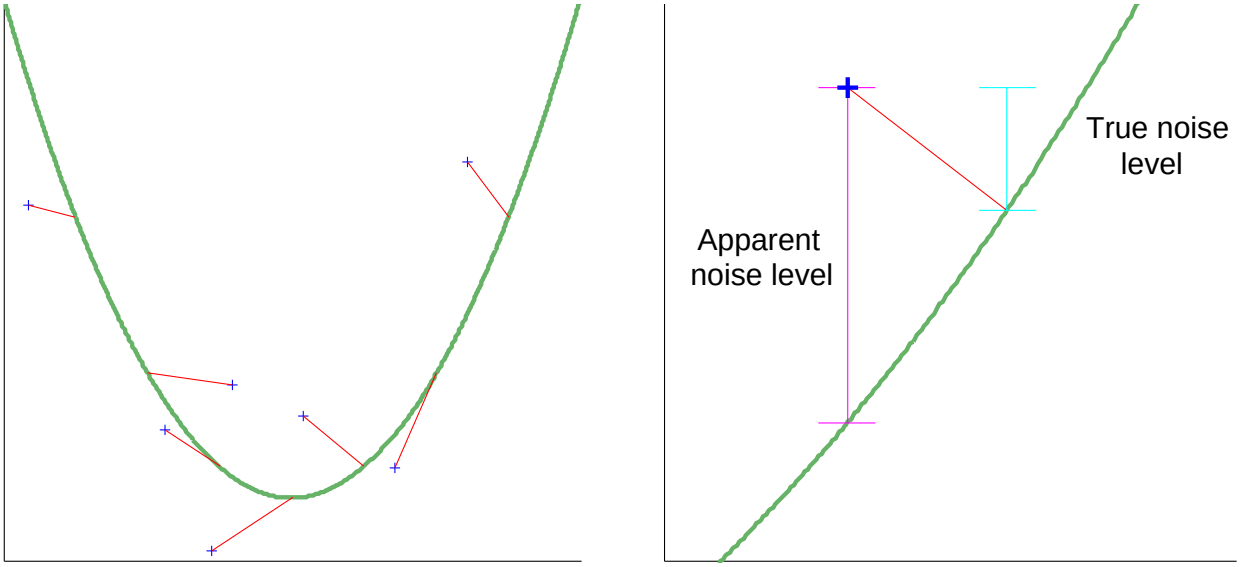


Figure 2.3: Input noise, if not taken into account, can lead regression methods to overestimate the amount of output noise present. The plot on the left shows the true function in green along with a number of observed values (blue crosses) corrupted by input and output noise. The red lines show where the corresponding noise-free points lie. The right hand plot shows a point in closer detail. The true amount of output noise corrupting the observation is shown with the cyan error bar on the right. However, as the observation has also been corrupted in the input dimension it appears to have a much larger output noise — larger than both the true input and true output noise levels combined.

corresponding noisy versions. For the sake of this argument we will assume Gaussian noise,

$$\begin{aligned} \boldsymbol{y}_{\text{in}}^i &= \boldsymbol{x}_{\text{in}}^i + \boldsymbol{\epsilon}_{\text{in}}^i, \qquad \boldsymbol{\epsilon}_{\text{in}}^i \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{in}}) \\ y_{\text{out}}^i &= x_{\text{out}}^i + \epsilon_{\text{out}}^i, \qquad \epsilon_{\text{out}}^i \sim \mathcal{N}(0, \sigma_{\text{out}}^2) \end{aligned} \tag{2.1}$$

Recall from section 1.4.2 that the equations for a GP prediction at a noise-free test input $\boldsymbol{x}_{\text{in}}$ are,

$$\begin{aligned} p(x_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta}) &= \mathcal{N}(x_{\text{out}}; \mathfrak{m}, \mathfrak{s}) \\ \mathfrak{m} &= \boldsymbol{k}(\boldsymbol{x}_{\text{in}}, X_{\text{in}}) \left[K(X_{\text{in}}, X_{\text{in}}) + \sigma_{\text{out}}^2 I\right]^{-1} X_{\text{out}} \\ \mathfrak{s} &= k(\boldsymbol{x}_{\text{in}}, \boldsymbol{x}_{\text{in}}) - \boldsymbol{k}(\boldsymbol{x}_{\text{in}}, X_{\text{in}}) \left[K(X_{\text{in}}, X_{\text{in}}) + \sigma_{\text{out}}^2 I\right]^{-1} \boldsymbol{k}(X_{\text{in}}, \boldsymbol{x}_{\text{in}}) \end{aligned} \tag{2.2}$$

In the case we are considering in this chapter, we do not have access to either the latent test input $\boldsymbol{x}_{\text{in}}$ or the latent training inputs X_{in}, only to the noisy versions, $\boldsymbol{y}_{\text{in}}$ and Y_{in}. We therefore need to integrate both of these quantities out, as well as integrating out X_{out} as is usually done in GP regression. We

start from the joint and expand it according to the graphical model in figure 2.2,

$$p(\boldsymbol{y}_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{x}_{\text{in}}, x_{\text{out}}, X_{\text{in}}, X_{\text{out}} \mid \boldsymbol{\theta}) = p(\boldsymbol{y}_{\text{in}}, Y_{\text{in}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(x_{\text{out}}, X_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(Y_{\text{out}} \mid X_{\text{out}})\, p(\boldsymbol{x}_{\text{in}}, X_{\text{in}}) \tag{2.3}$$

$$p(x_{\text{out}}, X_{\text{out}}, \boldsymbol{x}_{\text{in}}, X_{\text{in}} \mid \boldsymbol{y}_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{y}_{\text{in}}, Y_{\text{in}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(x_{\text{out}}, X_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(Y_{\text{out}} \mid X_{\text{out}})\, p(\boldsymbol{x}_{\text{in}}, X_{\text{in}})}{\int p(\boldsymbol{y}_{\text{in}}, Y_{\text{in}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(x_{\text{out}}, X_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(Y_{\text{out}} \mid X_{\text{out}})\, p(\boldsymbol{x}_{\text{in}}, X_{\text{in}})\; dx_{\text{out}}\; dX_{\text{out}}\; d\boldsymbol{x}_{\text{in}}\; dX_{\text{in}}} \tag{2.4}$$

$$p(x_{\text{out}} \mid \boldsymbol{y}_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta}) = \frac{\int p(\boldsymbol{y}_{\text{in}}, Y_{\text{in}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(x_{\text{out}}, X_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(Y_{\text{out}} \mid X_{\text{out}})\, p(\boldsymbol{x}_{\text{in}}, X_{\text{in}})\; dX_{\text{out}}\; d\boldsymbol{x}_{\text{in}}\; dX_{\text{in}}}{\int p(\boldsymbol{y}_{\text{in}}, Y_{\text{in}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(x_{\text{out}}, X_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}})\, p(Y_{\text{out}} \mid X_{\text{out}})\, p(\boldsymbol{x}_{\text{in}}, X_{\text{in}})\; dx_{\text{out}}\; dX_{\text{out}}\; d\boldsymbol{x}_{\text{in}}\; dX_{\text{in}}} \tag{2.5}$$

The GP prior implies,

$$p(x_{\text{out}}, X_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, X_{\text{in}}) = \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x}_{\text{in}}) \\ m(X_{\text{in}}) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_{\text{in}}, \boldsymbol{x}_{\text{in}}) & k(\boldsymbol{x}_{\text{in}}, X_{\text{in}}) \\ k(X_{\text{in}}, \boldsymbol{x}_{\text{in}}) & k(X_{\text{in}}, X_{\text{in}}) \end{bmatrix} \right) \tag{2.6}$$

Given that the covariance function k is nearly always a nonlinear function, integrating out the latent inputs, $\boldsymbol{x}_{\text{in}}$, and X_{in} will result in a non-Gaussian distribution over the latent outputs, for all but the most trivial of priors, $p(\boldsymbol{x}_{\text{in}}, X_{\text{in}})$. THis means we cannot integrate out the latent outputs analytically. Thus, exact inference in the GP framework with each input location as a distribution is intractable. One could design a sampling scheme (e.g. MCMC) to compute the posteriors on both the hyperparameters and the prediction approximately. However, such schemes are not straightforward to implement and are likely to be computationally very expensive. They can also be awkward to make predictions with, if it is necessary to run sampling for every prediction. The alternative approach is to use an approximate-analytic method. A number of authors have attempted to tackle this problem and we now look at two approaches: using the expected covariance matrix (section 2.2.1), and via use of the Taylor series (section 2.2.2).

2.2.1 The Expected Covariance Matrix Approach

One approximate solution presented in Dallaire et al. (2008) is to replace the training point covariance matrix K in equation 2.2 with the expected covariance matrix,

$$\tilde{K} = \int K(X_{\text{in}}, X_{\text{in}})\, p(X_{\text{in}} \mid Y_{\text{in}})\; dX_{\text{in}} \tag{2.7}$$

As we stated above regarding $\boldsymbol{x}_{\text{in}}$, for the squared exponential covariance function and a Gaussian distribution on X_{in} this integral is solvable. Assuming independent noise on the training inputs,

$$\begin{aligned}\tilde{K}_{ij} &= \int k(\boldsymbol{x}_{\text{in}}^i, \boldsymbol{x}_{\text{in}}^j)\, p(\boldsymbol{x}_{\text{in}}^i \mid \boldsymbol{y}_{\text{in}}^i)\, p(\boldsymbol{x}_{\text{in}}^j \mid \boldsymbol{y}_{\text{in}}^j)\; d\boldsymbol{x}_{\text{in}}^i\; d\boldsymbol{x}_{\text{in}}^j \qquad (2.8)\\ &= \int \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_{\text{in}}^i - \boldsymbol{x}_{\text{in}}^j)^T \Lambda^{-1} (\boldsymbol{x}_{\text{in}}^i - \boldsymbol{x}_{\text{in}}^j)\right) \mathcal{N}(\boldsymbol{x}_{\text{in}}^i;\; \boldsymbol{y}_{\text{in}}^i,\, \Sigma_{\text{in}})\, \mathcal{N}(\boldsymbol{x}_{\text{in}}^j;\; \boldsymbol{y}_{\text{in}}^j,\, \Sigma_{\text{in}})\; d\boldsymbol{x}_{\text{in}}^i\; d\boldsymbol{x}_{\text{in}}^j \\ &= \left|2\Lambda^{-1}\Sigma_{\text{in}} + I\right|^{-1/2} \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{y}_{\text{in}}^i - \boldsymbol{y}_{\text{in}}^j)^T (\Lambda + 2\Sigma_{\text{in}})^{-1} (\boldsymbol{y}_{\text{in}}^i - \boldsymbol{y}_{\text{in}}^j)\right) \qquad \text{for } i \neq j \qquad (2.9)\end{aligned}$$

This is the equation as presented in Dallaire et al. (2008), however, it is actually only valid for $i \neq j$ due to the independence assumption between $\boldsymbol{x}_{\text{in}}^i$ and $\boldsymbol{x}_{\text{in}}^j$ in equation 2.8. For the diagonal elements of the covariance matrix, $i = j$, the distance between $\boldsymbol{x}_{\text{in}}^i$ and $\boldsymbol{x}_{\text{in}}^j$ will always be zero as they covary 'perfectly'; thus the the diagonal elements are actually all equal to their usual value of σ_f^2,

$$\tilde{K}_{ij} = \sigma_f^2 \qquad \text{if } i = j \qquad (2.10)$$

In the work of Dallaire et al. (2008) the model is used in the situation of known Gaussian distributions on the training points. In our case the input noise variance in unknown, however we can extend the approach by taking derivatives of $\tilde{K}$ w.r.t. Σ_{in} and optimising the input noise level along with the other hyperparameters. Differentiating w.r.t. the input noise variance, or rather, the log of the standard deviation as this is easier to optimise,

$$\frac{\partial \tilde{K}_{ij}}{\partial \log \sigma_{\text{in},d}} = \left(-\frac{2\Lambda_d^{-1}\,\Sigma_{\text{in},d}}{2\Lambda_d^{-1}\Sigma_{\text{in},d} + 1} + \frac{2(y_{\text{in},d}^i - y_{\text{in},d}^j)^2 \Sigma_{\text{in},d}}{(\Lambda_d + 2\Sigma_{\text{in},d})^2}\right) \tilde{K}_{ij} \qquad i \neq j \qquad (2.11)$$

and zero if $i = j$. However, we need not proceed any further with this method as we can show that when the input noise variance is a parameter the model reduces to a standard GP: if we assume the hyperparameters from this standard GP are $[\Lambda, \sigma_f^2, \sigma_{\text{out}}^2]$, and the corresponding hyperparameters from the GP with the expected covariance matrix are $[\tilde{\Lambda}, \tilde{\sigma}_f^2, \tilde{\sigma}_{\text{out}}^2, \tilde{\sigma}_{\text{in}}^2]$, then if we

1. Choose: $0 < \tilde{\sigma}_{\text{out}}^2 < \sigma_{\text{out}}^2$

2. Set: $\tilde{\sigma}_f^2 = \sigma_f^2 + \sigma_{\text{out}}^2 - \tilde{\sigma}_{\text{out}}^2$

3. Set: $\tilde{\Lambda} = \Lambda \dfrac{\sigma_f^4}{\tilde{\sigma}_f^4}$

4. Set: $\sigma_{\text{in}}^2 = \frac{1}{2}(\Lambda - \tilde{\Lambda})$

we have that,

$$\left|2\tilde{\Lambda}^{-1}\sigma_{\text{in}}^2 + I\right|^{-1/2} \tilde{\sigma}_f^2 = \left|\Lambda^{-1}\tilde{\Lambda}\right|^{1/2} \tilde{\sigma}_f^2 = \frac{\sigma_f^2}{\tilde{\sigma}_f^2}\,\tilde{\sigma}_f^2 = \sigma_f^2 \qquad (2.12)$$

and,

$$\tilde{\Lambda} + 2\sigma_{\text{in}}^2 = \Lambda \qquad (2.13)$$

which implies $\tilde{K}_{ij} = K_{ij}$. In other words, using the expected covariance matrix and optimising the input noise variance provides no extra flexibility to the model: the model classes are the same and thus the optimal setting of the hyperparameters and predictive distributions are also identical.

2.2.2 A Taylor Series Approach

The computational difficulty arises due to the complexity of the function f implied by the GP prior. An alternative method therefore, proposed in Girard and Murray-Smith (2003), is to use a local approximation to the GP function, which is simpler to work with. A natural form for this local approximation is to use the Taylor series to approximate the GP latent function f about an input point $\boldsymbol{y}_{\text{in}}$,

$$
\begin{aligned}
y_{\text{out}} &= f(\boldsymbol{x}_{\text{in}}, \boldsymbol{\theta}) + \epsilon_{\text{out}} && (2.14)\\
&= f(\boldsymbol{y}_{\text{in}} - \boldsymbol{\epsilon}_{\text{in}}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}_{\text{out}} && (2.15)\\
&= f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) - \boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} + \frac{1}{2}\boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial^2 f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}^T \partial \boldsymbol{y}_{\text{in}}} \boldsymbol{\epsilon}_{\text{in}} + \ldots + \epsilon_{\text{out}} && (2.16)
\end{aligned}
$$

Girard and Murray-Smith (2003) make their approximation by selecting terms up to and including the quadratic term in the Taylor expansion. However, rather than then marginalising over the uncertainty in the input points they proceed by just taking the expectation over $\boldsymbol{\epsilon}_{\text{in}}$,

$$
\begin{aligned}
y_{\text{out}} &\approx \mathbb{E}_{\boldsymbol{\epsilon}_{\text{in}}}\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) - \boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} + \frac{1}{2}\boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial^2 f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}^T \partial \boldsymbol{y}_{\text{in}}} \boldsymbol{\epsilon}_{\text{in}} + \epsilon_{\text{out}} \right] && (2.17)\\
&= f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) + \frac{1}{2}\operatorname{Tr}\left\{ \frac{\partial^2 f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}^T \partial \boldsymbol{y}_{\text{in}}} \Sigma_{\text{in}} \right\} + \epsilon_{\text{out}} && (2.18)
\end{aligned}
$$

as $\boldsymbol{\epsilon}_{\text{in}}$ is zero-mean. As far as we can tell there is no justification made for only taking the expectation, except perhaps for computational reasons. We shall return to this in our model, NIGP. For the sake of simplifying the following derivations we will assume independent noise on each of the input dimensions. This allows us to write,

$$
y_{\text{out}} = f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) + \frac{1}{2}\sum_{d=1}^{D}\left\{ \frac{\partial^2 f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial y_{\text{in},d}^2} \sigma_{\text{in},d}^2 \right\} + \epsilon_{\text{out}} \quad (2.19)
$$

We can then proceed to marginalise out the GP latent function by making use of the fact that the derivatives of a GP are also GPs and thus have closed forms for their mean and covariance (Solak et al., 2003). As shown in their paper (Girard and Murray-Smith, 2003), this leads to GP predictions with the following moments,

$$
\begin{aligned}
\mathbb{E}[y_{\text{out}} \mid \boldsymbol{y}_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta}] &= \boldsymbol{q}^T Q^{-1} Y_{\text{out}} && (2.20)\\
\mathbb{V}[y_{\text{out}} \mid \boldsymbol{y}_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta}] &= q - \boldsymbol{q}^T Q^{-1} \boldsymbol{q} && (2.21)
\end{aligned}
$$

where q and $\boldsymbol{q}$ represent different variables (notation taken from Girard and Murray-Smith (2003)),

$$Q_{ij} = k(Y^i_{\text{in}}, Y^j_{\text{in}}) + \sum_{d=1}^{D} \frac{\partial^2 k(Y^i_{\text{in}}, Y^j_{\text{in}})}{(\partial Y^i_{\text{in},d})^2} \sigma^2_{\text{in},d} + \frac{1}{4} \sum_{d=1}^{D} \sum_{e=1}^{D} \frac{\partial^4 k(Y^i_{\text{in}}, Y^j_{\text{in}})}{(\partial Y^i_{\text{in},d})^2 (\partial Y^j_{\text{in},e})^2} \sigma^2_{\text{in},d} \sigma^2_{\text{in},e} + \sigma^2_{\text{out}} \delta_{ij} \quad (2.22)$$

$$\boldsymbol{q}_i = k(Y^i_{\text{in}}, \boldsymbol{y}_{\text{in}}) + \sum_{d=1}^{D} \frac{\partial^2 k(Y^i_{\text{in}}, \boldsymbol{y}_{\text{in}})}{(\partial Y^i_{\text{in},d})^2} \sigma^2_{\text{in},d} + \frac{1}{4} \sum_{d=1}^{D} \sum_{e=1}^{D} \frac{\partial^4 k(Y^i_{\text{in}}, \boldsymbol{y}_{\text{in}})}{(\partial Y^i_{\text{in},d})^2 \partial y^2_{\text{in},e}} \sigma^2_{\text{in},d} \sigma^2_{\text{in},e} \quad (2.23)$$

$$q = k(\boldsymbol{y}_{\text{in}}, \boldsymbol{y}_{\text{in}}) + \sum_{d=1}^{D} \frac{\partial^2 k(\boldsymbol{y}_{\text{in}}, \boldsymbol{y}_{\text{in}})}{\partial y^2_{\text{in},d}} \sigma^2_{\text{in},d} + \frac{1}{4} \sum_{d=1}^{D} \sum_{e=1}^{D} \frac{\partial^4 k(\boldsymbol{y}_{\text{in}}, \boldsymbol{y}_{\text{in}})}{\partial y^2_{\text{in},d} \partial y^2_{\text{in},e}} \sigma^2_{\text{in},d} \sigma^2_{\text{in},e} \quad (2.24)$$

with the Kronecker $\delta_{ij} = 1$ if $i = j$ and zero otherwise. This method has a significant drawback in that it requires fourth order derivatives of the covariance function; or, even worse, if we are to train the covariance function hyperparameters by gradient descent we will need fifth order derivatives. As each derivative adds a factor of the dimension D to the complexity, this method has a potential computational cost of $\mathcal{O}(D^5)$. This is likely to be too expensive for systems with even a moderate number of dimensions. A much cheaper approximation based on the Taylor series would be to only use first order terms. In the derivation of Girard and Murray-Smith (2003) however these terms disappear as they base their method on only the expectation over the input noise. Because this method does not scale well we exclude it from our experiments.

2.2.3 Input Noise can be Treated as Heteroscedastic Output Noise

In regression we are interested in finding the output as a function of the input. In this setting, the effect of input noise on the output is modulated by the shape of the function. We can see from figure 2.1 that, broadly speaking, in areas where the function is flat the input noise has very little effect and in areas where the function is steep input noise has a large effect. This leads to the output noise variance varying across the input space, a feature often called *heteroscedasticity*. Figure 2.4 demonstrates this for the function considered in figure 2.1. This intuition suggests that, rather than modelling the input noise directly we can instead model its effects on the output if we can handle heteroscedastic output noise. One method for modelling datasets with input noise is, therefore, to hold the input measurements to be deterministic and then use a heteroscedastic GP model. This approach has been strengthened by the breadth of research published recently on extending GPs to heteroscedastic data.

A common approach to modelling changing variance with a GP, as proposed by Goldberg et al. (Goldberg et al., 1998), is to make the noise variance, or rather its log, a random variable and attempt to estimate its form at the same time as estimating the posterior mean. Goldberg et al. suggested using a second GP to model the noise level as a function of the input location,

$$\log \sigma^2_{\text{out}} = g(\boldsymbol{y}_{\text{in}}) \quad (2.25)$$

$$g \sim \mathcal{GP}(m_g, k_g) \quad (2.26)$$

Inference is intractable in this model and so Goldberg et al. (1998) introduced a sampling scheme to both find posteriors on the hyperparameters and to make predictions. This approach proved to be computationally expensive and so Kersting et al. (2007) suggested an approximate "most likely" training scheme. In this approach the uncertainty in the noise-GP is ignored and the mean value

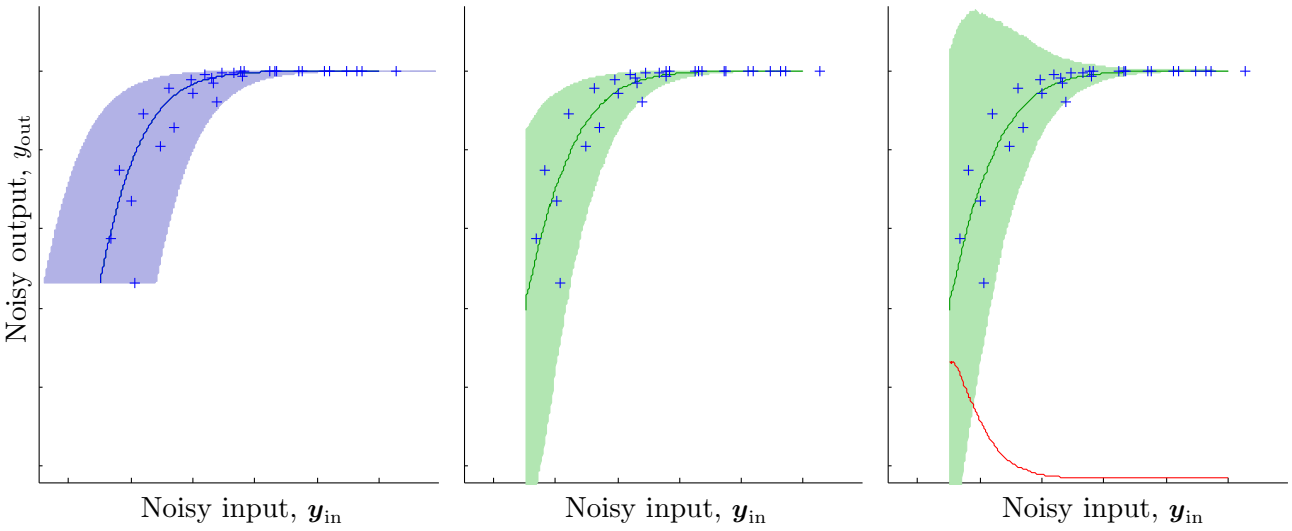


Figure 2.4: Illustration of how input noise can be treated as heteroscedastic output noise. The plot on the left (repeated from figure 2.1) shows a function and two standard deviations of input noise. The middle figure shows a reinterpretation of the input noise as heteroscedastic output noise (note that the noise variance on the left of the plot is much greater than the variance on the right of the plot), which is more appropriate for regression. The shaded region shows the fifth to ninety fifth confidence interval — the distribution is *not* Gaussian. The right hand plot shows a moment-matched Gaussian fit to the output distribution. The red line at the bottom of the right plot shows how the Gaussian's variance changes as a function of the input.

used,

$$\log \sigma_{\text{out}} \;=\; \mathbb{E}_g \left[g(\boldsymbol{y}_{\text{in}}) \right] \tag{2.27}$$

The hyperparameters are then optimised by using the following iterative scheme,

1. Train a standard (homoscedastic) GP on the observed data
2. Compute an estimate of the log noise variance at each training point, $\log \sigma^2_{\text{out},i}$ for $i = 1$ to N, using the observed data and the trained GP (GP1)
3. Train a second homoscedastic GP (GP2) using the pairs of observed inputs and estimated noise levels from step 2, $\{\boldsymbol{y}_{\text{in},i}, \log \sigma^2_{\text{out},i}\}_{i=1}^N$
4. Train a third GP (GP3) to fit the observed data, $\{\boldsymbol{y}_{\text{in},i}, y_{\text{out},i}\}_{i=1}^N$, using the posterior mean of GP2 to provide the noise variance at each training point
5. Set GP1 = GP3 and go back to step 2. Repeat until converged (although convergence is not guaranteed).

Other related work with heteroscedastic models includes Yuan and Wahba (Yuan and Wahba, 2004), and Le at al. (Le et al., 2005) who proposed a scheme to find the variance via a maximum-a-posteriori estimate set in the exponential family. Snelson and Ghahramani (Snelson and Ghahramani, 2006b) suggest a different approach whereby the importance of points in a pseudo-training set can be varied, allowing the posterior variance to vary as well. Wilson and Ghahramani broadened the scope still further and proposed Copula and Wishart Process methods (Wilson and Ghahramani, 2010, 2011).

Referring the input noise to the output results in heteroscedasticity with a very particular structure. Therefore, although all of these methods could be applied to datasets with input noise, they are designed for a more general class of heteroscedastic problems and so none of them exploits the structure inherent in input noise datasets. This structure can be used to improve upon current heteroscedastic GP models for datasets with input noise. As we stated previously, one can imagine that in regions where a process is changing its output value rapidly, corrupted input measurements will have a much greater effect than in regions where the output is almost constant. In other words, the effect of the input noise is related to the gradient of the function mapping input to output; this idea can be captured in a first order Taylor series expansion.

In McHutchon and Rasmussen (2011) we introduced a method (NIGP) for training GPs on data with input noise, based on a first order Taylor expansion of the posterior (although we could use higher order terms if desired). The method refers the input noise, through the Taylor expansion, to the output, proportional to the square of the posterior mean function's gradient (with optional additional terms to capture the uncertainty in the gradient). This means we treat the input measurements as if they were deterministic, and inflate the corresponding output variance to compensate. This approach is particularly powerful in the case of time-series data where the output at time $t-1$ becomes the input at time t. In this situation, input measurements are clearly not noise-free: the noise on a particular measurement is the same whether it is considered an input or output. By also assuming the inputs are noisy, NIGP is better able to fit datasets of this type. Furthermore, we can estimate the noise variance on each input dimension, which is often very useful for analysis. We will present and discuss NIGP in section 2.4.

2.3 The Variational Approach

Before we consider the NIGP method we first demonstrate how the problem can be solved by using a variational approach (see section 1.5 for an introduction to variational methods). Titsias and Lawrence (2010) showed how a clever choice of variational distribution can lead to a tractable lower bound on the marginal likelihood of a GP model in the face of uncertainty over the input points. In Titsias and Lawrence (2010), the authors study the GP latent variable model (Lawrence, 2004) (also see chapter 3) although they point out their approximation scheme could be applied to the problem of input noise. In this section we unpack the variational approach and show how it leads to a tractable lower bound, $\mathcal{L}$, for hyperparameter optimisation and how we can make predictions on unseen data.

Using the structure of the graphical model (figure 2.2), we can write the marginal likelihood as,

$$\begin{aligned} p(Y_{\text{out}} \mid Y_{\text{in}}, \boldsymbol{\theta}) &= \frac{1}{p(Y_{\text{in}})} \int p(Y_{\text{out}}, X_{\text{out}}, X_{\text{in}}, Y_{\text{in}}, \boldsymbol{\theta})\, dX_{\text{out}}\, dX_{\text{in}} \\ &= \frac{1}{p(Y_{\text{in}})} \int p(Y_{\text{out}} \mid X_{\text{out}}, \boldsymbol{\theta})\, p(X_{\text{out}} \mid X_{\text{in}}, \boldsymbol{\theta})\, p(Y_{\text{in}} \mid X_{\text{in}}, \boldsymbol{\theta})\, p(X_{\text{in}})\, dX_{\text{out}}\, dX_{\text{in}} \end{aligned} \tag{2.28}$$

Recall that, the terms including the observations in this integral are Gaussian,

$$p(Y_{\text{out}} \mid X_{\text{out}}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}\left(y_{\text{out}}^i;\ x_{\text{out}}^i,\ \sigma_{\text{out}}^2\right) \qquad \text{and} \qquad p(Y_{\text{in}} \mid X_{\text{in}}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}\left(\boldsymbol{y}_{\text{in}}^i;\ \boldsymbol{x}_{\text{in}}^i,\ \Sigma_{\text{in}}\right) \tag{2.29}$$

and whilst the second term is also Gaussian it is a nonlinear function of the latent variables X_{in},

$$p(X_{\text{out}} \mid X_{\text{in}}, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(X_{\text{out}};\; \mathbf{0},\; K\left(X_{\text{in}}, X_{\text{in}}\right)\right) \tag{2.30}$$

which makes the integral in equation 2.28 intractable. However, we can compute a lower bound by using a variational approach. Following Titsias and Lawrence (2010), we first need to augment the Gaussian Process with a set of M 'inducing points' (a.k.a. pseudo-outputs), $\boldsymbol{u}$, an $M \times 1$ vector, and their corresponding inputs Z, a $M \times D$ matrix, such that the inducing points and the latent outputs X_{out} are jointly distributed according the the GP,

$$X_{\text{out}}, \boldsymbol{u} \mid X_{\text{in}}, Z, \boldsymbol{\theta} \;\sim\; \mathcal{GP}\left(m, k\right) \tag{2.31}$$

These inducing points are not introduced for the sake of computational speed (as they were in Snelson and Ghahramani (2006b)) but rather to act as an 'information store', which allows learning to take place. The graphical model with these extra variables is shown in figure 2.5. Given the joint distribution in equation 2.31, we can find the conditional distribution,

$$p(x_{\text{out}} \mid \boldsymbol{x}_{\text{in}}, \boldsymbol{u}, Z, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(x_{\text{out}};\; \boldsymbol{k}(\boldsymbol{x}_{\text{in}}, Z)\, K(Z, Z)^{-1}\boldsymbol{u},\; k(\boldsymbol{x}_{\text{in}}, \boldsymbol{x}_{\text{in}}) - \boldsymbol{k}(\boldsymbol{x}_{\text{in}}, Z)\, K(Z, Z)^{-1}\boldsymbol{k}(Z, \boldsymbol{x}_{\text{in}})\right) \tag{2.32}$$

For now we will treat the inducing inputs Z as extra parameters. The marginal likelihood for the extended model is,

$$p(Y_{\text{out}} \mid Y_{\text{in}}, Z, \boldsymbol{\theta}) \;=\; \frac{1}{p(Y_{\text{in}})} \int p(Y_{\text{out}} \mid X_{\text{out}}, \boldsymbol{\theta})\, p(X_{\text{out}} \mid X_{\text{in}}, \boldsymbol{u}, Z, \boldsymbol{\theta})\, p(Y_{\text{in}} \mid X_{\text{in}}, \boldsymbol{\theta})\, p(\boldsymbol{u} \mid Z, \boldsymbol{\theta})\, p(X_{\text{in}})\; dX_{\text{out}}\; dX_{\text{in}}\; d\boldsymbol{u} \tag{2.33}$$

We now introduce a variational distribution on the latent variables, which we choose to be of the

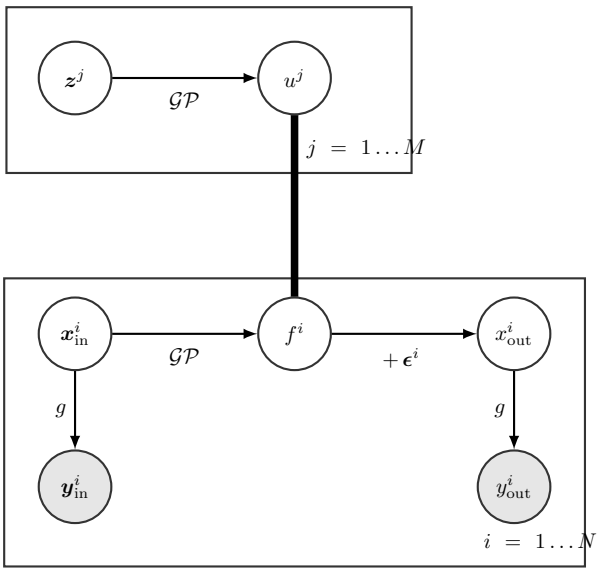


Figure 2.5: Graphical model of Gaussian Process regression with noisy observations and with a set of inducing points $\{Z, \boldsymbol{u}\}$. The thick line indicates that every u^j is connected to every f^i.

form,

$$q(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u}) \;=\; p(X_{\text{out}} \mid X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, q(X_{\text{in}})\, q(\boldsymbol{u}) \tag{2.34}$$

We will restrict the variational distribution on in latent inputs to be a fully factored Gaussian,

$$q(X_{\text{in}}) \;=\; \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{x}_{\text{in}}^i;\; \boldsymbol{\mu}_x^i,\; \Sigma_x^i) \tag{2.35}$$

With this form for the variational distributions, the lower bound on the marginal likelihood $\mathcal{L}\big(q(\boldsymbol{u},\, X_{\text{in}}),\, \boldsymbol{\theta}\big)$ is,

$$\begin{aligned}
&p(Y_{\text{out}} \mid Y_{\text{in}},\, Z,\, \boldsymbol{\theta}) \\
&\quad\geq \int q(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u}) \log \frac{p(Y_{\text{out}} \mid X_{\text{out}},\, \boldsymbol{\theta})\, p(X_{\text{out}} \mid X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, p(X_{\text{in}} \mid Y_{\text{in}},\, \boldsymbol{\theta})\, p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})}{q(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u})}\; dX_{\text{out}}\; dX_{\text{in}}\; d\boldsymbol{u} \\
&\quad= \int q(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u}) \log \frac{p(Y_{\text{out}} \mid X_{\text{out}},\, \boldsymbol{\theta})\, p(X_{\text{out}} \mid X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, p(X_{\text{in}} \mid Y_{\text{in}},\, \boldsymbol{\theta})\, p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})}{p(X_{\text{out}} \mid X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, q(X_{\text{in}})\, q(\boldsymbol{u})}\; dX_{\text{out}}\; dX_{\text{in}}\; d\boldsymbol{u} \\
&\quad= \int q(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u}) \log \frac{p(Y_{\text{out}} \mid X_{\text{out}},\, \boldsymbol{\theta})\, p(X_{\text{in}} \mid Y_{\text{in}},\, \boldsymbol{\theta})\, p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})}{q(X_{\text{in}})\, q(\boldsymbol{u})}\; dX_{\text{out}}\; dX_{\text{in}}\; d\boldsymbol{u} \\
&\quad= \int p(X_{\text{out}} \mid X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, q(X_{\text{in}})\, q(\boldsymbol{u}) \log p(Y_{\text{out}} \mid X_{\text{out}},\, \boldsymbol{\theta})\; dX_{\text{out}}\; dX_{\text{in}}\; d\boldsymbol{u} \\
&\qquad\qquad + \int q(X_{\text{in}}) \log p(X_{\text{in}} \mid Y_{\text{in}},\, \boldsymbol{\theta})\; dX_{\text{in}} \;-\; \mathrm{KL}\left(q(\boldsymbol{u}) \,||\, p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})\right) \;+\; \mathrm{H}\left(q(X_{\text{in}})\right)
\end{aligned} \tag{2.36}$$

$$\triangleq\; \mathcal{L}\big(q(\boldsymbol{u},\, X_{\text{in}}),\, \boldsymbol{\theta}\big) \tag{2.37}$$

Note that the lower bound is currently a functional of the variational distributions on $\boldsymbol{u}$ and X_{in}, as well as the GP hyperparameters $\boldsymbol{\theta}$. There are two terms with integrals remaining to be solved in equation 2.36, which we have highlighted in colour. The first of these terms requires integrating over X_{out}, X_{in}, and $\boldsymbol{u}$; we solve the integral over X_{out} for this term in equation 2.38, then over X_{in} in equation 2.39, and finally over $\boldsymbol{u}$ in equation 2.49. We solve the integral over X_{in} in the second highlighted term, in equation 2.40. First the integral over X_{out} for the first highlighted term, introducing the shorthand $K_{ZZ} \;=\; K(Z,\, Z)$,

$$\begin{aligned}
&\int p(X_{\text{out}} |\, X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta}) \log p(Y_{\text{out}} \mid X_{\text{out}},\, \boldsymbol{\theta})\; dX_{\text{out}} \\
&\quad= \sum_{i=1}^{N} \int p(x_{\text{out}}^i \mid \boldsymbol{x}_{\text{in}}^i,\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta}) \log \mathcal{N}\left(y_{\text{out}}^i;\; x_{\text{out}}^i,\; \sigma_{\text{out}}^2\right)\; dx_{\text{out}} \\
&\quad= -\frac{N}{2} \log 2\pi \;-\; \frac{N}{2} \log \sigma_{\text{out}}^2 \;-\; \frac{1}{2} \sum_{i=1}^{N} \int p(x_{\text{out}}^i \mid \boldsymbol{x}_{\text{in}}^i,\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, \frac{(y_{\text{out}}^i - x_{\text{out}}^i)^2}{\sigma_{\text{out}}^2}\; dx_{\text{out}} \\
&\quad= -\frac{N}{2} \log 2\pi \;-\; \frac{N}{2} \log \sigma_{\text{out}}^2 \\
&\qquad\quad - \frac{1}{2\sigma_{\text{out}}^2} \sum_{i=1}^{N} \Bigg[\underbrace{\left(y_{\text{out}}^i - \boldsymbol{k}(\boldsymbol{x}_{\text{in}}^i,\, Z)\, K_{ZZ}^{-1}\, \boldsymbol{u}\right)^2}_{\mathbb{E}_{x_{\text{out}}}[y_{\text{out}} - x_{\text{out}}]^2} \;+\; \underbrace{k(\boldsymbol{x}_{\text{in}}^i, \boldsymbol{x}_{\text{in}}^i) - \boldsymbol{k}(\boldsymbol{x}_{\text{in}}^i,\, Z)\, K_{ZZ}^{-1}\, \boldsymbol{k}(Z,\, \boldsymbol{x}_{\text{in}}^i)}_{\mathbb{V}_{x_{\text{out}}}[y_{\text{out}} - x_{\text{out}}]} \Bigg]
\end{aligned} \tag{2.38}$$

Note that if we had not introduced the inducing points then we would have taken the expectations in equation 2.38 over the distribution $p(X_{\text{out}} \mid X_{\text{in}},\, \boldsymbol{\theta}) \;=\; \mathcal{N}(\boldsymbol{0},\; K)$. This would have led to the first term (in green) being just $(y_{\text{out}}^i)^2$ and the second (blue), $k(\boldsymbol{x}_{\text{in}}^i,\, \boldsymbol{x}_{\text{in}}^i)$, which for a stationary kernel is

a constant, and thus neither of these terms would depend on $\boldsymbol{x}_{\text{in}}$. We would not therefore be able to learn anything meaningful with this bound. However, by introducing the inducing points we have circumvented this problem by providing additional variables for learning. Having solved the integral over X_{out} for the first term in equation 2.36 we now solve the integral w.r.t. X_{in},

$$\begin{aligned}
&\int q(X_{\text{in}})\, p(X_{\text{out}}|\, X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, \log p(Y_{\text{out}} \mid X_{\text{out}},\, \boldsymbol{\theta})\; dX_{\text{out}}\; dX_{\text{in}} \\
&\quad = -\frac{N}{2}\log 2\pi \;-\; \frac{N}{2}\log \sigma^2_{\text{out}} \;-\; \frac{1}{2\sigma^2_{\text{out}}}\sum_{i=1}^{N}\Bigg[\left(y^i_{\text{out}} - \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\,\boldsymbol{u}\right)^2 \\
&\qquad + \boldsymbol{u}^T K_{ZZ}^{-1}\, \mathbb{V}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\,\boldsymbol{u} \;+\; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[k(\boldsymbol{x}^i_{\text{in}}, \boldsymbol{x}^i_{\text{in}})\right] \\
&\qquad - \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\, \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right] \;-\; \text{tr}\left\{K_{ZZ}^{-1}\, \mathbb{V}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right]\right\}\Bigg]
\end{aligned} \tag{2.39}$$

With a Gaussian form for $q(X_{\text{in}})$ we can solve the required expectations in equation 2.39 for a limited set of covariance functions, such as the squared exponential (see Appendix A) and polynomial kernels. We have solved two of the three integrals in the first highlighted term of equation 2.36: the integral over $\boldsymbol{u}$ still remains. However, we have not yet specified a form for the variational distribution on the inducing points, $q(\boldsymbol{u})$, and so we shall pause here and look instead at the second highlighted integral term in equation 2.36,

$$\begin{aligned}
&\int q(X_{\text{in}}) \log p(X_{\text{in}} \mid Y_{\text{in}},\, \boldsymbol{\theta})\; dX_{\text{in}} \\
&\quad = \sum_{i=1}^{N}\int q(\boldsymbol{x}^i_{\text{in}})\; \log p(\boldsymbol{x}^i_{\text{in}} \mid \boldsymbol{y}^i_{\text{in}},\, \boldsymbol{\theta})\; d\boldsymbol{x}^i_{\text{in}} \\
&\quad = -\frac{N}{2}\log 2\pi \;-\; \frac{N}{2}\log|\Sigma_{\text{in}}| \;-\; \frac{1}{2}\,\mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\left(\boldsymbol{x}^i_{\text{in}} - \boldsymbol{y}^i_{\text{in}}\right)^T \Sigma_{\text{in}}^{-1}\left(\boldsymbol{x}^i_{\text{in}} - \boldsymbol{y}^i_{\text{in}}\right)\right] \\
&\quad = -\frac{N}{2}\log 2\pi \;-\; \frac{N}{2}\log|\Sigma_{\text{in}}| \;-\; \frac{1}{2}\sum_{i=1}^{N}\left(\boldsymbol{\mu}^i_x - \boldsymbol{y}^i_{\text{in}}\right)^T \Sigma_{\text{in}}^{-1}\left(\boldsymbol{\mu}^i_x - \boldsymbol{y}^i_{\text{in}}\right) \;-\; \frac{1}{2}\sum_{i=1}^{N}\text{tr}\left\{\Sigma_{\text{in}}^{-1}\Sigma^i_x\right\}
\end{aligned} \tag{2.40}$$

At this point, the expression for the lower bound is,

$$\begin{aligned}
\mathcal{L}\big(q(\boldsymbol{u}),\, q(X_{\text{in}}),\, \boldsymbol{\theta}\big) = \; & -\frac{N}{2}\log 2\pi \;-\; \frac{N}{2}\log \sigma^2_{\text{out}} \;-\; \frac{1}{2\sigma^2_{\text{out}}}\sum_{i=1}^{N}\Bigg[\int q(\boldsymbol{u})\left(y^i_{\text{out}} - \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\,\boldsymbol{u}\right)^2 d\boldsymbol{u} \\
& + \int q(\boldsymbol{u})\, \boldsymbol{u}^T K_{ZZ}^{-1}\, \mathbb{V}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\,\boldsymbol{u}\; d\boldsymbol{u} \;+\; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[k(\boldsymbol{x}^i_{\text{in}}, \boldsymbol{x}^i_{\text{in}})\right] \\
& - \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\, \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right] \;-\; \text{tr}\left\{K_{ZZ}^{-1}\, \mathbb{V}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right]\right\}\Bigg] \\
& - \frac{N}{2}\log 2\pi \;-\; \frac{N}{2}\log|\Sigma_{\text{in}}| \;-\; \frac{1}{2}\sum_{i=1}^{N}\left(\boldsymbol{\mu}^i_x - \boldsymbol{y}^i_{\text{in}}\right)^T \Sigma_{\text{in}}^{-1}\left(\boldsymbol{\mu}^i_x - \boldsymbol{y}^i_{\text{in}}\right) \;-\; \frac{1}{2}\sum_{i=1}^{N}\text{tr}\left\{\Sigma_{\text{in}}^{-1}\Sigma^i_x\right\} \\
& - \text{KL}\left(q(\boldsymbol{u})\,||\,p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})\right) \;+\; \text{H}\left(q(X_{\text{in}})\right)
\end{aligned} \tag{2.41}$$

We still need to specify the variational distribution $q(\boldsymbol{u})$ and solve the integrals over $\boldsymbol{u}$. However, rather than picking a particular $q(\boldsymbol{u})$, we can use variational calculus to find the optimal setting. The

only terms in the lower bound (equation 2.41) which depend on the inducing points are,

$$\mathcal{L}\big(q(\boldsymbol{u}),\, q(X_{\text{in}}),\, \boldsymbol{\theta}\big) \;=\; \int q(\boldsymbol{u}) \Bigg[\log \frac{p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})}{q(\boldsymbol{u})} \;-\; \frac{1}{2\sigma^2_{\text{out}}} \sum_{i=1}^{N} \Big((y^i_{\text{out}})^2 - 2\, y^i_{\text{out}}\, \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})} \big[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z) \big]\, K_{ZZ}^{-1}\, \boldsymbol{u} \Big) \\ -\; \frac{1}{2\sigma^2_{\text{out}}} \sum_{i=1}^{N} \boldsymbol{u}^T\, K_{ZZ}^{-1}\, \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})} \big[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\, \boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z) \big]\, K_{ZZ}^{-1}\, \boldsymbol{u} \Bigg]\, d\boldsymbol{u} \;+\; c \tag{2.42}$$

where c represents all the terms which do not depend on the inducing points and we have rewritten the KL term in the lower bound in terms of an integral. We now introduce some shorthand to keep the expressions manageable,

$$E^i_k \;\triangleq\; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})} \big[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z) \big] \qquad \text{and} \qquad E_{kk} \;\triangleq\; \sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})} \big[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\, \boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z) \big] \tag{2.43}$$

where the subscript k does not indicate an index but rather represents the covariance function, E indicates the expectation, and the superscript i indicates the expectation is taken over the ith input point, $\boldsymbol{x}^i_{\text{in}}$. Thus E^i_k is a $1 \times M$ row vector and E_{kk} is a $M \times M$ matrix. We now use variational calculus to solve for the optimal distribution on $\boldsymbol{u}$.

$$\frac{\partial \mathcal{L}\big(q(\boldsymbol{u}),\, q(X_{\text{in}}),\, \boldsymbol{\theta}\big)}{\partial q(\boldsymbol{u})}$$

$$= \log \frac{p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})}{q(\boldsymbol{u})} \;-\; 1 \;-\; \frac{1}{2\sigma^2_{\text{out}}}\, \boldsymbol{u}^T K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \boldsymbol{u} \;-\; \frac{1}{2\sigma^2_{\text{out}}} \sum_{i=1}^{N} (y^i_{\text{out}}\, y^i_{\text{out}} - 2\, y^i_{\text{out}}\, E^i_k\, K_{ZZ}^{-1}\, \boldsymbol{u}) \;=\; 0$$

$$\frac{p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})}{q(\boldsymbol{u})} \;\propto\; \exp\left(1 \;+\; \frac{1}{2\sigma^2_{\text{out}}}\, \boldsymbol{u}^T K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \boldsymbol{u} \;+\; \frac{1}{2\sigma^2_{\text{out}}} \sum_{i=1}^{N} (y^i_{\text{out}}\, y^i_{\text{out}} - 2\, y^i_{\text{out}}\, E^i_k\, K_{ZZ}^{-1}\, \boldsymbol{u}) \right)$$

$$q(\boldsymbol{u}) \;\propto\; \mathcal{N}(\boldsymbol{u};\; \boldsymbol{0},\; K_{ZZ}) \exp\left(-1 \;-\; \frac{1}{2\sigma^2_{\text{out}}}\, \boldsymbol{u}^T K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \boldsymbol{u} \;-\; \frac{1}{2\sigma^2_{\text{out}}} \sum_{i=1}^{N} \big((y^i_{\text{out}})^2 - 2\, y^i_{\text{out}}\, E^i_k\, K_{ZZ}^{-1}\, \boldsymbol{u} \big) \right)$$

$$= \exp\left(-\frac{1}{2} \boldsymbol{u}^T\, K_{ZZ}^{-1}\, \boldsymbol{u} \;-\; 1 \;-\; \frac{1}{2\sigma^2_{\text{out}}}\, \boldsymbol{u}^T K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \boldsymbol{u} \;-\; \frac{1}{2\sigma^2_{\text{out}}} \sum_{i=1}^{N} \big(\big((y^i_{\text{out}})^2 - 2\, y^i_{\text{out}}\, \mathbb{E}^i_k\, K_{ZZ}^{-1}\, \boldsymbol{u} \big) \right)$$

$$= \exp\left(-\frac{1}{2}\, \boldsymbol{u}^T \left(K_{ZZ}^{-1} + \frac{K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}}{\sigma^2_{\text{out}}} \right) \boldsymbol{u} \;+\; \frac{1}{\sigma^2_{\text{out}}} \sum_{i=1}^{N} y^i_{\text{out}}\, E^i_k\, K_{ZZ}^{-1}\, \boldsymbol{u} \;-\; \frac{1}{2\sigma^2_{\text{out}}} \sum_{i=1}^{N} (y^i_{\text{out}})^2 \;-\; 1 \right) \tag{2.44}$$

$$\propto\; \exp\left(-\frac{1}{2} \left[\boldsymbol{u}^T \left(K_{ZZ}^{-1} + \frac{K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}}{\sigma^2_{\text{out}}} \right) \boldsymbol{u} \;-\; \frac{2\, E_{yk}\, K_{ZZ}^{-1}\, \boldsymbol{u}}{\sigma^2_{\text{out}}} \right] \right) \tag{2.45}$$

Equation 2.44 is the form of an un-normalised Gaussian. Thus, with some rearranging, we find that the optimal variational distribution on the inducing points, $q^*(\boldsymbol{u})$, is,

$$q^*(\boldsymbol{u}) \;=\; \mathcal{N}(\boldsymbol{u};\; \boldsymbol{\mu}_u,\; \Sigma_u) \tag{2.46}$$

$$\Sigma_u \;=\; K_{ZZ} \left(K_{ZZ} \;+\; \frac{1}{\sigma^2_{\text{out}}} \sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})} \big[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\, \boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z) \big] \right)^{-1} K_{ZZ} \tag{2.47}$$

$$\boldsymbol{\mu}_u \;=\; \frac{1}{\sigma^2_{\text{out}}}\, \Sigma_u\, K_{ZZ}^{-1} \sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})} \big[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}}) \big]\, y^i_{\text{out}} \tag{2.48}$$

Note that (as expected) this has the same form as the optimal $q(\boldsymbol{u})$ found in Titsias and Lawrence

(2010). We can now find the lower bound by completing the integrals over the inducing points. Looking at equation 2.36 we can see that there are only two to do: one in the first highlighted term (which we left in our previous derivations with this term) and one in the KL term. We will first solve the integral in the first highlighted term of equation 2.36. We previously solved the integrals over X_{out} and X_{in} for this term and so we now integrate the result of those integrals, shown in equation 2.39, w.r.t. $q^*(\boldsymbol{u})$,

$$
\begin{aligned}
&\int p(X_{\text{out}} \mid X_{\text{in}},\, \boldsymbol{u},\, Z,\, \boldsymbol{\theta})\, q(X_{\text{in}})\, q^*(\boldsymbol{u})\, \log p(Y_{\text{out}} \mid X_{\text{out}},\, \boldsymbol{\theta})\; dX_{\text{out}}\; dX_{\text{in}}\; d\boldsymbol{u} \\
&\quad = \; -\frac{N}{2}\log 2\pi \;-\; \frac{N}{2}\log \sigma^2_{\text{out}} \;-\; \frac{1}{2\sigma^2_{\text{out}}}\Bigg(\mathbb{E}_{q^*(u)}\left[\boldsymbol{u}^T K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \boldsymbol{u}\right] \;-\; \mathbb{E}_{q^*(u)}\left[\boldsymbol{u}\right]^T K_{ZZ}^{-1} \sum_{i=1}^{N} (E_k^i)^T\, y^i_{\text{out}} \\
&\qquad\qquad + \; y^i_{\text{out}}\, y^i_{\text{out}} \;+\; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[k(\boldsymbol{x}^i_{\text{in}}, \boldsymbol{x}^i_{\text{in}})\right] \;-\; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\, \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right] \\
&\qquad\qquad\qquad\qquad - \; \text{tr}\left\{ K_{ZZ}^{-1}\, \mathbb{V}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right]\right\} \Bigg) \\
&\quad = \; -\frac{N}{2}\log 2\pi \;-\; \frac{N}{2}\log \sigma^2_{\text{out}} \;-\; \frac{1}{2\sigma^2_{\text{out}}}\Bigg(\boldsymbol{\mu}_u^T K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \boldsymbol{\mu}_u \;+\; \text{tr}\left\{ K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \Sigma_u\right\} \\
&\qquad - \; \boldsymbol{\mu}_u^T K_{ZZ}^{-1} \sum_{i=1}^{N} (E_k^i)^T\, y^i_{\text{out}} \;+\; y^i_{\text{out}}\, y^i_{\text{out}} \;+\; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[k(\boldsymbol{x}^i_{\text{in}}, \boldsymbol{x}^i_{\text{in}})\right] \\
&\qquad\qquad - \; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\, \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right] \;-\; \text{tr}\left\{ K_{ZZ}^{-1}\, \mathbb{V}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right]\right\} \Bigg)
\end{aligned}
\tag{2.49}
$$

Secondly, we compute the KL term in the lower bound,

$$
-\text{KL}\left(q^*(\boldsymbol{u}) \,||\, p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})\right) \;=\; -\frac{1}{2}\left(\text{tr}\left\{K_{ZZ}^{-1}\, \Sigma_u\right\} \;+\; \boldsymbol{\mu}_u^T K_{ZZ}^{-1}\, \boldsymbol{\mu}_u \;-\; M \;-\; \log|\Sigma_u| \;+\; \log|K_{ZZ}|\right) \tag{2.50}
$$

The complete variational lower bound on the marginal likelihood is,

$$
\begin{aligned}
&p(Y_{\text{out}} \mid Y_{\text{in}},\, Z,\, \boldsymbol{\theta}) \\
&\quad \geq \; -N\log 2\pi \;-\; \frac{N}{2}\log \sigma^2_{\text{out}} \;-\; \frac{1}{2\sigma^2_{\text{out}}}\Bigg(\boldsymbol{\mu}_u^T K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \boldsymbol{\mu}_u \;+\; \text{tr}\left\{ K_{ZZ}^{-1}\, E_{kk}\, K_{ZZ}^{-1}\, \Sigma_u\right\} \\
&\qquad - \; \boldsymbol{\mu}_u^T K_{ZZ}^{-1} \sum_{i=1}^{N} (E_k^i)^T\, y^i_{\text{out}} \;+\; y^i_{\text{out}}\, y^i_{\text{out}} \;+\; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[k(\boldsymbol{x}^i_{\text{in}}, \boldsymbol{x}^i_{\text{in}})\right] \\
&\qquad - \; \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^i_{\text{in}},\, Z)\right]\, K_{ZZ}^{-1}\, \mathbb{E}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right] \;-\; \text{tr}\left\{ K_{ZZ}^{-1}\, \mathbb{V}_{q(\boldsymbol{x}^i_{\text{in}})}\left[\boldsymbol{k}(Z,\, \boldsymbol{x}^i_{\text{in}})\right]\right\} \Bigg) \\
&\qquad - \; \frac{N}{2}\log|\Sigma_{\text{in}}| \;-\; \frac{1}{2}\sum_{i=1}^{N}\left(\boldsymbol{\mu}_x^i - \boldsymbol{y}_{\text{in}}^i\right)^T \Sigma_{\text{in}}^{-1}\left(\boldsymbol{\mu}_x^i - \boldsymbol{y}_{\text{in}}^i\right) \;-\; \frac{1}{2}\sum_{i=1}^{N}\text{tr}\left\{\Sigma_{\text{in}}^{-1}\Sigma_x^i\right\} \\
&\quad - \frac{1}{2}\left(\text{tr}\left\{K_{ZZ}^{-1}\, \Sigma_u\right\} \;+\; \boldsymbol{\mu}_u^T K_{ZZ}^{-1}\, \boldsymbol{\mu}_u \;-\; M \;-\; \log|\Sigma_u| \;+\; \log|K_{ZZ}|\right) \;+\; \frac{MD}{2}\log(2\pi e) \;+\; \sum_{i=1}^{N}\log|\Sigma_x^i|
\end{aligned}
\tag{2.51}
$$

$$
\triangleq \; \mathcal{L}^*\left(q(X_{\text{in}}),\, Z,\, \boldsymbol{\theta}\right) \tag{2.52}
$$

In summary, we cannot compute the true marginal likelihood for GP regression with noisy inputs. However, we have now formed a lower bound on the marginal likelihood $\mathcal{L}^*\left(q(X_{\text{in}}),\, Z,\, \boldsymbol{\theta}\right)$, which is a

functional of the variational distribution on the latent inputs $q(X_{\text{in}})$, the inducing inputs Z, and the GP hyperparameters $\boldsymbol{\theta}$. Note that the lower bound no longer explicitly depends on $q(\boldsymbol{u})$ as we have an (optimally) defined $q^*(\boldsymbol{u})$ in terms of the other parameters. For a particular setting of the GP hyperparameters the true marginal likelihood is a fixed value, as we have stated however, the lower bound additionally depends on $q(X_{\text{in}})$ and Z. Thus how we set these *variational* parameters affects only how tight the lower bound is to the true marginal likelihood. Given that we have a lower bound, we can find the tightest approximation to the true marginal likelihood by maximising $\mathcal{L}^*\big(q(X_{\text{in}}),\, Z,\, \boldsymbol{\theta}\big)$ w.r.t. $q(X_{\text{in}})$ and Z. Ultimately, we are not interested in evaluating the marginal likelihood but rather finding the setting of $\boldsymbol{\theta}$ which leads to the maximum true marginal likelihood. We can approximate this by also maximising the lower bound with respect to $\boldsymbol{\theta}$, concurrently with optimising the variational parameters. Note that this does not guarantee that the $\boldsymbol{\theta}$ found by maximising the lower bound is the same as the setting which maximises the true marginal likelihood, although if the bound is tight we would expect that the values are close.

Optimisation can be performed by gradient ascent as we can differentiate the lower bound in equation 2.51 w.r.t. the GP hyperparameters, the noise variances Σ_{in} and σ^2_{out}, and the moments of the variational distribution $q(X_{\text{in}})$, $\boldsymbol{\mu}_x^i$ and Σ_x^i. There will be a large number of parameters compared to the number of data points as we have a mean and variance for each noisy input value, plus the hyperparameters and noise variances. The variational approach protects us from overfitting as we are always optimising a lower bound on the marginal likelihood. However, the large number of parameters does make optimisation more difficult — there may be many local optima. A particularly unwelcome local optimum can occur due to the fact that our variational distribution contains the inducing points. This means that by maximising the lower bound we are trying to find the distribution q such that,

$$q(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u}) \;=\; \arg\min\; \mathrm{KL}\left(\, q(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u}) \,||\, p(X_{\text{out}},\, X_{\text{in}},\, \boldsymbol{u} \mid Y_{\text{in}},\, Y_{\text{out}}) \,\right) \tag{2.53}$$

Thus the variational approach wants to model the posterior over the inducing points as well as the data. This can lead to a local optimum whereby the data is treated as pure noise and $q^*(\boldsymbol{u})$ is set to the inducing point prior $p(\boldsymbol{u} \mid Z,\, \boldsymbol{\theta})$, which leads to the KL term in the lower bound (equation 2.50) being exactly zero. Usually reinitialisation is sufficient to escape from this optimum but it is a pitfall that the user must be aware of.

We can make predictions at a clean test input $\boldsymbol{x}^*_{\text{in}}$ by using the approximation,

$$\begin{aligned}
p(x^*_{\text{out}} \mid \boldsymbol{x}^*_{\text{in}},\, Y_{\text{in}},\, Y_{\text{out}},\, Z) \;&=\; \int p(x^*_{\text{out}} \mid \boldsymbol{x}^*_{\text{in}},\, \boldsymbol{u},\, Z)\, p(\boldsymbol{u} \mid Y_{\text{in}},\, Y_{\text{out}},\, Z)\; d\boldsymbol{u} \\
&\approx\; \int p(x^*_{\text{out}} \mid \boldsymbol{x}^*_{\text{in}},\, \boldsymbol{u},\, Z)\, q^*(\boldsymbol{u})\; d\boldsymbol{u} \\
&=\; \int \mathcal{N}\left(x^*_{\text{out}};\; \boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Z)\, K_{ZZ}^{-1}\, \boldsymbol{u},\; k(\boldsymbol{x}^*_{\text{in}}, \boldsymbol{x}^*_{\text{in}}) - \boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Z)\, K_{ZZ}^{-1}\, \boldsymbol{k}(Z, \boldsymbol{x}^*_{\text{in}})\right)\, q^*(\boldsymbol{u})\; d\boldsymbol{u} \\
&=\; \mathcal{N}\left(x^*_{\text{out}};\; \boldsymbol{\mu}^*,\; \Sigma^*\right)
\end{aligned} \tag{2.54}$$

with

$$\boldsymbol{\mu}^* \;=\; \boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Z)\, K_{ZZ}^{-1}\, \boldsymbol{\mu}_u \tag{2.55}$$

$$\Sigma^* \;=\; k(\boldsymbol{x}^*_{\text{in}}, \boldsymbol{x}^*_{\text{in}}) \;-\; \boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Z)\, K_{ZZ}^{-1}\, \boldsymbol{k}(Z, \boldsymbol{x}^*_{\text{in}}) \;+\; \boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Z)\, K_{ZZ}^{-1}\, \Sigma_u\, K_{ZZ}^{-1}\, \boldsymbol{k}(Z, \boldsymbol{x}^*_{\text{in}}) \tag{2.56}$$

To make a prediction at a noisy test input $\boldsymbol{y}^*_{\text{in}}$ we need to integrate out the latent test point,

$$p(x^*_{\text{out}} \mid \boldsymbol{x}^*_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, Z) \;\approx\; \int \mathcal{N}\left(x^*_{\text{out}};\; \boldsymbol{\mu}^*,\; \Sigma^*\right) p(\boldsymbol{x}^*_{\text{in}} \mid \boldsymbol{y}^*_{\text{in}})\; d\boldsymbol{x}^*_{\text{in}}$$

Unfortunately, this is no longer a Gaussian as $\boldsymbol{x}^*_{\text{in}}$ appears inside the nonlinear covariance function. However, as we have discussed earlier for a subset of covariance functions we can still find the first two predictive moments,

$$\boldsymbol{\mu}^*_{x|y} \;=\; \mathbb{E}_{\boldsymbol{x}^*_{\text{in}} \sim \mathcal{N}(\boldsymbol{y}^*_{\text{in}}, \Sigma_{\text{in}})}\left[\boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Z)\right]\; K_{ZZ}^{-1}\, \boldsymbol{\mu}_u \tag{2.57}$$

$$\begin{aligned}\Sigma^*_{x|y} \;=\; & \mathbb{E}_{\boldsymbol{x}^*_{\text{in}} \sim \mathcal{N}(\boldsymbol{y}^*_{\text{in}}, \Sigma_{\text{in}})}\left[k(\boldsymbol{x}^*_{\text{in}}, \boldsymbol{x}^*_{\text{in}})\right] \;+\; \mathbb{E}_{\boldsymbol{x}^*_{\text{in}}}\left[\boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Z)\right] K_{ZZ}^{-1}\, (\Sigma_u \;-\; K_{ZZ})\; K_{ZZ}^{-1}\, \mathbb{E}_{\boldsymbol{x}^*_{\text{in}}}\left[\boldsymbol{k}(Z, \boldsymbol{x}^*_{\text{in}})\right] \\ & +\; \text{tr}\left\{K_{ZZ}^{-1}\, \left(\Sigma_u \;-\; K_{ZZ} \;+\; \boldsymbol{\mu}_u\, \boldsymbol{\mu}_u^T\right)\; K_{ZZ}^{-1}\, \mathbb{V}_{\boldsymbol{x}^*_{\text{in}}}\left[\boldsymbol{k}(Z, \boldsymbol{x}^*_{\text{in}})\right]\right\}\end{aligned} \tag{2.58}$$

where the moments of the squared exponential covariance function can be found in Appendix A. We test this variational approach in section 2.5.

2.4 The Noisy Input Gaussian Process

The variational approach presented in the previous section provides a clear framework within which it makes its approximation. This is very desirable, however it comes at a very large computational cost. This is largely due to the sheer number of parameters we must optimise, which makes computing the derivatives very slow. We now present NIGP, a simpler but faster method for tackling input noise, based on referring the input noise to the output.

2.4.1 Model Specification

Recall that we are modelling an unknown function, f, with a Gaussian Process, based on noisy measurements $\boldsymbol{y}_{\text{in}}$ and y_{out} of the true, latent, variables $\boldsymbol{x}_{\text{in}}$ and x_{out},

$$y_{\text{out}} \;=\; f(\boldsymbol{y}_{\text{in}} \;-\; \boldsymbol{\epsilon}_{\text{in}}, \boldsymbol{\theta}) \;+\; \epsilon_{\text{out}} \tag{2.59}$$

$$f \;\sim\; \mathcal{GP}(m, k) \tag{2.60}$$

where m is the GP prior mean function, k is the GP covariance function, and $\boldsymbol{\theta}$ is a parameter vector. In this section $\boldsymbol{\theta}$ contains the GP hyperparameters, the output noise variance σ^2_{out}, and the input noise variance Σ_{in}. Under this model we can write,

$$p(y_{\text{out}} \mid \boldsymbol{y}_{\text{in}}, \boldsymbol{\epsilon}_{\text{in}}, \boldsymbol{\theta}) \;=\; \mathcal{N}\big(m(\boldsymbol{y}_{\text{in}} - \boldsymbol{\epsilon}_{\text{in}}),\; k(\boldsymbol{y}_{\text{in}} - \boldsymbol{\epsilon}_{\text{in}},\; \boldsymbol{y}_{\text{in}} - \boldsymbol{\epsilon}_{\text{in}})\big) \tag{2.61}$$

We need to marginalise over the input noise vector $\boldsymbol{\epsilon}_{\text{in}}$ as this is an unknown quantity. However, as we stated at the beginning of this chapter, doing this leads to difficulties due to the position of $\boldsymbol{\epsilon}_{\text{in}}$ within the covariance function k. We can use a first order Taylor series expansion of the Gaussian Process latent function f to write an approximation to equation 2.59 as,

$$y_{\text{out}} \;\approx\; f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) \;-\; \boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} \;+\; \epsilon_{\text{out}} \tag{2.62}$$

We can compute the moments of $p(y_{\text{out}} \mid \boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})$ implied by this Taylor series approximation: given that the noise is modelled as zero mean the expected value of y_{out} given the GP prior is still just the GP prior mean (the same as for a standard Gaussian Process),

$$\begin{aligned}\mathbb{E}\left[y_{\text{out}}\right] &= \mathbb{E}_{f,\epsilon_{\text{in}},\epsilon_{\text{out}}}\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) - \boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} + \epsilon_{\text{out}}\right] \\ &= \mathbb{E}_f\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})\right] \\ &= m(y_{\text{out}}, \boldsymbol{\theta}) \end{aligned} \tag{2.63}$$

The variance, however, is different,

$$\begin{aligned}\mathbb{V}\left[y_{\text{out}}\right] &= \mathbb{V}_{f,\epsilon_{\text{in}},\epsilon_{\text{out}}}\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) - \boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} + \epsilon_{\text{out}}\right] \\ &= \mathbb{V}_f\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})\right] + \mathbb{V}_{f,\epsilon_{\text{in}}}\left[\boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] - 2\,\mathbb{C}_{f,\epsilon_{\text{in}}}\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}),\ \boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] \\ &\quad - 2\,\mathbb{C}_{f,\epsilon_{\text{out}}}\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}),\ \epsilon_{\text{out}}\right] - 2\,\mathbb{C}_{f,\epsilon_{\text{in}},\epsilon_{\text{out}}}\left[\boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}},\ \epsilon_{\text{out}}\right] + \sigma_{\text{out}}^2 \end{aligned} \tag{2.64}$$

We will look at each of these terms individually. First, the prior variance of the latent function is given by the GP covariance function,

$$\mathbb{V}_f\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})\right] = k(\boldsymbol{y}_{\text{in}}, \boldsymbol{y}_{\text{in}}) \tag{2.65}$$

The derivative of a GP is also a GP and thus the second variance term in equation 2.64 is a product of two Gaussian distributed vectors. This results in a non-Gaussian solution, however we can still compute the variance,

$$\begin{aligned}\mathbb{V}_{f,\epsilon_{\text{in}}}\left[\boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] &= \text{Tr}\left\{\mathbb{V}\left[\boldsymbol{\epsilon}_{\text{in}}\right]\mathbb{V}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]\right\} + \mathbb{E}\left[\boldsymbol{\epsilon}_{\text{in}}\right]^T \mathbb{V}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] \mathbb{E}\left[\boldsymbol{\epsilon}_{\text{in}}\right] \\ &\quad + \mathbb{E}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]^T \mathbb{V}\left[\boldsymbol{\epsilon}_{\text{in}}\right] \mathbb{E}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] \\ &= \text{Tr}\left\{\Sigma_{\text{in}}\,\mathbb{V}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]\right\} + \mathbb{E}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]^T \Sigma_{\text{in}}\,\mathbb{E}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] \end{aligned} \tag{2.66}$$

where we used $\mathbb{E}[\boldsymbol{\epsilon}_{\text{in}}] = 0$. The mean and variance of the derivative of the GP are computed in section 2.7 at the end of this chapter. The first and second covariance terms in equation 2.64 are zero as the noise is independent of the GP function,

$$2\,\mathbb{C}_{f,\epsilon_{\text{in}}}\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}),\ \boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] = \mathbb{C}_{f,\epsilon_{\text{out}}}\left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}),\ \epsilon_{\text{out}}\right] = 0 \tag{2.67}$$

The third covariance term is,

$$\mathbb{C}_{f,\epsilon_{\text{in}},\epsilon_{\text{out}}}\left[\boldsymbol{\epsilon}_{\text{in}}^T \frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}},\ \epsilon_{\text{out}}\right] = \mathbb{E}[\boldsymbol{\epsilon}_{\text{in}}]^T \mathbb{C}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}},\ \epsilon_{\text{out}}\right] + \mathbb{E}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]^T \mathbb{C}\left[\boldsymbol{\epsilon}_{\text{in}},\ \epsilon_{\text{out}}\right] \tag{2.68}$$

The expectation of the input noise is zero therefore the first term in equation 2.68 is zero, as is the second as we assume the noise corrupts different data points independently. Thus this covariance term

is also zero. The complete variance of y_{out} under our Taylor series approximation is thus,

$$\mathbb{V}\left[y_{\text{out}}\right] \;=\; k(\boldsymbol{y}_{\text{in}}) \;+\; \text{Tr}\left\{\Sigma_{\text{in}}\,\mathbb{V}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]\right\} \;+\; \mathbb{E}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]^{T}\Sigma_{\text{in}}\;\mathbb{E}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right] \;+\; \sigma^{2}_{\text{out}} \quad (2.69)$$

We can easily extend equation 2.69 to find the covariance between multiple observations. However, as we are modelling the noise as independent between different observations we find that all the additional variance contributions in equation 2.69 evaluate to zero for the covariance; thus,

$$\mathbb{C}\left[y^{i}_{\text{out}},\; y^{j}_{\text{out}}\right] \;=\; k(\boldsymbol{y}^{i}_{\text{in}},\,\boldsymbol{y}^{j}_{\text{in}}) \quad (2.70)$$

We have now computed the mean (equation 2.63) and variance (equation 2.69) of the distribution on y_{out}, $p(y_{\text{out}} \mid \boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta})$, implied by our model in equation 2.62. However, if we return to the Taylor series expansion in equation 2.62, we can see that this distribution is not Gaussian due to the product of the two Gaussian vectors $\boldsymbol{\epsilon}_{\text{in}}$ and the derivative of the GP. This means that inference is still not tractable yet. Therefore we must make an approximation if we wish to use analytic methods. We look at two different approximations here,

1. Replace the derivative of the GP in equation 2.62 with its expectation over the GP uncertainty, $\mathbb{E}_{f}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}},\boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]$. This means that the derivative term in equation 2.62 is now deterministic and thus the distribution on y_{out} is Gaussian. This approximation is the same as just taking the derivative w.r.t. the GP posterior mean, rather than w.r.t. the complete GP posterior distribution. We will refer to this approach using the subscript 'm' (for 'mean').

2. Moment match a Gaussian to the true distribution on y_{out} using the exact moments we have computed above. We will refer to this approach with the subscript 'u' (as this approach includes the uncertainty in the derivative of the GP).

Neither of these approximations lead to a different approximate mean for $p(y_{\text{out}} \mid \boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta})$, which remains equal to the GP prior mean $m(\boldsymbol{y}_{\text{in}})$. The variances are different however. Under the first approximation we no longer consider the uncertainty in the derivative as we are just using its expected value. This results in an approximate variance which neglects the trace term in equation 2.69, and so the prior probability of an observation is,

$$q_{\text{m}}(y_{\text{out}} \mid \boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta}) \;=\; \mathcal{N}\left(m(\boldsymbol{y}_{\text{in}}),\; k(\boldsymbol{y}_{\text{in}},\,\boldsymbol{y}_{\text{in}}) \;+\; \partial\bar{f}(\boldsymbol{y}_{\text{in}})^{T}\Sigma_{\text{in}}\;\partial\bar{f}(\boldsymbol{y}_{\text{in}}) \;+\; \sigma^{2}_{\text{out}}\right) \quad (2.71)$$

where we have used the shorthand $\partial\bar{f}(\boldsymbol{y}_{\text{in}}) \;=\; \mathbb{E}_{f}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}},\boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]$. Under the second approximation the prior probability variance contains the additional trace term which represents the uncertainty in the derivative,

$$q_{\text{u}}(y_{\text{out}} \mid \boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta}) \;=\; \mathcal{N}\left(m(\boldsymbol{y}_{\text{in}}),\; k(\boldsymbol{y}_{\text{in}},\,\boldsymbol{y}_{\text{in}}) \;+\; \text{Tr}\left\{\Sigma_{\text{in}}\,\mathbb{V}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}},\,\boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]\right\} \;+\; \partial\bar{f}(\boldsymbol{y}_{\text{in}})^{T}\Sigma_{\text{in}}\;\partial\bar{f}(\boldsymbol{y}_{\text{in}}) \;+\; \sigma^{2}_{\text{out}}\right) \quad (2.72)$$

The first approximation is computationally more attractive as the variance of the derivative involves computations that scale with the number of data points squared. The second approximation is much more accurate as it includes the uncertainty in the derivative, which is often non-negligible. We will look at both of these approximations in the forthcoming sections, referring to our model as NIGP

when the distinction between the two approximations is not required, and 'NIGPm'/'NIGPu' when it is.

Under these approximations multiple data points follow a joint Gaussian distribution, as given by the GP prior plus a corrective diagonal variance term, which we designate as $\tilde{\Sigma}(\boldsymbol{y}_{\text{in}})$,

$$\tilde{\Sigma}(\boldsymbol{y}_{\text{in}}) \;=\; \partial\bar{f}(\boldsymbol{y}_{\text{in}})^T \Sigma_{\text{in}}\; \partial\bar{f}(\boldsymbol{y}_{\text{in}}) \;+\; \text{Tr}\left\{\Sigma_{\text{in}}\, \mathbb{V}\left[\frac{\partial f(\boldsymbol{y}_{\text{in}},\, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}}\right]\right\} \tag{2.73}$$

This extra variance term can either include the trace term or not, depending on computational requirements. In terms of notation, $\tilde{\Sigma}$ should be thought of as a function, returning a scalar when written as a function of a single data point (as in equation 2.73), and a diagonal matrix when written with a set of data points, $\tilde{\Sigma}(Y_{\text{in}})$. It captures the extra uncertainty resulting from the input noise but now referred to the output. As we expected the extra variance is heteroscedastic — largest where the function gradient is steepest and smallest where the gradient is at its most flat (the derivative uncertainty term will cloud this interpretation somewhat if the uncertainty also varies across the space). Note that, if the posterior gradient and its uncertainty is constant across the input space the heteroscedasticity is removed and NIGP is essentially identical to a standard GP. As all observations are jointly Gaussian we can find the predictive distribution at a new test point straightforwardly using conditioning, as in the classic GP derivation. The joint covariance matrix is,

$$\mathbb{C}\left[\boldsymbol{y}^1_{\text{out}},\, \boldsymbol{y}^2_{\text{out}},\, \ldots,\, \boldsymbol{y}^N_{\text{out}},\, \boldsymbol{y}^*_{\text{out}}\right] \;=$$

$$\begin{bmatrix} k(\boldsymbol{y}^1_{\text{in}}, \boldsymbol{y}^1_{\text{in}}) + \tilde{\Sigma}(\boldsymbol{y}^1_{\text{in}}) & k(\boldsymbol{y}^1_{\text{in}}, \boldsymbol{y}^2_{\text{in}}) & \cdots & k(\boldsymbol{y}^1_{\text{in}}, \boldsymbol{y}^N_{\text{in}}) & k(\boldsymbol{y}^1_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) \\ k(\boldsymbol{y}^2_{\text{in}}, \boldsymbol{y}^1_{\text{in}}) & k(\boldsymbol{y}^2_{\text{in}}, \boldsymbol{y}^2_{\text{in}}) + \tilde{\Sigma}(\boldsymbol{y}^2_{\text{in}}) & & k(\boldsymbol{y}^2_{\text{in}}, \boldsymbol{y}^N_{\text{in}}) & k(\boldsymbol{y}^2_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) \\ \vdots & & \ddots & & \\ k(\boldsymbol{y}^N_{\text{in}}, \boldsymbol{y}^1_{\text{in}}) & k(\boldsymbol{y}^N_{\text{in}}, \boldsymbol{y}^2_{\text{in}}) & & k(\boldsymbol{y}^N_{\text{in}}, \boldsymbol{y}^N_{\text{in}}) + \tilde{\Sigma}(\boldsymbol{y}^N_{\text{in}}) & k(\boldsymbol{y}^N_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) \\ k(\boldsymbol{y}^*_{\text{in}}, \boldsymbol{y}^1_{\text{in}}) & k(\boldsymbol{y}^*_{\text{in}}, \boldsymbol{y}^2_{\text{in}}) & & k(\boldsymbol{y}^*_{\text{in}}, \boldsymbol{y}^N_{\text{in}}) & k(\boldsymbol{y}^*_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) + \tilde{\Sigma}(\boldsymbol{y}^*_{\text{in}}) \end{bmatrix} \;+\; \sigma^2_{\text{out}}\, I \tag{2.74}$$

$$= \begin{bmatrix} K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}}\, I & \boldsymbol{k}(Y_{\text{in}},\, \boldsymbol{y}^*_{\text{in}}) \\ \boldsymbol{k}(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}) & k(\boldsymbol{y}^*_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) + \tilde{\Sigma}(\boldsymbol{y}^*_{\text{in}}) + \sigma^2_{\text{out}} \end{bmatrix} \tag{2.75}$$

which, after conditioning on Y_{out}, gives

$$\begin{aligned} p(y_{\text{out}} \mid Y_{\text{in}},\, Y_{\text{out}},\, \boldsymbol{y}^*_{\text{in}}) \;&=\; \mathcal{N}\left(y^*_{\text{out}};\; \mu^*_{\text{out}},\; \sigma^{*2}_{\text{out}}\right) \\ \mu^*_{\text{out}} \;&=\; \boldsymbol{k}(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}) \left[K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I\right]^{-1} \boldsymbol{y} \\ \sigma^{*2}_{\text{out}} \;&=\; k(\boldsymbol{y}^*_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) \;-\; \boldsymbol{k}(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}) \left[K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I\right]^{-1} \boldsymbol{k}(Y_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) \;+\; \tilde{\Sigma}(\boldsymbol{y}^*_{\text{in}}) \;+\; \sigma^2_{\text{out}} \end{aligned} \tag{2.76}$$

An advantage of our approach can be seen in the case of multiple output dimensions for the same set of input dimensions. As the input noise levels are the same for each of the output dimensions, NIGP can use data from all of the outputs when learning the input noise variances. Not only does this give more information about the noise variances without needing further input measurements but it also reduces over-fitting as the learnt noise variances must agree with all output dimensions.

For time-series datasets (where the model has to predict the next state given the current), each dimension's input and output noise variance can be constrained to be the same, since the noise level on a measurement is independent of whether it is an input or output. This further constraint increases the ability of the model to recover the actual noise variances. The model is thus suited to the common task of multivariate time series modelling, which is the focus of chapter 3.

2.4.2 Training

NIGP introduces an extra D hyperparameters compared to the standard GP — one noise variance hyperparameter per input dimension. A major advantage of NIGP is that these hyperparameters can be trained alongside any others by maximisation of the marginal likelihood. This approach automatically includes regularisation of the noise parameters and reduces the effect of over-fitting. The approximate marginal likelihood for a single data point was derived in equations 2.71 and 2.72 (without and with the uncertainty in the derivatives), and we write it here for a set of points,

$$q_{\text{NIGP}}(Y_{\text{out}} \mid Y_{\text{in}}, \boldsymbol{\theta}) = \mathcal{N}\left(Y_{\text{out}};\ \boldsymbol{m}(Y_{\text{in}}),\ K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I\right) \tag{2.77}$$

$$\begin{aligned} \Rightarrow\ -\log q_{\text{NIGP}}(Y_{\text{out}} \mid Y_{\text{in}}, \boldsymbol{\theta}) &= \frac{D}{2}\log 2\pi + \frac{1}{2}\log\left|K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I\right| \\ &\quad + \frac{1}{2}\left(\boldsymbol{m}(Y_{\text{in}}) - Y_{\text{out}}\right)^T \left[K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I\right]^{-1} \left(\boldsymbol{m}(Y_{\text{in}}) - Y_{\text{out}}\right) \\ &= \frac{D}{2}\log 2\pi + \frac{1}{2}\log|K_{\text{n}}| + \frac{1}{2}\left(\boldsymbol{m}(Y_{\text{in}}) - Y_{\text{out}}\right)^T \tilde{\boldsymbol{\beta}} \end{aligned} \tag{2.78}$$

where,

$$K_{\text{n}} = K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I \tag{2.79}$$

$$\tilde{\boldsymbol{\beta}} = K_{\text{n}}^{-1}\left(\boldsymbol{m}(Y_{\text{in}}) - Y_{\text{out}}\right) \tag{2.80}$$

Note that the subscript 'n' indicates that this is the training covariance matrix with added noise, rather than implying an index. Equation 2.78 has the same form as the standard GP negative log marginal likelihood (NLML) except that the noisy covariance matrix K_{n} has the additional, corrective variance term $\tilde{\Sigma}(Y_{\text{in}})$. We can train the hyperparameters using gradient descent on the NLML just as with a standard GP. Assuming a zero mean function, the relevant derivatives are,

$$\begin{aligned} \frac{\partial}{\partial K_{\text{n}}}\left\{-\log q_{\text{NIGP}}(Y_{\text{out}} \mid Y_{\text{in}}, \boldsymbol{\theta})\right\} &= \frac{1}{2}\frac{\partial}{\partial K_{\text{n}}}\left\{\frac{D}{2}\log 2\pi + \frac{1}{2}\log|K_{\text{n}}| + \frac{1}{2}\left(\boldsymbol{m}(Y_{\text{in}}) - Y_{\text{out}}\right)^T \tilde{\boldsymbol{\beta}}\right\} \\ &= \frac{1}{2}\operatorname{tr}\left\{K_{\text{n}}^{-1}\frac{\partial K_{\text{n}}}{\partial K_{\text{n}}}\right\} - \frac{1}{2}\tilde{\boldsymbol{\beta}}^T \frac{\partial K_{\text{n}}}{\partial K_{\text{n}}}\tilde{\boldsymbol{\beta}} \end{aligned} \tag{2.81}$$

$$\frac{\partial K_{\text{n}}}{\partial \boldsymbol{\theta}} = \frac{\partial K(Y_{\text{in}}, Y_{\text{in}})}{\partial \boldsymbol{\theta}} + \frac{\partial \tilde{\Sigma}(Y_{\text{in}})}{\partial \boldsymbol{\theta}} + \frac{\partial \sigma^2_{\text{out}} I}{\partial \boldsymbol{\theta}} \tag{2.82}$$

The first term in equation 2.82 is the derivative of the training point covariance matrix w.r.t. the hyperparameters, which is the same as required for standard GP training. The third term is trivial, thus the interesting term to compute is the derivative of the NIGP correction term. Recall that $\tilde{\Sigma}(Y_{\text{in}})$ is a diagonal matrix and so we look at the derivative of an arbitrary diagonal entry, the point

$\boldsymbol{y}_{\text{in}} \in Y_{\text{in}}$,

$$\frac{\partial \tilde{\Sigma}(\boldsymbol{y}_{\text{in}})}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left\{ \mathbb{E}_f \left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} \right]^T \Sigma_{\text{in}} \, \mathbb{E}_f \left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} \right] + \text{tr} \left\{ \Sigma_{\text{in}} \, \mathbb{V}_f \left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} \right] \right\} \right\} \tag{2.83}$$

To proceed further we need to choose a form for the covariance function. In order to compute the required moments the covariance function must be twice differentiable w.r.t. the input points (once if we neglect the variance of the derivative term). We will choose the popular squared exponential covariance function although many other allowable choices are available,

$$k(\boldsymbol{y}_{\text{in}}^1, \, \boldsymbol{y}_{\text{in}}^2) = \sigma_f^2 \exp\left(-\frac{1}{2} (\boldsymbol{y}_{\text{in}}^1 - \boldsymbol{y}_{\text{in}}^2)^T \Lambda^{-1} (\boldsymbol{y}_{\text{in}}^1 - \boldsymbol{y}_{\text{in}}^2) \right) \tag{2.84}$$

where σ_f^2 and Λ are hyperparameters. As derived at the end of the chapter in section 2.7, for this choice of kernel,

$$\mathbb{E}_f \left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} \right] = \frac{\partial \mathbb{E}_f \left[f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta}) \right]}{\partial \boldsymbol{y}_{\text{in}}} = \Lambda^{-1} \sum_{i=1}^{N} (Y_{\text{in}}^i - \boldsymbol{y}_{\text{in}}) \, \boldsymbol{k}(Y_{\text{in}}^i, \, \boldsymbol{y}_{\text{in}}) \, \boldsymbol{\beta}^i \tag{2.85}$$

$$\mathbb{V}_f \left[\frac{\partial f(\boldsymbol{y}_{\text{in}}, \boldsymbol{\theta})}{\partial \boldsymbol{y}_{\text{in}}} \right] = \frac{\partial^2 k(\boldsymbol{y}_{\text{in}}, \, \boldsymbol{y}_{\text{in}})}{\partial \boldsymbol{y}_{\text{in}} \, \partial \boldsymbol{y}_{\text{in}}^T} - \frac{\partial \boldsymbol{k}(\boldsymbol{y}_{\text{in}}, \, Y_{\text{in}})}{\partial \boldsymbol{y}_{\text{in}}} \, K^{-1} \, \frac{\partial \boldsymbol{k}(Y_{\text{in}}, \, \boldsymbol{y}_{\text{in}})}{\partial \boldsymbol{y}_{\text{in}}} \tag{2.86}$$

In order to calculate the marginal likelihood of the training data we need the posterior distribution, and the slope of its mean, at each of the training points. However, evaluating the posterior from equation 2.76 with $\boldsymbol{y}_{\text{in}}^* \in Y_{\text{in}}$, results in an analytically unsolvable differential equation: the distribution on f is a complicated function of its own derivative. Therefore, we define a two-step approach: first we evaluate a standard GP with the training data, using our initial hyperparameter settings and ignoring the input noise. We then evaluate the derivative terms on this GP at each of the training points and use it to add in the corrective variance term, $\tilde{\Sigma}(Y_{\text{in}})$. This process is summarised in figures 2.6a and 2.6b.

The marginal likelihood of the GP with the corrected variance is then computed, along with its derivatives with respect to the initial hyperparameters, which include the input noise variances. This step involves chaining the derivatives of the marginal likelihood back through the slope calculation. Gradient descent can then be used to improve the hyperparameters. Figure 2.6c shows the GP posterior for the trained hyperparameters and shows how NIGP can reduce output noise level estimates by taking input noise into account. Figure 2.6d shows the NIGP fit for the trained hyperparameters.

To improve the fit further we can iterate this procedure: we use the slopes of the current trained NIGP, instead of a standard GP, to calculate the effect of the input noise, i.e. replace the fit in figure 2.6a with the fit from figure 2.6d and re-train.

2.4.3 Prediction

Making predictions at noisy input locations with NIGP is particularly easy as the standard GP predictive equations can be used, along with the extra covariance term to account for the referred input noise in both the test points and the training points. We therefore use the trained hyperparameters

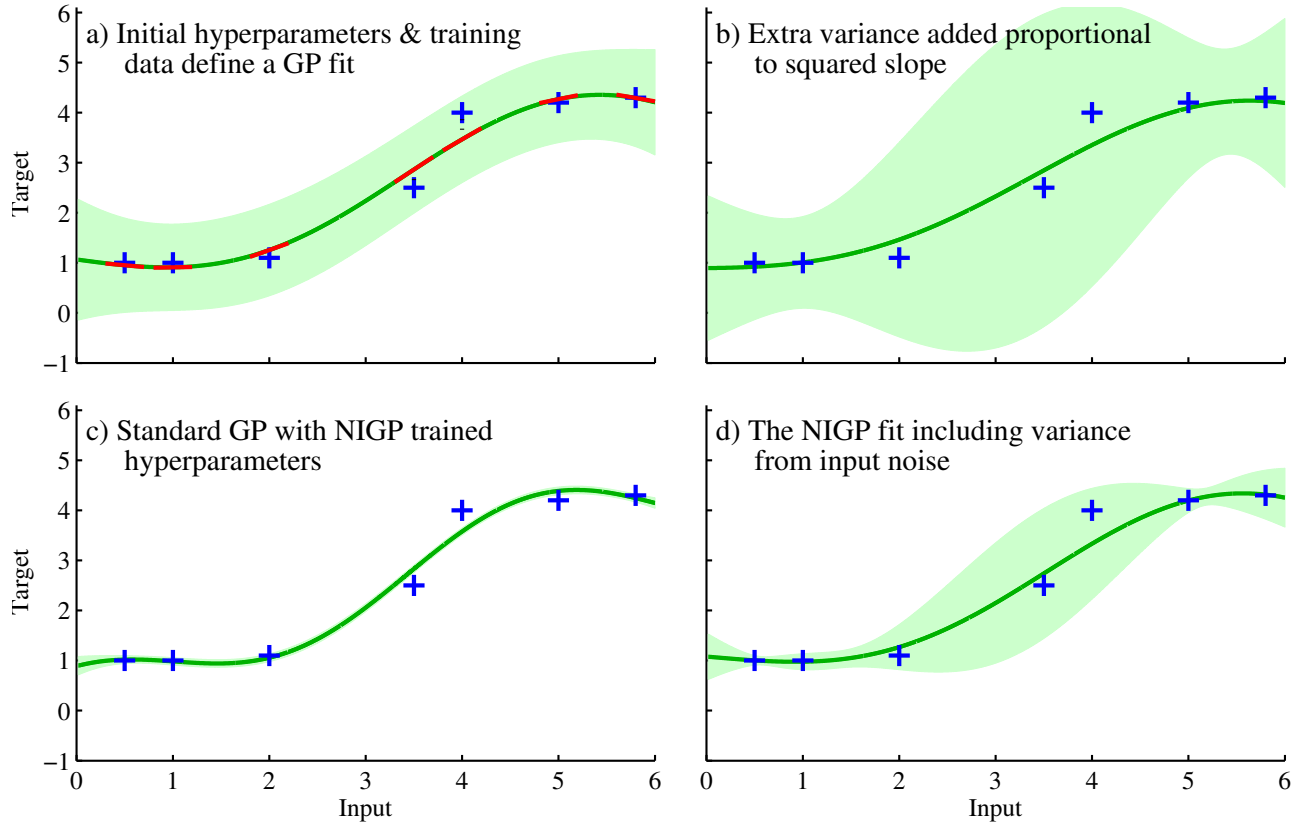


Figure 2.6: Training with NIGP. **(a)** A standard GP posterior distribution can be computed from an initial set of hyperparameters and a training data set, shown by the blue crosses. The gradients of the posterior mean (neglecting the uncertainty of the derivatives for this illustration) at each training point can then be found analytically. **(b)** The NIGP method increases the posterior variance by the square of the posterior mean slope multiplied by the current setting of the input noise variance hyperparameter. The marginal likelihood of this fit is then calculated along with its derivatives w.r.t. initial hyperparameter settings. Gradient descent is used to train the hyperparameters. **(c)** This plot shows the standard GP posterior using the newly trained hyperparameters. Comparing to plot (a) shows that the output noise hyperparameter has been greatly reduced. **(d)** This plot shows the NIGP fit - plot(c) with the input noise corrective variance term, $\text{diag}\{\Delta_{\bar{f}}\,\Sigma_x\,\Delta_{\bar{f}}^T\}$. Plot (d) is related to plot (c) in the same way that plot (b) is related to plot (a).

and the training data to define a GP posterior, which we differentiate at each test point and each training point. The calculated gradients are then used to add in the corrective variance terms.

$$p(y^*_{\text{out}} \mid \boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(y^*_{\text{out}};\; \boldsymbol{\mu}^*,\; \Sigma^*\right) \tag{2.87}$$

$$\boldsymbol{\mu}^* \;=\; \boldsymbol{k}(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}) \left[K(Y_{\text{in}}, Y_{\text{in}}) \;+\; \tilde{\Sigma}(Y_{\text{in}}) \;+\; \sigma^2_{\text{out}}\, I\right]^{-1} Y_{\text{out}} \tag{2.88}$$

$$\Sigma^* \;=\; k(\boldsymbol{y}^*_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) \;-\; \boldsymbol{k}(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}) \left[K(Y_{\text{in}}, Y_{\text{in}}) \;+\; \tilde{\Sigma}(Y_{\text{in}}) \;+\; \sigma^2_{\text{out}}\, I\right]^{-1} \boldsymbol{k}(Y_{\text{in}}, \boldsymbol{y}^*_{\text{in}}) \;+\; \tilde{\Sigma}(\boldsymbol{y}^*_{\text{in}}) \;+\; \sigma^2_{\text{out}} \tag{2.89}$$

There is an alternative option, however. As we discussed at the beginning of this chapter, Gaussian distributed test points are not a problem for a GP as we can compute the exact predictive moments analytically (Girard et al., 2003), it is the uncertain training points that are the problem. As NIGP estimates the input noise variance Σ_{in} during training, we can write $\boldsymbol{x}^*_{\text{in}} \sim \mathcal{N}(\boldsymbol{y}^*_{\text{in}}, \Sigma_{\text{in}})$, and use the uncertain test point equations to compute the predictive mean and variance of the posterior distribution

(for a squared exponential kernel),

$$\mathbb{E}\left[p(y^*_{\text{out}} \mid \boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta})\right] = \boldsymbol{a}^T \left[K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I\right]^{-1} Y_{\text{out}} \tag{2.90}$$

where we have replaced $\boldsymbol{k}(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}})$ with its expected value $\boldsymbol{a}$,

$$\boldsymbol{a} = \int \boldsymbol{k}(\boldsymbol{x}^*_{\text{in}}, Y_{\text{in}})\, p(\boldsymbol{x}^*_{\text{in}} \mid \boldsymbol{y}^*_{\text{in}}, \Sigma_{\text{in}})\, d\boldsymbol{x}^*_{\text{in}} \tag{2.91}$$

$$a_i = \sigma_f^2 \left|\Sigma_{\text{in}} \Lambda^{-1} + I\right|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}^i_{\text{in}} - \boldsymbol{x}^*_{\text{in}})^T (\Sigma_x + \Lambda)^{-1} (\boldsymbol{x}^i_{\text{in}} - \boldsymbol{x}^*_{\text{in}})\right) \tag{2.92}$$

where σ_f^2 and Λ are hyperparameters of the covariance function (see equation 2.84). Note that $\boldsymbol{a}$ is termed $\boldsymbol{q}$ in Deisenroth (2009), which we avoid to prevent confusion with the approximate distributions denoted by q in this thesis. The predictive variance,

$$\mathbb{V}\left[p(y^*_{\text{out}} \mid \boldsymbol{y}^*_{\text{in}}, Y_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta})\right] = \sigma_f^2 - \text{tr}\left(\left[K(Y_{\text{in}}, Y_{\text{in}}) + \tilde{\Sigma}(Y_{\text{in}}) + \sigma^2_{\text{out}} I\right]^{-1} A\right) + \boldsymbol{\beta}^T A \boldsymbol{\beta} - \bar{y_{\text{out}}}_*^2 \tag{2.93}$$

where we used $\bar{y_{\text{out}}}_*$ as shorthand for the predictive mean in equation 2.90 and where,

$$A_{ij} = \frac{k(\boldsymbol{y}^i_{\text{in}}, \boldsymbol{y}^*_{\text{in}})\, k(\boldsymbol{y}^j_{\text{in}}, \boldsymbol{y}^*_{\text{in}})}{|2\Sigma_{\text{in}} \Lambda^{-1} + I|^{\frac{1}{2}}} \exp\left((\boldsymbol{z} - \boldsymbol{y}^*_{\text{in}})^T (\Lambda + \frac{1}{2}\Lambda \Sigma^{-1}_{\text{in}} \Lambda)^{-1} (\boldsymbol{z} - \boldsymbol{y}^*_{\text{in}})\right) \tag{2.94}$$

with $\boldsymbol{z} = \frac{1}{2}(\boldsymbol{y}^i_{\text{in}} + \boldsymbol{y}^j_{\text{in}})$. If the input noise is zero then A is just the outer product of the test point covariance vector $\boldsymbol{k}(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}})$ and the predictive variance reverts to the standard GP form 1.16. This method is computationally slower than using equation 2.87 and is vulnerable to worse results if the learnt input noise variance Σ_{in} is very different from the true value. However, it gives proper consideration to the uncertainty surrounding the test point and exactly computes the moments of the correct posterior distribution. This can lead it to outperform predictions based on equation 2.87, although experimentation in this area tended to be somewhat inconclusive.

2.4.4 Comparison to the Variational Approach

We can see some parallels in the prediction equations for NIGP (equation 2.87) and for the variational approach (equations 2.57 and 2.58). In particular if we look at the predictive variance equations we can see that the variational approach includes a term,

$$\boldsymbol{\mu}_u^T K_{ZZ}^{-1}\, \mathbb{V}_{\boldsymbol{x}^*_{\text{in}} \sim \mathcal{N}(\boldsymbol{y}_{\text{in}},\, \Sigma_{\text{in}})}[k(Z, \boldsymbol{x}^*_{\text{in}})]\, K_{ZZ}^{-1}\, \boldsymbol{\mu}_u$$

which measures how great an effect on the mean predicted output value the uncertainty in the input has; it does this by measuring the variance in the output due to the input noise. In an NIGP prediction there is also an extra variance term to account for the output noise. In the simple case (excluding the uncertainty in the derivative) this extra term is given by

$$Y_{\text{out}}^T K(Y_{\text{in}}, Y_{\text{in}})^{-1} \frac{\partial k(Y_{\text{in}}, \boldsymbol{y}^*_{\text{in}})}{\partial \boldsymbol{y}^*_{\text{in}}}\, \Sigma_{\text{in}}\, \frac{\partial k(\boldsymbol{y}^*_{\text{in}}, Y_{\text{in}})}{\partial \boldsymbol{y}^*_{\text{in}}}\, K(Y_{\text{in}}, Y_{\text{in}})^{-1} Y_{\text{out}}$$

Assuming that there are sufficient inducing points, $K_{ZZ}^{-1}\, \boldsymbol{\mu}_u$ is equivalent to $K(Y_{\text{in}}, Y_{\text{in}})^{-1} Y_{\text{out}}$ and thus we see that the difference is that where the variational approach calculates the variance of the output,

NIGP calculates the squared derivative multiplied by the input noise variance. Of course, these quantities are identical for a linear model as this is how NIGP was derived. We could remove this difference by using the exact output moment calculations for prediction with NIGP, as described at the end of the prediction section above, although it isn't clear that this is always beneficial. It is worth pointing out that there are other terms in the predictive variances of the two approaches: the variational approach also includes terms to account for the uncertainty in the inducing points, likewise NIGP can include a term to incorporate the uncertainty in the derivatives.

2.5 Analysis

We tested the various approaches on a variety of functions and datasets. We start our analysis by illustrating a key difference between the NIGP method and the more general heteroscedastic GP of Kersting et al. (2007).

2.5.1 Near-square wave test set

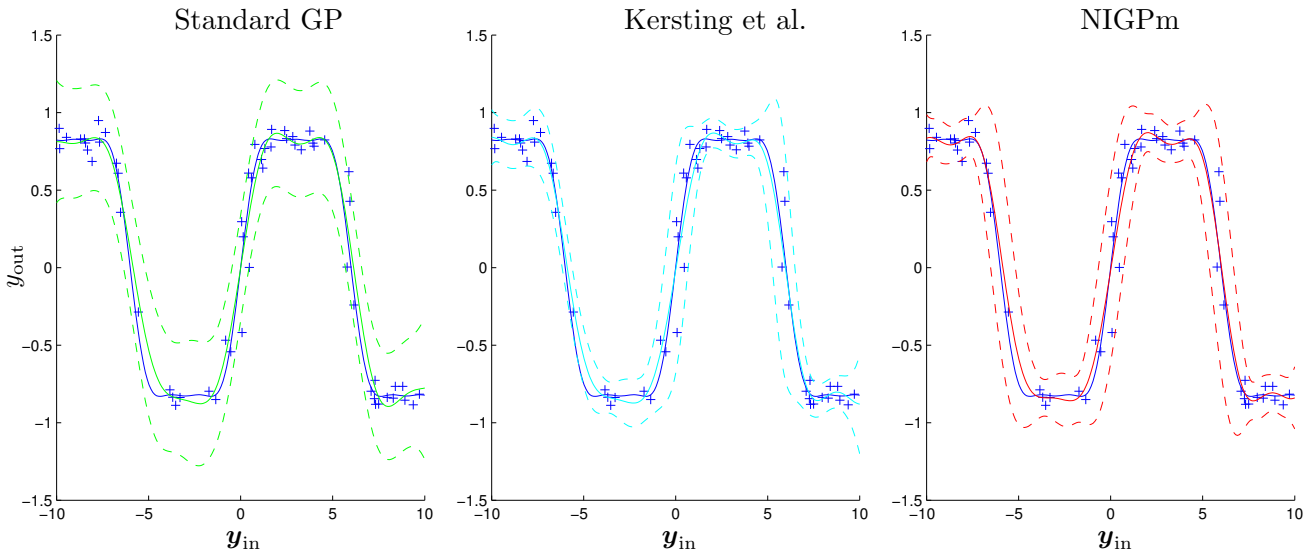


Figure 2.7: Posterior distribution for a near-square wave with $\sigma^2_{\text{out}} = 0.05^2$, $\Sigma_{\text{in}} = 0.3^2$, and 60 data points. The solid line represents the predictive mean and the dashed lines are two standard deviations either side. Also shown are the training points and the underlying function (for $\boldsymbol{x}_{\text{in}}$ to x_{out}). The left image is for standard GP regression, the middle uses Kersting et al.'s MLHGP algorithm, the right image shows NIGPm (neglecting the uncertainty in the derivatives). While the predictive means are similar, both NIGP and MLHGP pinch in the variance around the low noise areas. NIGP correctly expands the variance around all steep areas whereas MLHGP can only do so where high noise is observed (see areas around $\boldsymbol{y}_{\text{in}} = -6$ and $\boldsymbol{y}_{\text{in}} = 1$).

Figure 2.7 shows an example comparison between standard GP regression, Kersting et al.'s MLHGP, and NIGP for a simple near-square wave function. This function was chosen as it has areas of steep gradient and near flat gradient and thus suffers from the heteroscedastic problems we are trying to solve. The posterior means are very similar for the three models, however the variances are quite different. The standard GP model has to take into account the large noise seen around the steep sloped areas by assuming large noise everywhere, which leads to the much larger error bars. NIGP can recover the actual noise levels by taking the input noise into account. Both NIGP and MLHGP

pinch the variance in around the flat regions of the function and expand it around the steep areas. For the example shown in figure 2.7 the standard GP estimated an output noise standard deviation of 0.16 (much too large) compared to NIGP's estimate of 0.052, which is very close to the correct value of 0.050. NIGP also learnt an input noise standard deviation of 0.305, very close to the real value of 0.300. MLHGP does not produce a single estimate of noise levels.

Predictions for 1000 noisy measurements were made using each of the models and the log probability of the test set was calculated. The standard GP model had a log probability per data point of 0.419, MLHGP 0.740, and NIGP 0.885, a significant improvement. Part of the reason for our improvement over MLHGP can be seen around $\boldsymbol{y}_{\text{in}} = 1$: NIGP has near-symmetric 'horns' in the variance around the corners of the square wave, whereas MLHGP only has one 'horn'. This is because in NIGP, the amount of noise expected is proportional to the derivative of the mean squared, which is the same for both sides of the square wave. In Kersting et al.'s model the noise is estimated from the training points themselves. In this example the training points around $\boldsymbol{y}_{\text{in}} = 1$ happen to have low noise and so the learnt variance is smaller. The same problem can be seen around $\boldsymbol{y}_{\text{in}} = -6$ where MLHGP has much too small variance. This illustrates an important aspect of NIGP: the accuracy in plotting the varying effect of noise is only dependent on the accuracy of the mean posterior function and not on an extra, learnt noise model. This means that NIGP typically requires fewer data points to achieve the same accuracy as MLHGP on input noise datasets.

2.5.2 Sigmoid test set

We now move to compare standard GP regression with NIGP, the variational approach, and MLHGP more generally. We first generated a simulated data set from a sigmoid function. Using a simulated data set allows us to manually set the noise levels and compare estimated values to the ground truth. We set the output noise standard deviation to a fairly small value of 0.025, and varied the input noise standard deviation from 0 to 3 (note that the input and output scales are different). Figure 2.8 shows the trained posteriors with an input noise standard deviation of 2. The figure shows how poor a standard GP posterior is: the uncertainty is too large and in the flat regions and too small in the steep region, which is a direct result of having a single variable to model the noise level. Both NIGP and the variational approach increase the uncertainty around the steep area of the function in a very similar manner, demonstrating that the linear approximation at the heart of NIGP is not particularly harming results. The heteroscedastic GP also learns to increase the noise level around the steep area of the function, although it picks a much smaller noise level.

The results of the complete comparison for this test are shown in the box-plots of figure 2.9. This figure shows the comparative performance of the methods on the rows with the methods on the columns, where values above zero indicate that the row method is outperforming the column method. The five different boxes in each plot show the results for the five different noise levels. The boxes are coloured green if the row method outperformed the column method in at least 75% of the trials. We can first see that for zero input noise all methods perform equivalently to a standard GP, and as the noise level is increased the four methods which model the input noise strongly outperform the standard GP. As the input noise reaches its highest level the data has become very difficult to model and so we see a tailing off of performance in all models, although the standard GP is still performing the worst by a very significant amount. The comparison between the two forms of NIGP (without and with the

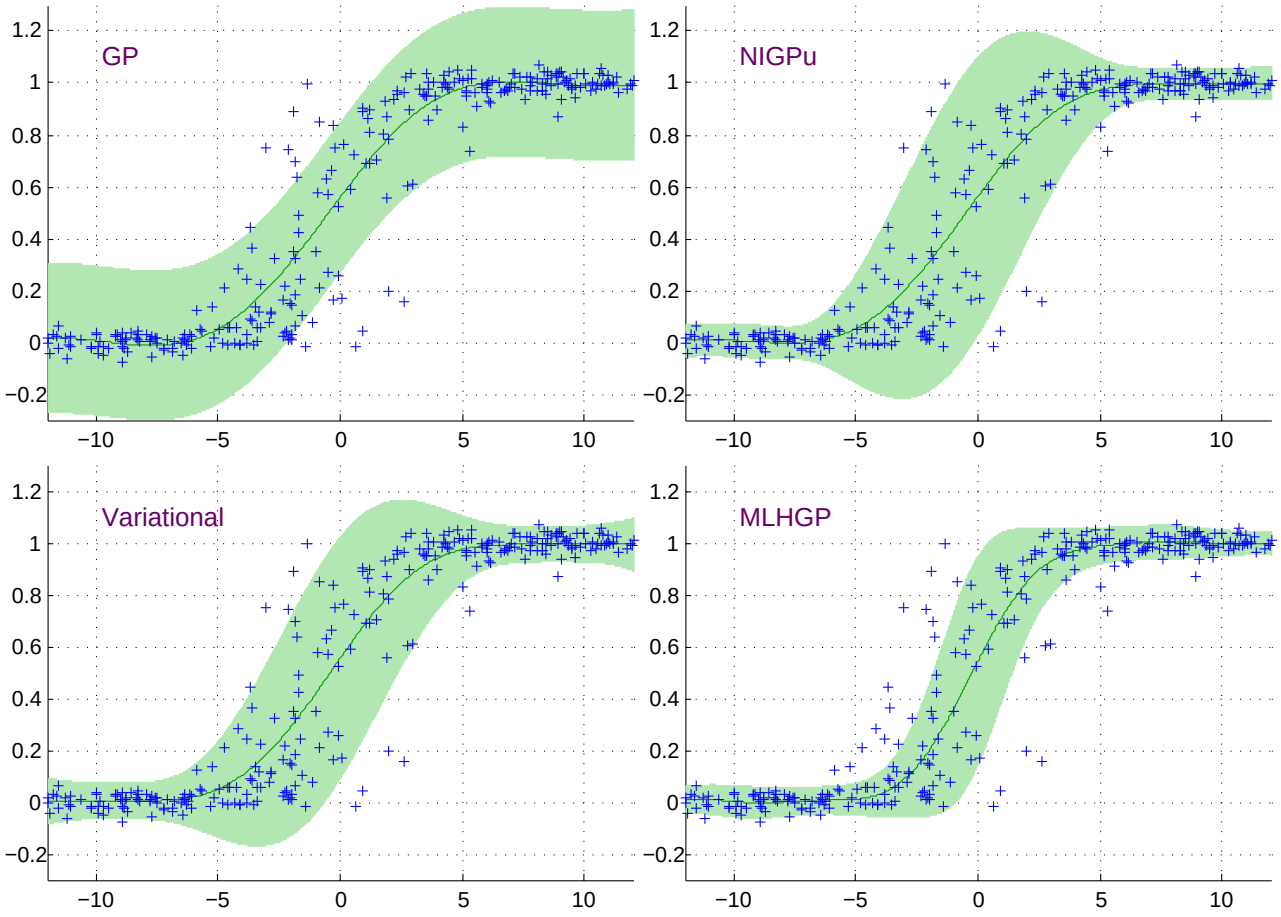


Figure 2.8: The trained posteriors on observed values for the sigmoid training set using the four different models. The true input noise standard deviation of the data for this data set is 2. The green line shows the posterior mean and the shaded region indicates two standard deviations of uncertainty (including both input and output noise).

derivative uncertainty) shows that performance is essentially the same (the scale on the y-axis is very small) although the difference does favour the more complex model. The comparison between NIGP and the variational approach is very illuminating: for moderate input noise levels NIGP outperforms the variational approach but for high noise levels the comparison is the other way around. This is likely to be because for the range of inputs covered by a large input noise variance the GP functions are poorly approximated by a linear model, thus NIGP sees a drop in performance. For small noise levels this is not the case and so NIGP performs well. The variational approach has a very large number of parameters to fit — for a training set consisting of five hundred points there are one thousand and four parameters. This means that optimisation can be difficult and gradient ascent requires a large number of steps to converge; it also increases the chance of being stuck in a local optimum. There are other possible explanations as well, which are common to most variational approaches, such as the optimum of the lower bound being in a different location to the optimum of the true marginal likelihood.

The heteroscedastic GP method was very variable in its performance. It's training procedure had the tendency to get stuck in bad local optima, which led to a number of the outliers seen in figure 2.9. Even when it avoided these situations it rarely outperformed NIGP or the variational approach as it was inclined to underestimate the noise levels, as can be seen in figure 2.8. There are a number of explanations for this, such as it only using a MAP approach and the fact that it is a much more

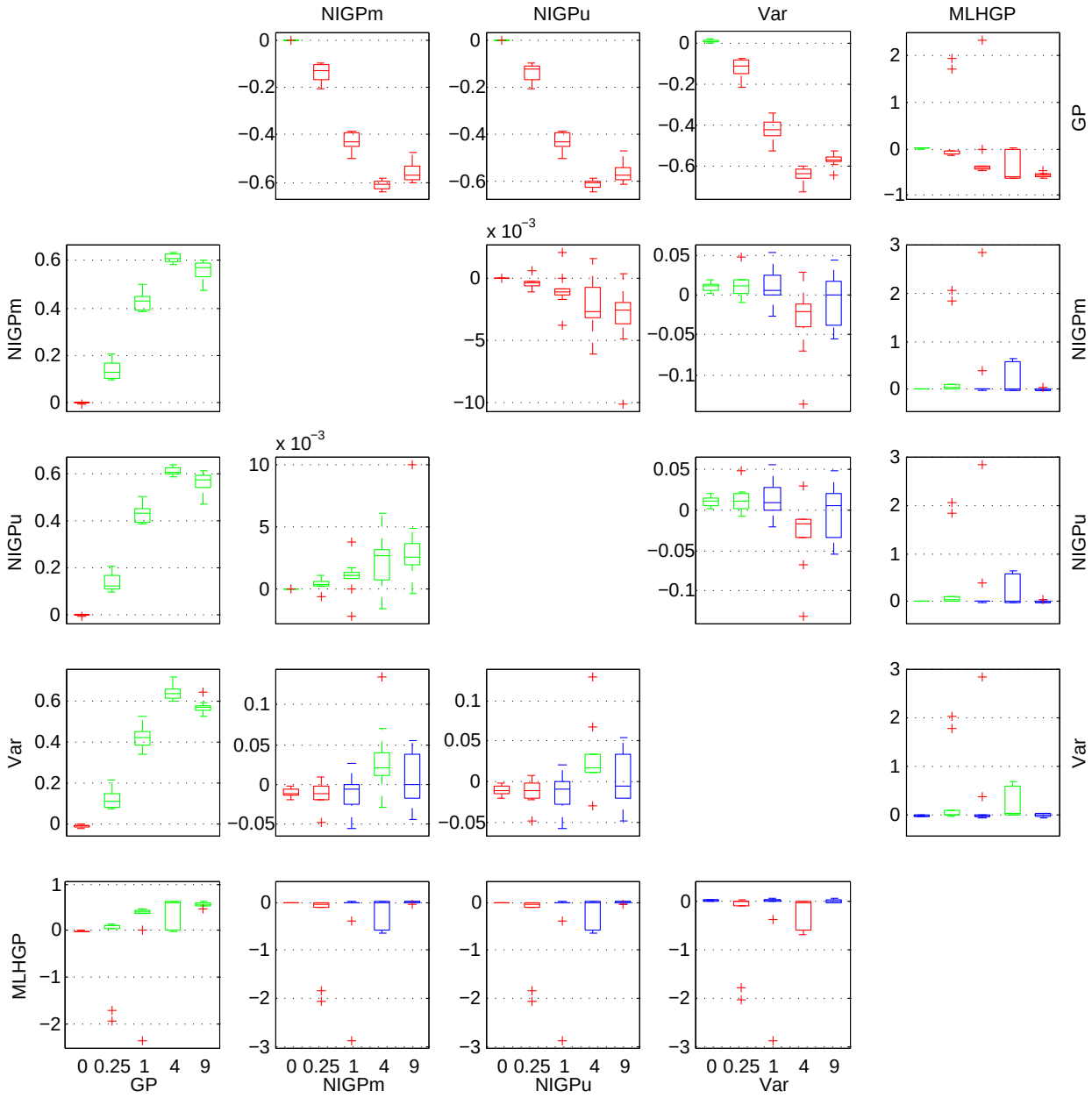


Figure 2.9: Pairwise differences in test log likelihood per test data point for a sigmoid function with five hundred training points and different levels of input noise. The pairwise differences were formed by taking the test log likelihood score of the method on the row and subtracting the score for the method on the column, thus values above zero imply the row is outperforming the column. The methods were trained on fifteen different training sets of five hundred points and test sets of one thousand points. The boxes are coloured green if the method on the row outperformed the method on the column in at least 75% of the data sets. Note that for the comparisons with MLHGP there are four outliers not shown where MLHGP performs very poorly. The five methods being compared are: GP — a standard Gaussian Process, NIGPm — the NIGP model only including the mean derivative term in $\tilde{\Sigma}$, NIGPu — the NIGP model also including the derivative uncertainty in $\tilde{\Sigma}$, Var — the variational approach, and MLHGP — the heteroscedastic GP of Kersting et al. (2007)

general model designed to tackle heteroscedasticity of a more complex form than input noise alone produces. Another way of judging the models is to look at the noise levels which they learn. These are shown in table 2.1, for all methods apart from MLHGP, which does not produce a single noise level. The table shows that recovery of the noise variances are very good for moderate noise levels. The variational approach always tends to underestimate the noise, whereas NIGP tends to overestimate it.

Including the uncertainty in the derivatives for NIGP has a small beneficial effect, particularly at high noise levels. In general the variational approach is slightly better at recovering the noise variances than NIGP is.

Table 2.1: Input and output noise standard deviations learnt by a standard GP, the two variants of NIGP, and the variational approach (MLHGP does not estimate a single noise level) for four different settings of input noise (the four columns of output noise correspond to the four columns of input noise). The values are averaged over 15 trials.

	Input Noise Std Dev				Output Noise Std Dev			
True	0.50	1.00	2.00	3.00	0.025	0.025	0.025	0.025
GP	-	-	-	-	0.049	0.085	0.150	0.202
NIGPm	0.51	1.05	2.28	3.70	0.026	0.026	0.026	0.031
NIGPu	0.51	1.04	2.27	3.65	0.025	0.025	0.025	0.029
Variational	0.50	0.95	1.76	2.64	0.026	0.027	0.029	0.035

It is important to also look at training computation times. Table 2.2 shows the mean training times for each method over the fifteen trials. It is immediately noticeable how much slower the variational method is compared to all the other approaches. There are several factors which contribute to this, a clear one being that with over a thousand parameters all the derivative matrices in the variational algorithm are considerably larger than the corresponding matrices in the other approaches. NIGP shows an order of magnitude decrease in speed compared to the standard GP, although this is slightly artificial as it was run with more optimisation iterations than it required and this figure could be reduced by perhaps a factor of two, depending on the speed-accuracy trade-off required. The variational approach is already being heavily restricted in only using one hundred optimisation steps when there are over a thousand parameters and so it is hard to see extra speed-ups being found in that manner, although there are no doubt savings to be found in the actual implementation.

Table 2.2: Training times in seconds for the various input noise methods on the 1D sigmoid data set with 500 training points. The variational method used 20 inducing points and 100 optimisation steps, the other methods trained until convergence. The times are from running on a quad-core i7 processor at 2.67GHz.

GP	NIGPm	NIGPu	Variational	MLHGP
0.7	6.4	6.7	441.0	18.9

2.5.3 The sunny field test set

We now compare performance on a two dimensional real data set. We collected readings on light intensity levels at various points in a sunny field using an Android phone, as shown in figure 2.10. The phone measured the location using GPS and the light level using its camera as we walked through sunny and shaded areas. The light levels are spatially correlated and so a Gaussian Process is a desirable model for the data. However, both the GPS and light sensor are noisy and so we would expect standard GP regression to struggle. The noise on the GPS locator is non-Gaussian, which adds an extra complexity to the problem. Figure 2.11 shows a heat-map of the posterior means learnt using a standard GP, NIGPu, the variational approach, and the MLHGP. These posterior means all look qualitatively similar, however the variational approach was extremely difficult to train. This is

Figure 2.10: The 'sunny field' data set. An Android phone was used to measure light intensity levels and GPS co-ordinates at various points in the area shown in the photo on the left (note that the photo is taken from a point around the bottom left corner of the data set shown on the right). The right hand plot shows the data points collected where the colour indicates the logarithm of the light intensity measurements.

due to the number of parameters it has to optimise, 1830, compared to the six which the standard GP and NIGP must fit. The extra 1826 parameters in the variational approach are the means and variances of the variational distribution on X_{in}. It is thus strongly desirable to find an alternative method for selecting $q(X_{\text{in}})$ as the current approach is only workable for small data sets. Having said that, there is recent work on parallelising the variational approach for the GP-LVM (Gal et al., 2014), which may also be applicable here and provide a sufficient speed increase, providing there is enough computational power available. We can quantify the models' performance by computing negative log test likelihoods, which are shown in table 2.3. This table shows that NIGPm achieves the best performance, followed by the NIGP variant which includes the derivative uncertainty — in this case the extra uncertainty leads to slightly worse test performance. The heteroscedastic GP lags behind the other methods and is comparable to a standard GP, this appears to be due to issues with training, quite possibly overfitting, due to the MAP approach.

Table 2.3: Negative log test likelihoods per test data point for the 'sunny field' data set. Lower numbers indicate better performance.

GP	NIGPm	NIGPu	Variational	MLHGP
-0.403	-0.425	-0.422	-0.413	-0.401

2.5.4 The cart and pendulum test set

As previously mentioned, NIGP can be adapted to work effectively with time series data, where the outputs become subsequent inputs, and with data which has multiple outputs sharing the same input. In the time series situation the input and output noise variance will be the same and so we combine these two parameters into one. In the case of multiple outputs, each output will share the same input noise levels, thus multiple outputs can greatly help NIGP infer the input noise variance. Our final experiment therefore is to test NIGP on a four dimensional output, time series dataset generated from the cart and pendulum dynamical system (see section 1.7.1 for more details). We also ran the

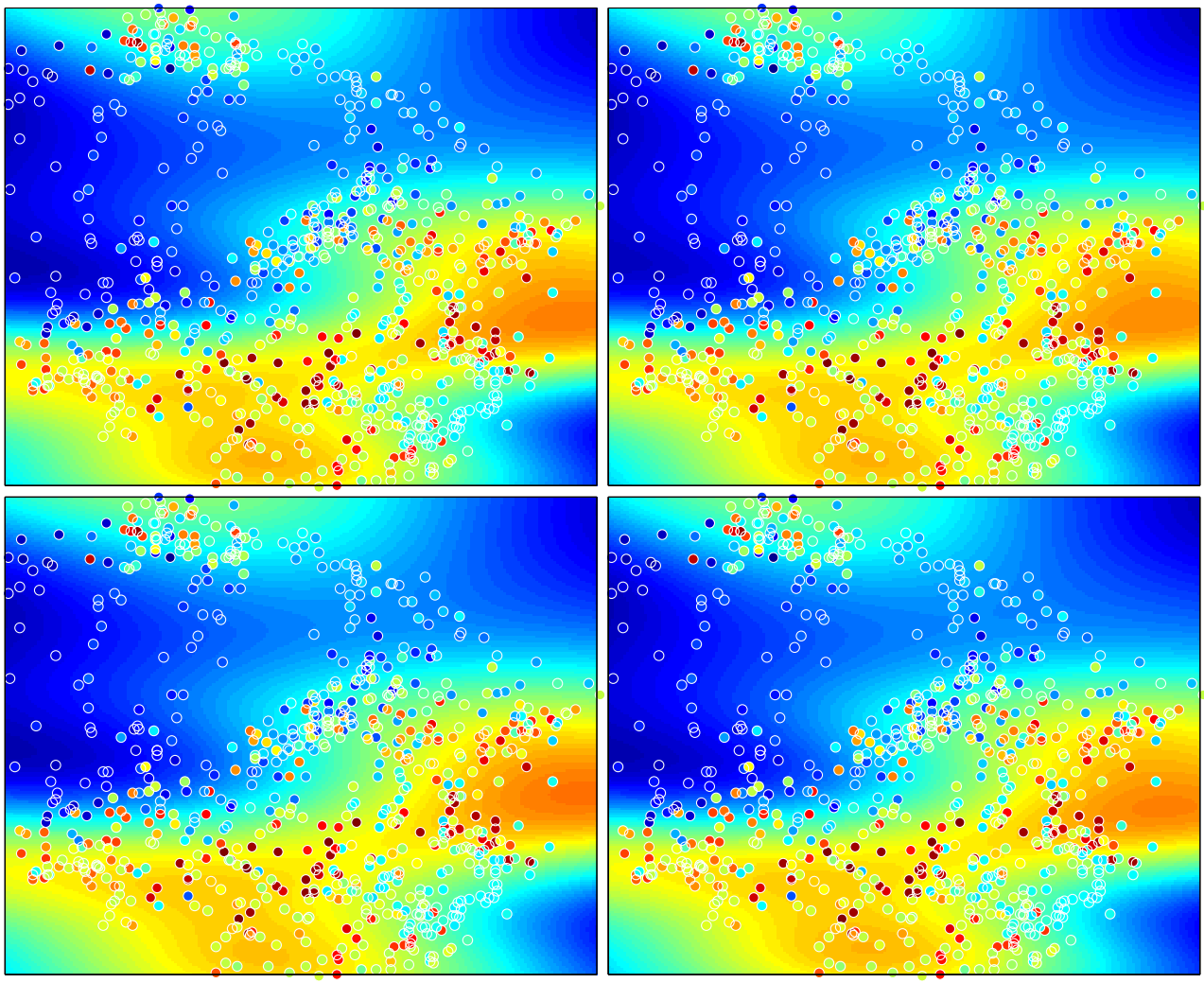


Figure 2.11: Trained posterior means for the 'sunny field' data set using a standard GP (top left), NIGPm (top right), the variational approach (bottom left) and MLHGP (bottom right). The colour shows the amount of light at each point with the test points values circled in white.

standard GP and the heteroscedastic GP on this data, but not the variational approach, firstly because of computational issues, and secondly because a more suitable variational approach to this problem is discussed in section 3.4.4. Figure 2.12 shows the comparative box plots for the cart and pendulum data set. NIGP outperformed the standard GP and MLHGP on all trials, with the two variants of NIGP having very similar results. The heteroscedastic GP performed worst of all on all trials, again likely due to a training issue. Also as it does not directly model input noise it is unable to exploit the structure in the time series data set nor can it use multiple outputs to improve performance.

2.6 Conclusion

Systems from which we can only make noisy observations are all around us, thus it is critically important that modelling strategies can cope with noise. From a Bayesian point-of-view, the 'correct' way of training on input points corrupted by noise is to consider every input point as a latent variable and to integrate them out. However, we started this chapter by demonstrating how this is intractable for standard GP regression as the training inputs appear inside the covariance matrix. We then

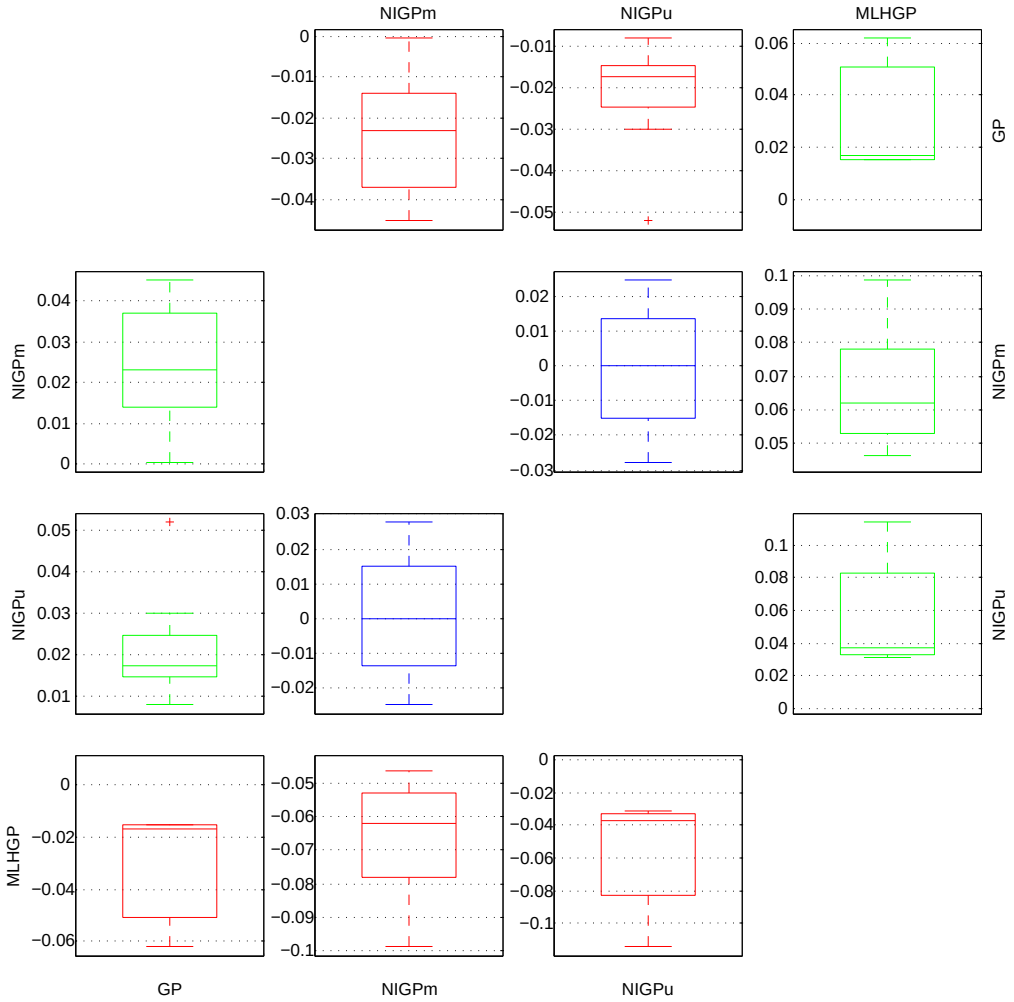


Figure 2.12: Pairwise differences in test log likelihood per test data point for a data set from the cart and pendulum dynamical system. The data set consists of 1500 points, which were split randomly, half-and-half into a training set and test set, over ten different trials. The input is five dimensional and the output four (there is one control variable which appears in the input). The pairwise differences were formed by taking the test log likelihood score of the method on the row and subtracting the score for the method on the column, thus values above zero imply the row is outperforming the column. The boxes are coloured green if the method on the row outperformed the method on the column in at least 75% of the data sets. Note that for the comparisons with MLHGP there are five outliers not shown where MLHGP performs very poorly.

presented and discussed a number of different approaches to Gaussian Process regression with noisy input data, including: taking expectations, a variational approach, and by referring the input noise to the output, viewing the problem as a specific type of heteroscedastic output noise. In the variational approach we use a carefully selected variational distribution to 'delete' the problematic term from the marginal likelihood and thus derive a tractable lower bound, whist treating the noisy inputs probabilistically. Although this method can be very effective it tends to underestimate the noise levels and is nearly two orders of magnitude slower than the NIGP method, as we must optimise the variational distribution on the latent inputs. If this bottleneck could be overcome or avoided then we would expect the variational approach to improve.

In NIGP, we refer the input noise to the output by passing it through a local linear expansion. This adds a term to the likelihood which is proportional to the squared posterior mean gradient, plus a term accounting for the uncertainty in the gradient. Not only does this lead to tractable computations

but it makes intuitive sense — input noise has a larger effect in areas where the function is changing its output rapidly. The model, although simple in its approach, has been shown to be very effective, often outperforming the variational approach, and the more general heteroscedastic GP, as well as the standard GP model in a variety of different regression tasks. Both NIGP and the variational method can make use of multiple outputs, if they are available, to better infer the noise on the shared inputs. Both can also recover a noise variance parameter for each input dimension, which is often useful for analysis of the data set. It is simple to apply NIGP to data sets with multiple outputs and use the shared information to improve performance. Likewise we can exploit properties of time series data, where the output noise is the same as (or a known scaling of) the input noise, to further improve performance on those types of data set.

It is important to state that NIGP has been designed to tackle a particular situation, that of constant-variance input noise, and would not perform so well on a general heteroscedastic problem. It could not be expected to improve over a standard GP on problems where noise levels are proportional to the function or input value for example. We do not see this limitation as too restricting however, as we maintain that constant input noise situations (including those where this is a sufficient approximation) are reasonably common.

In this chapter we limited ourselves to only considering NIGP with a first order Taylor expansion of the GP functions. We would expect this to be a good approximation for any function providing the input noise levels are not too large (i.e. small perturbations around the point we linearised about). In practice, we could require that the input noise level is not larger than the input characteristic length scale. A more accurate model could use a second order Taylor series, which would still be analytic although computationally slower, as we pointed out in the discussion of the method in Girard and Murray-Smith (2003).

2.7 Derivatives of a Gaussian Process

The NIGP method requires differentiating functions distributed according to a Gaussian Process. In this section we derive the distribution on these derivatives in terms of the covariance function. For the derivatives of the squared exponential covariance function see appendix A.

2.7.1 Posterior Distribution on the Derivatives

Let the posterior distribution at a particular (noise-free) test point $\boldsymbol{x}_*$ (we drop the 'in' subscript to keep the notation brief) induced by a GP with training inputs X_{in}, training targets Y_{out}, and hyperparameters $\boldsymbol{\theta}$ be,

$$p(f_* \mid \boldsymbol{x}_*, X_{\text{in}}, Y_{\text{out}}, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(f_*;\; \bar{f}_*,\; \Sigma_*\right) \tag{2.95}$$

We can write the unknown GP function value f_* as the sum of its mean and an auxiliary latent variable z_* such that,

$$\begin{aligned} f(\boldsymbol{x}_*) \;&=\; \bar{f}(\boldsymbol{x}_*) \;+\; z_* \\ f(\boldsymbol{x}_* + \boldsymbol{\delta}_i) \;&=\; \bar{f}(\boldsymbol{x}_* + \boldsymbol{\delta}_i) \;+\; z_{\delta_i} \end{aligned} \tag{2.96}$$

where,

$$p(z_*, z_{\delta_i}) \;=\; \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k_{**} - \boldsymbol{k}_*^T K^{-1} \boldsymbol{k}_* & k_{*\delta_i} - \boldsymbol{k}_*^T K^{-1} \boldsymbol{k}_{\delta_i} \\ k_{\delta_i *} - \boldsymbol{k}_{\delta_i}^T K^{-1} \boldsymbol{k}_* & k_{\delta_i \delta_i} - \boldsymbol{k}_{\delta_i}^T K^{-1} \boldsymbol{k}_{\delta_i} \end{bmatrix}\right) \tag{2.97}$$

and $\boldsymbol{\delta}_i$ is a perturbation on $\boldsymbol{x}_*$, with all zero entries apart from the ith, which is δ,

$$\boldsymbol{\delta}_i \;=\; [0_1,\; \ldots,\; 0_{i-1},\; \delta,\; 0_{i+1},\; \ldots,\; 0_D]^T \tag{2.98}$$

Given that $f(\boldsymbol{x}_*)$ is a scalar, its derivative $\frac{\partial f(\boldsymbol{x}_*)}{\partial \boldsymbol{x}_*}$ is a D-dimensional vector, where the ith element is the derivative of f w.r.t. the ith element of $\boldsymbol{x}$, $x_*^{(i)}$. From the definition of a derivative,

$$\begin{aligned} \frac{\partial f_*}{\partial x_*^{(i)}} \;&=\; \lim_{\delta \to 0} \frac{f(\boldsymbol{x}_* + \boldsymbol{\delta}_i) - f(\boldsymbol{x}_*)}{x_*^{(i)} + \delta - x_*^{(i)}} \\ &=\; \lim_{\delta \to 0} \frac{\bar{f}(\boldsymbol{x}_* + \boldsymbol{\delta}_i) \;+\; z_{\delta_i} - \bar{f}(\boldsymbol{x}_*) \;-\; z_*}{\delta_i} \\ &=\; \lim_{\delta \to 0} \frac{\bar{f}(\boldsymbol{x}_* + \boldsymbol{\delta}_i) \;-\bar{f}(\boldsymbol{x}_*)}{\delta} \;+\; \lim_{\delta \to 0} \frac{z_{\delta_i} \;-\; z_*}{\delta} \\ &=\; \frac{\partial \bar{f}_*}{\partial x_*^{(i)}} \;+ \lim_{\delta \to 0} \frac{z_{\delta_i} \;-\; z_*}{\delta_i} \end{aligned} \tag{2.99}$$

The first term is the derivative of the posterior mean and the second term is a Gaussian distributed variable with a zero mean (as both z_δ and z_* are zero mean, Gaussian variables). Thus the mean of the derivative of f_* is just the derivative of the mean — this can be seen immediately after recalling that both differentiation and expectation are linear operators and thus are commutative. The posterior mean of a GP is given by,

$$\begin{aligned} \bar{f}_* \;&=\; \boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})\, K(X_{\text{in}}, X_{\text{in}})^{-1}\, Y_{\text{out}} \\ \Rightarrow\; \mathbb{E}_{f_*}\left[\frac{\partial f_*}{\partial \boldsymbol{x}_*}\right]^T \;&=\; \frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})}{\partial \boldsymbol{x}_*}\, K(X_{\text{in}}, X_{\text{in}})^{-1}\, Y_{\text{out}} \end{aligned} \tag{2.100}$$

where the derivative of $\boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})$ is a $D \times N$ matrix. To find the variance of the derivative we need to consider the second term in equation 2.99. We will use the shorthand that,

$$k_{aa} \;\triangleq\; k(\boldsymbol{a}, \boldsymbol{a}), \qquad \boldsymbol{k}_a \;\triangleq\; \boldsymbol{k}(X_{\text{in}}, \boldsymbol{a}), \qquad K \;\triangleq\; K(X_{\text{in}}, X_{\text{in}}) \tag{2.101}$$

$$\begin{aligned} \mathbb{V}_{f_*}\left[\frac{\partial f_*}{\partial x_*^{(i)}}\right] \;&=\; \mathbb{V}\left[\lim_{\delta \to 0} \frac{z_{\delta_i} \;-\; z_*}{\delta}\right] \\ &=\; \lim_{\delta \to 0} \frac{1}{\delta^2} \left(\mathbb{V}\left[z_{\delta_i}\right] \;+\; \mathbb{V}\left[z_*\right] \;-\; \mathbb{C}\left[z_{\delta_i},\, z_*\right] \;-\; \mathbb{C}\left[z_*,\, z_{\delta_i}\right]\right) \\ &=\; \lim_{\delta \to 0} \frac{1}{\delta^2} \left(k_{\delta_i \delta_i} - \boldsymbol{k}_{\delta_i}^T K^{-1} \boldsymbol{k}_{\delta_i} + k_{**} - \boldsymbol{k}_*^T K^{-1} \boldsymbol{k}_* - (k_{*\delta_i} - \boldsymbol{k}_*^T K^{-1} \boldsymbol{k}_{\delta_i}) - (k_{\delta_i *} - \boldsymbol{k}_{\delta_i}^T K^{-1} \boldsymbol{k}_*)\right) \\ &=\; \lim_{\delta \to 0} \frac{1}{\delta^2} \left(k_{\delta_i \delta_i} \;-\; k_{*\delta_i} \;-\; k_{\delta_i *} \;+\; k_{**} \;-\; (\boldsymbol{k}_{\delta_i} - \boldsymbol{k}_*)^T K^{-1} (\boldsymbol{k}_{\delta_i} - \boldsymbol{k}_*)\right) \\ &=\; \frac{\partial^2 k(\boldsymbol{x}_*,\, \boldsymbol{x}_*)}{\partial x_*^{(i)}\, \partial x_*^{(i)}} \;-\; \frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})}{\partial x_*^{(i)}}\, K^{-1}\, \frac{\partial \boldsymbol{k}(X_{\text{in}},\, \boldsymbol{x}_*)}{\partial x_*^{(i)}} \end{aligned} \tag{2.102}$$

It is straightforward to extend this to find,

$$\mathbb{C}_{f_*}\left[\frac{\partial f_*}{\partial x_*^{(i)}},\; \frac{\partial f_*}{\partial x_*^{(j)}}\right] \;=\; \frac{\partial^2 k(\boldsymbol{x}_*,\, \boldsymbol{x}_*)}{\partial x_*^{(i)}\, \partial x_*^{(j)}} \;-\; \frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})}{\partial x_*^{(i)}}\, K^{-1}\, \frac{\partial \boldsymbol{k}(X_{\text{in}},\, \boldsymbol{x}_*)}{\partial x_*^{(j)}} \tag{2.103}$$

from which we can construct the full $D \times D$ covariance matrix. Thus,

$$p\left(\frac{\partial f_*}{\partial \boldsymbol{x}_*} \mid \boldsymbol{x}_*,\, X_{\text{in}},\, Y_{\text{out}},\, \boldsymbol{\theta}\right) = \mathcal{N}\left(\frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})}{\partial \boldsymbol{x}_*} K(X_{\text{in}}, X_{\text{in}})^{-1} Y_{\text{out}},\; \frac{\partial^2 k(\boldsymbol{x}_*,\, \boldsymbol{x}_*)}{\partial \boldsymbol{x}_* \partial \boldsymbol{x}_*^T} - \frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})}{\partial \boldsymbol{x}_*} K^{-1} \frac{\partial \boldsymbol{k}(X_{\text{in}},\, \boldsymbol{x}_*)}{\partial \boldsymbol{x}_*}\right) \tag{2.104}$$

2.7.2 Expected Squared Derivative

If we pass a Gaussian random variable through a linear function then we know that the output variance is proportional to the square of the gradient of the function,

$$\mathbb{V}_x[\, g(\boldsymbol{x})\,] = \frac{\partial g}{\partial \boldsymbol{x}} \Sigma_x \frac{\partial g}{\partial \boldsymbol{x}}^T \qquad \text{where} \qquad g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}, \qquad \text{and} \qquad \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_x,\, \Sigma_x) \tag{2.105}$$

If the derivative of $g(\boldsymbol{x})$ is uncertain then the expected variance is given by,

$$\mathbb{E}_g\left[\, \mathbb{V}_x[\, g(\boldsymbol{x})\,]\right] = \text{tr}\left\{\Sigma_x\, \mathbb{V}_g\left[\frac{\partial g}{\partial \boldsymbol{x}}\right]\right\} + \mathbb{E}_g\left[\frac{\partial g}{\partial \boldsymbol{x}}\right] \Sigma_x\, \mathbb{E}_g\left[\frac{\partial g}{\partial \boldsymbol{x}}\right]^T \tag{2.106}$$

$$= \text{tr}\left\{\Sigma_x \left(\frac{\partial^2 k(\boldsymbol{x}_*,\, \boldsymbol{x}_*)}{\partial \boldsymbol{x}_* \partial \boldsymbol{x}_*^T} - \frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X_{\text{in}})}{\partial \boldsymbol{x}_*} K^{-1} \frac{\partial \boldsymbol{k}(X_{\text{in}},\, \boldsymbol{x}_*)}{\partial \boldsymbol{x}_*} + \frac{\partial \bar{f}_*}{\partial \boldsymbol{x}_*} \frac{\partial \bar{f}_*}{\partial \boldsymbol{x}_*}^T\right)\right\} \tag{2.107}$$

Which we can recognise as the same as $\tilde{\Sigma}(\boldsymbol{x}_*)$ from equation 2.73. This gives us an additional view on the inclusion of the derivative uncertainty in NIGP: it is equivalent to computing the expected square derivative rather than the square expected derivative when propagating the input noise to the output.

Chapter 3

Gaussian Process State Space Models

3.1 Chapter Overview

Learning from data is a powerful way to gain understanding of dynamical systems, which is a prerequisite for prediction and control. Learning in the ubiquitous linear dynamical system with Gaussian noise can be treated analytically, without approximation, using either direct gradient descent coupled with Kalman filtering or the Expectation Maximisation (EM) algorithm with Kalman smoothing. Unfortunately, linear dynamical systems have limited expressiveness, and are too simplistic for many applications.

Nonlinear extensions allowing learning from noisy data are not currently very mature, although this area is receiving a lot of attention. The computations required for nonlinear models are generally intractable, so approximations are used, based on linearisations (EKF, UKF (Julier and Uhlmann, 1997; Wan and Van Der Merwe, 2000; Ko et al., 2007)), moment matching (Ghahramani and Roweis, 1999; Roweis and Ghahramani; Deisenroth et al., 2009; Turner et al., 2010), or sampling (Kantas et al., 2009; Cappé et al., 2005). Many of these models rely on specifying a set of *basis functions* which implement the nonlinearities, but identifying suitable basis functions is often difficult. Furthermore, for systems of moderate complexity and non-negligible noise, it is very unlikely that we will be able to determine the system dynamics with complete (or near complete) certainty. This suggests that it is desirable to take a Bayesian approach to capture our uncertainty about the dynamics. However, it is often too complex to integrate out all unknowns and thus some parameters are optimised.

Gaussian Processes (GPs) (Rasmussen and Williams, 2006) have become the method of choice for principled inference on functions. Recently, several authors have proposed the use of GPs in dynamical model learning (Wang et al., 2008; Ko et al., 2007; Deisenroth et al., 2009; Turner et al., 2010; Frigola et al., 2013). The parameter learning approaches adopted in these, and other relevant papers, broadly fall into three categories: approximate-analytic EM algorithms, gradient descent using a particle filter, and particle EM approaches. In this chapter, we introduce a new approximate-analytic category, direct gradient descent, and apply all four categories to the same set of tasks. We then compare these approaches both theoretically and through rigorous testing. To our knowledge, approximate-analytic and particle approaches have not been compared in this way before.

3.1.1 Chapter Outline

This chapter is fairly long as it goes into detail for a number of different modelling approaches. This outline is thus intended to guide the reader to the sections they are most interested in. Sections which contain novel material are highlighted in green.

3.2 Background: An introduction to dynamical modelling along with some historical context for the chapter

3.3 State Space Models: Sets out the formal definition of a state space model as used in the rest of the chapter

3.4 Gaussian Process State Space Models: Describes how Gaussian Processes can be used as transition models with a state space model and derives the joint probability distribution on the key variables. This section then proceeds to discuss how learning can be carried out and looks at some previous approaches under the following sub-sections:

3.4.1 Learning in Gaussian Process State Space Models: Demonstrates how learning in the Gaussian Process state space model is intractable and shows how a conditional independence approximation can be used to render the calculations tractable. This approximation requires the introduction of 'pseudo-data' (a.k.a. inducing points) and this concept is explored here.

3.4.2 Previous Work with Gaussian Process State Space Models: Discusses a number of models proposed in the literature:

i. Gaussian Process Latent Variable Model (GP-LVM) (Lawrence and Moore, 2007)

ii. Gaussian Process Dynamical Model (GPDM) (Wang et al., 2008)

iii. GP-BayesFilterLearn (Ko and Fox, 2011)

iv. Variational GP Dynamical System (Damianou et al., 2011)

v. Gaussian Process Inference and Learning (GPIL) (Turner et al., 2010)

3.4.3 The Fully Bayesian Approach: A key piece of recent work (Frigola et al., 2013) shows how learning can be carried out in the GP state space model using a particle MCMC approach.

3.4.4 The Variational Approach: Introduces a variational approach to learning in GP state space models, building on work in Frigola et al. (2014). We take their sampling-based approach and develop an analytic variational approach instead. We test this novel methodology on some one dimensional problems.

3.5 The Direct Method: Introduces a completely novel approach to learning in GP state space models, which only uses forward-filtering, based upon Gaussian moment-matching approximations. Initial analysis of the algorithm is carried out in the corresponding sub-section (3.5.1).

3.6 Analytic Expectation Maximisation: Discusses in depth an alternative approximate-analytic approach based on the EM algorithm. A number of different methods for solving the E step are investigated:

3.6.1 E step using assumed density smoothing (ADS): a simple approach previously introduced in the literature (Turner et al., 2010) but which suffers from a key weakness which we identify and explain.

3.6.2 E step using Expectation Propagation (EP): explores using EP as a more sophisticated alternative to ADS in the E step. We show how a previously published method (Deisenroth and Mohamed, 2012) is flawed and introduce some simple sampling steps to avoid this flaw.

3.6.3 Comparison of E Steps: a theoretical and empirical comparison of the discussed E step methods

3.6.4 M Step: discusses the M step, as presented in Turner et al. (2010)

3.6.5 Analysis: provides a thorough analysis of the different EM algorithms discussed in the section.

3.7 Particle Filter shows how to apply the sequential Monte Carlo algorithm of Poyiadjis et al. (2011) to GP state space models, and combines it with a derivative-only, BFGS optimisation algorithm to produce a viable method. This section also provides some initial analysis of this approach.

3.8 Particle EM: explains the particle EM methodology and shows how to apply the particle Gibbs algorithm of Lindsten et al. (2012) to the GP state space model problem. A novel comparison of the particle E step with the previously derived approximate-analytic methods is provided along with some analysis of the algorithm's effectiveness for solving problems of the type considered in this chapter. Finally, there is a brief introduction to to particle stochastic approximation EM, which may provide benefits, although this algorithm is taken no further here.

3.9 Comparison: a thorough and completely novel comparison of most of the methods discussed in this chapter in terms of their theory, computational efficiency, parameter estimation performance, and their predictive performance.

3.10 Conclusion: ties the material of the chapter together and summarises the main results

3.2 Background

The task of determining a system's dynamics from observed data over time is fundamental to many fields. This is because obtaining a model of a system is a prerequisite for making predictions about the system and its evolution and, beyond that, for designing controllers to shape the system's responses to our wishes. Thus this is a problem which appears in control engineering, from mechanical to chemical systems, but also more widely in communications, biology, medicine, economics, astronomy, climatology, and so on.

As one might expect, given the broad application area, there are many different approaches to building models of systems. Many of these approaches rely on the application of scientific laws to build a model. This is most prevalent for mechanical systems, which can often be readily analysed using the laws of Newtonian physics. However, to do so even for simple mechanical real-world systems mostly requires making a number of modelling simplifications or approximations. For example, modelling friction is notoriously hard to do, the most accurate models we have today are very complex and computationally heavy to run (Olsson et al., 1998). In other fields, such as biology or economics, the

underlying principles are less well understood and so building models based on them is significantly harder.

In both of these cases an alternative modelling strategy is to build a model based on observed data from the system. Typically a class of models is presupposed and the observations are used to select a model from this class, for example, by estimating some parameters. In this chapter, we denote such model parameters with $\boldsymbol{\theta}$. This approach, often termed 'black-box modelling' allows you to model complex systems without having to understand the underlying equations first. However, in many cases this modelling methodology is complicated by not being able to observe all the relevant quantities with perfect accuracy. Indeed, it may be that one cannot measure a particular quantity at all. In these situations we need to not only model the system of interest but also our data collection mechanisms, so that we can take their shortcomings into account. Because of this, modelling strategies can often be split into a *state estimation* step and a *parameter estimation* step. In Bayesian reasoning this corresponds to finding posteriors on the state and model parameters. Here we define the concept of the system 'state' to mean the collection of variables which contain all the information to define how the system will respond to the present inputs without reference to previous inputs or behaviour. We shall denote the system state at a particular time t by the column vector $\boldsymbol{x}_t$. As we have just mentioned, it is often the case that we only partially observe the system state and/or we observe some of the variables inaccurately, for example our measurements are corrupted by noise. We will denote an observation of the system state made at time t by $\boldsymbol{y}_t$.

State estimation involves using the current model of the system to estimate the true states $\boldsymbol{x}$ based on measurements $\boldsymbol{y}$, whereas parameter estimation is focussed on using the current estimate of the unobserved states to fit model parameters. These steps together are typically referred to as the dual estimation problem (Nelson, 2000).

If we restrict ourselves to only considering the class of linear models, then this is a problem for which a very large body of work exists (Kalman, 1963; Ghahramani and Hinton, 1996; Zhou et al., 1996). If we further assume that the state is observed with additive Gaussian noise corruptions, then the state estimation problem for linear models is solved optimally (in the maximum likelihood sense) by the Kalman filter. *Filtering* is the term given to the task of estimating the system state at a particular time point using current and previous measurements. In the language of probability theory this can be written as finding $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})$, where $\boldsymbol{y}_{1:t}$ implies all the measurements up to and including time t. In offline settings, where we can use the information from future measurements to update our estimate of the state at a particular time point, the Kalman filter can be combined with a backwards-in-time step to obtain an improved estimate of the state—this is referred to as *smoothing*. From a probabilistic point of view we can write smoothing as the task of finding $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, where T is the time of the final measurement. Once we have obtained the smoothing estimate of the unknown states it is straightforward to fit a linear model to this data, for example one method is to use linear least squares (Ljung, 1998).

Unfortunately, whilst linear models are extremely attractive due to their mathematical properties they are far too simplistic to explain most real-world systems: they can only capture exponential decay/rise responses and constant-period oscillatory behaviour. On the other hand, most nonlinear systems are too complex to have analytic solutions to the dual estimation problem. Therefore, we mostly seek approximate solutions to state and parameter estimation in nonlinear systems. For example, by appealing to the Taylor series expansion of a nonlinear function it can be argued that a linear model can

still be a sufficiently accurate *local* model of a nonlinear system. This is the principle of the Extended Kalman Filter (EKF), which is perhaps the simplest method of state estimation in nonlinear systems. In the EKF, the current nonlinear model is linearised about the current estimate of the state. This linearisation is then used to estimate the next state and subsequently the model parameters. Whereas for the linear Gaussian case the Kalman filter gave the optimal estimate, in the nonlinear case the EKF gives a suboptimal approximation to the true smoothing distribution. However, the EKF has a number of unappealing properties (Wan et al., 1999), which led to the development of alternative methods of approximate nonlinear state and parameter estimation, such as the Unscented Kalman Filter (UKF) (Julier and Uhlmann, 1997; Wan et al., 1999) and the assumed density filter (ADF). In addition to these approximate-analytic methods there is a large body of work tackling this problem using sampling methods. Central to these approaches is the Sequential Monte Carlo (SMC) algorithm, commonly referred to as Particle Filtering (Lindsten, 2013c; Schön et al., 2011).

A further key difficulty with data-driven modelling approaches for nonlinear systems is how to choose the class of models to fit to the data. If the model class is too simplistic then the resulting model will be a poor approximation to the real system; if the class is too complex then estimating the model parameters will be significantly more difficult and we are prone to overfitting. Previously studied model classes include Hammerstein-Wiener systems (Bai, 1998; Hunter and Korenberg, 1986), artificial neural networks (Narendra and Parthasarathy, 1990), and radial basis functions (Ghahramani and Roweis, 1999), amongst others. For a complete overview of this field see, for example, Ljung (1998); Sjöberg et al. (1995).

A possible answer to the problem of model class selection can be found in the area of Bayesian nonparametric models. These are a class of models which are parameterised directly by the data and so can adapt their complexity to the observations. The assumptions made by these models are typically on a different level to those made by parametric model classes, for example assumptions of smoothness or periodicity rather than of model shape or order. Perhaps the most famous Bayesian nonparametric model is the Gaussian Process, discussed in section 1.4. There are many advantages of using a GP for modelling nonlinear dynamical systems, for example, their flexibility and tractability. They are also firmly built upon a full probabilistic interpretation of the data, which allows uncertainty to be explicitly accounted for and modelled. This is particularly important in the situation where there is a limited amount of data available to model the system. In this case there are likely to be a large number of possible models, which fit the data. A probabilistic model can capture this uncertainty, whereas a deterministic model, which only provides one possible explanation for the data, cannot. Furthermore, it can often be the case that the model is asked to make a prediction at an input location where it has very little (or even no) data in the vicinity. A deterministic model would make a prediction without any indication that it is extremely untrustworthy. Even some probabilistic models, such as Bayesian radial basis functions (see section 1.4.5), cannot represent this uncertainty without nearby data points. A GP, however, does capture the uncertainty, which makes it a very attractive choice. Unfortunately, we cannot immediately apply a GP model to this problem: as discussed in chapter 2, we must take care in situations where we cannot observe the data directly.

This chapter discusses and analyses a number of different methods and algorithms for fitting Gaussian Process dynamical models to observed data. We start by defining what we mean by a state space model and a GP state space model (GP-SSM).

3.3 State Space Models

We consider a continuous-state, discrete-time dynamical system which is in the unobserved state $\boldsymbol{x}_t$ at time t. We make a sequence of noisy measurements, $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_T\}$ (shorthand $\boldsymbol{y}_{1:T}$), and from these we wish to build a model of the system. We may also apply controls $\boldsymbol{u}_t$ which can be functions of the noisy measurement $\boldsymbol{y}_t$ (they may also depend on previous measurements). This setup is shown in figure 3.1. This task is central to many areas of engineering, statistics, econometrics, and biology, which all contain problems of this form.

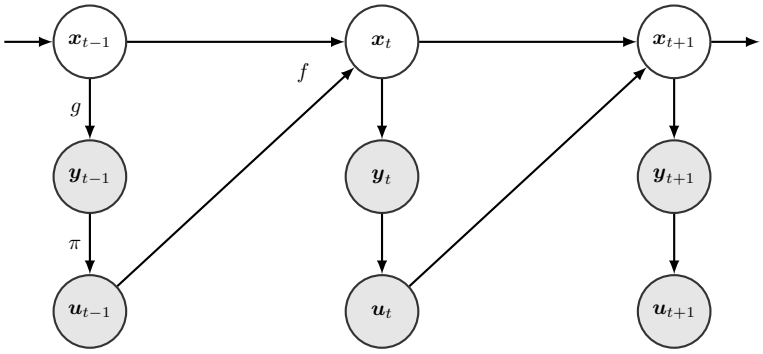


Figure 3.1: Graphical model of the dynamical system problem. A system transitions between hidden states $\boldsymbol{x}$ according to an unknown nonlinear transition function f. We make observations $\boldsymbol{y}$ which are related to the latent state via g which is also unknown. We can influence the system trajectory by applying controls $\boldsymbol{u}$ which can depend on the observations via a policy π. For simplicity, the controls are shown as depending on only the observation at the current time step.

Figure 3.1 is an example of a state space model (SSM), which assumes that the system dynamics and the observation are fully determined by the current state $\boldsymbol{x}_t$. We assume time invariant transition and observation functions, f and g, such that the dynamics can be written as

$$\begin{aligned} \boldsymbol{x}_t &= f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}_t \\ \boldsymbol{y}_t &= g(\boldsymbol{x}_t, \boldsymbol{\theta}) + \boldsymbol{\nu}_t \\ \boldsymbol{u}_t &= \pi(\boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \end{aligned} \tag{3.1}$$

where $\boldsymbol{\epsilon}$ and $\boldsymbol{\nu}$ are both noise vectors, ($\boldsymbol{\epsilon}$ is typically called the process noise and $\boldsymbol{\nu}$ the observation noise), and $\boldsymbol{\theta}$ is a parameter vector. For brevity we will mostly drop the explicit conditioning on the controls. In chapter 2 we looked at regression models, mapping a set of inputs to an output. Here each transition can be considered as a similar mapping, however the output at one time step then becomes the input at the next. Hence we replace the 'in' and 'out' subscript notation from that chapter with the time index.

The analytic methods we discuss later are only tractable with a Gaussian noise model,

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \Sigma_\epsilon), \qquad \boldsymbol{\nu} \sim \mathcal{N}(0, \Sigma_\nu) \tag{3.2}$$

Given that we are free to define the state $\boldsymbol{x}$ as we wish, it is always possible to set it such that the observation function is linear by augmenting the state space to include the measurement nonlinearities. For example, suppose we have a state, $\tilde{\boldsymbol{x}}_t$, defined such that there is a nonlinear relationship between

the state and the observations. We can redefine the state as follows,

$$\begin{aligned} \boldsymbol{x}_t &= \left[\begin{array}{cc} \tilde{\boldsymbol{x}}_t, & g(\tilde{\boldsymbol{x}}_t) \end{array} \right]^T \\ \boldsymbol{y}_t &= \left[\begin{array}{cc} 0 & 0 \\ 0 & I \end{array} \right] \boldsymbol{x}_t + \boldsymbol{\nu}_t \end{aligned} \tag{3.3}$$

which results in a new, linear observation function. We will take this approach for the rest of this chapter as it simplifies much of the following analysis. Thus,

$$g(\boldsymbol{x}) = C\boldsymbol{x} \tag{3.4}$$

and we can write the observation probability as,

$$p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y}_t;\ C\boldsymbol{x}_t,\ \Sigma_\nu) \tag{3.5}$$

Learning in a state space model involves identifying the transition function and the parameters $\boldsymbol{\theta}$. These include the variance of the process and observation noise as well as the parameters of the observation model, C, and the transition function, $\boldsymbol{\theta}_{\text{trans}}$, which we look at in the next section.

$$\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{trans}},\ C,\ \Sigma_\epsilon,\ \Sigma_\nu\} \tag{3.6}$$

3.4 Gaussian Process State Space Models

Gaussian Processes (GPs) (Rasmussen and Williams, 2006) are a highly flexible model class, for example allowing us to perform inference on all functions of a certain smoothness. As a stochastic process, they also capture model uncertainty, returning a predictive probability distribution rather than a point estimate. We therefore place a Gaussian Process prior on the transition function $f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}, \boldsymbol{\theta})$,

$$f \sim \mathcal{GP}(m, k) \tag{3.7}$$

where $m(\boldsymbol{x})$ is the GP mean function, usually either fixed to zero or a linear function, and $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the GP covariance function. For tractability, the analytic methods discussed below require the use of covariance functions which can be integrated against a Gaussian. This restricts our choice of k to covariance functions such as linear, polynomial, and Gaussian (also known as squared exponential or 'exponentiated quadratic'). For the rest of this chapter we will use the Gaussian kernel,

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_f^2 \exp\left[-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \Lambda^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)\right] \tag{3.8}$$

where σ_f and Λ are hyperparameters to be optimised: we add them to the parameter vector $\boldsymbol{\theta}$. For simplicity we will also set the GP mean function to zero, although this is not a requirement. We introduce the random variable $\boldsymbol{f}_t$ to represent the GP function value evaluated at $\boldsymbol{x}_{t-1}$,

$$\begin{aligned} \boldsymbol{f}_t &= f(\boldsymbol{x}_{t-1},\ \boldsymbol{u}_{t-1}, \boldsymbol{\theta}) \\ \boldsymbol{x}_t &= \boldsymbol{f}_t + \boldsymbol{\epsilon}_t \ \Rightarrow\ p(\boldsymbol{x}_t \mid \boldsymbol{f}_t) = \mathcal{N}(\boldsymbol{f}_t,\ \Sigma_\epsilon) \end{aligned} \tag{3.9}$$

By using a GP we have placed a very flexible prior on the space of dynamical models, which can capture a very broad range of behaviours. Figure 3.2 shows, on the top left, an example GP prior with a number of sampled transition functions. State trajectories resulting from these transition functions are shown in the other three plots. The top right plot shows a trajectory undergoing exponential decay as one might expect from a linear function and, indeed, looking at the part of the transition function being used in the top left plot shows that it is close to linear. The bottom left plot shows a trajectory with oscillations growing over time but then saturating, which is a nonlinear feature. The final trajectory shows something resembling a limit cycle, although the cycles are not exact repeats of each other. All of these behaviours, and more, can be modelled with a GP state space model, which makes them a very powerful class of models.

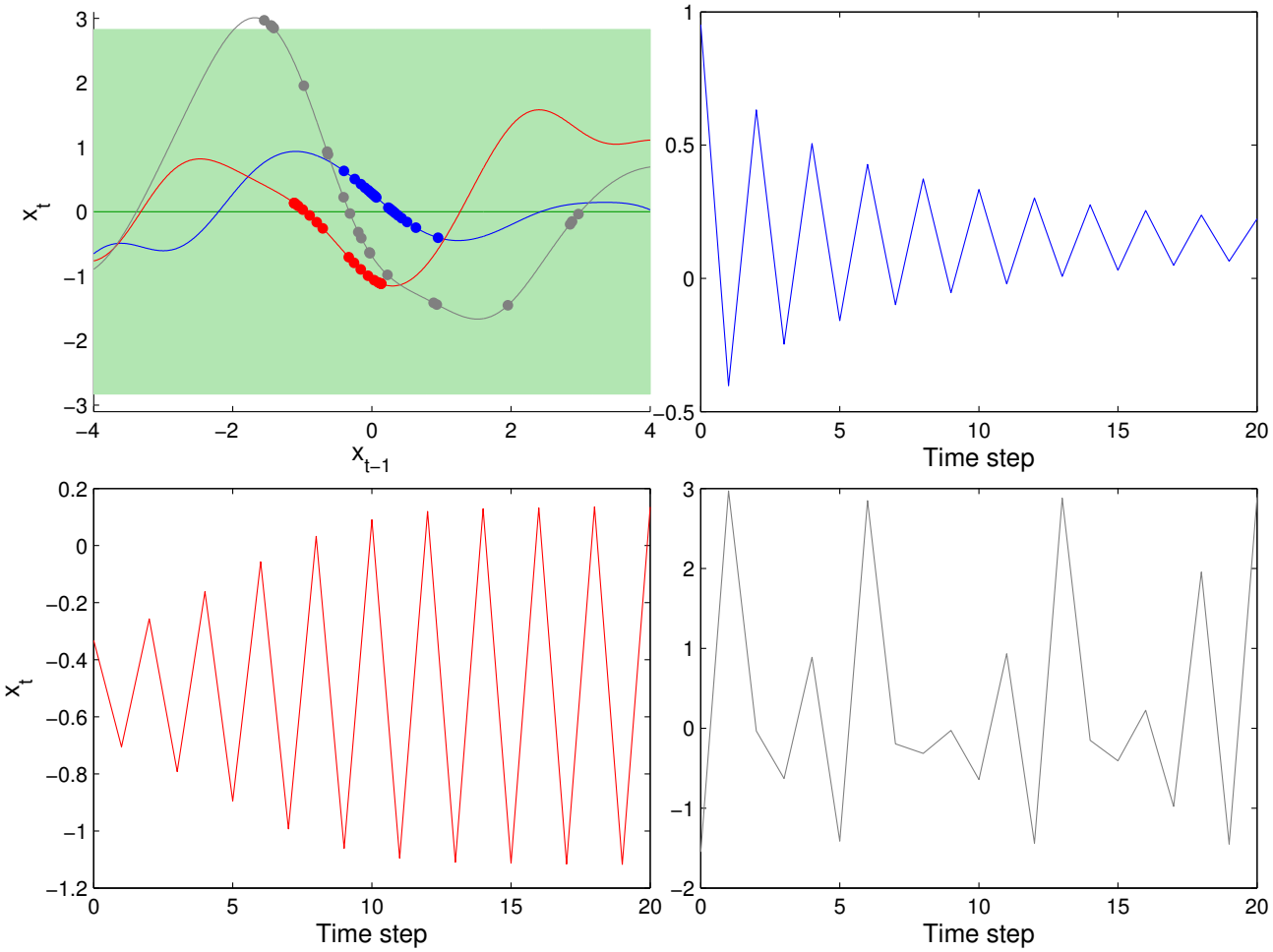


Figure 3.2: Example transition functions and state trajectories generated from a noise-free, 1D GP state space model with a particular fixed set of hyperparameters. The top left plot shows a simple GP prior on transition functions, with a mean of zero and a standard deviation of 1. The blue, red, and grey lines are sampled transition functions from this prior. The other three plots show state trajectories which are generated from the correspondingly coloured transition functions. The dots on the top left plot show where each transition lies.

For a D dimensional state vector we use D independent GPs to model the mapping from the current state and controls to each of the state variables at the next time step. Each of these GPs has a separate set of hyperparameters. Thus, a priori, the GP function value $\boldsymbol{f}_t$ is distributed according to

a D dimensional Gaussian with diagonal covariance,

$$p(\boldsymbol{f}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(\boldsymbol{f}_t;\; \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix},\; \begin{bmatrix} k_1(\hat{\boldsymbol{x}}_{t-1}, \hat{\boldsymbol{x}}_{t-1}) & & \\ & \ddots & \\ & & k_D(\hat{\boldsymbol{x}}_{t-1}, \hat{\boldsymbol{x}}_{t-1}) \end{bmatrix}\right) \tag{3.10}$$

where,

$$\hat{\boldsymbol{x}}_{t-1} \;=\; [\boldsymbol{x}_{t-1}\; \boldsymbol{u}_{t-1}]^T \tag{3.11}$$

and the subscripts on the covariance functions indicates the different GPs for each output dimension. To keep the notation as clear as possible we will mostly drop the controls from the expressions. Whenever the latent state $\boldsymbol{x}$ is considered as an input to the GP it is implied that the controls are concatenated with the state.

If we wish we can introduce correlation between the different dimensions of the state variable transition, a problem often termed multi-task learning. Two examples of how to do this are by placing a separate covariance function on the GP outputs (i.e. on the columns of the GP training targets matrix) (Rakitsch et al., 2013) or by using a multiple-output kernel (Melkumyan and Ramos, 2011). In our experience, using these extra covariance terms gave little benefit on the problems we considered, and added extra complexity. We shall therefore maintain the independent GP model in this chapter.

We can draw a more detailed graphical model to represent a Gaussian Process state space model (GP-SSM) as shown in figure 3.3. The thick line connecting the GP function values indicates that these variables are fully connected.

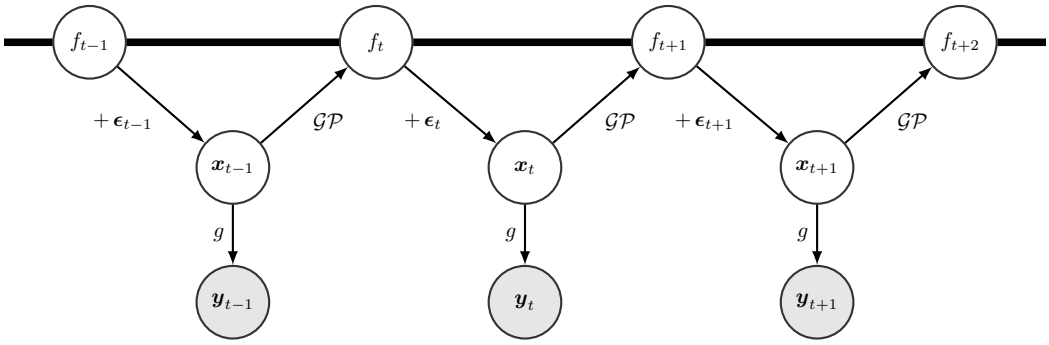


Figure 3.3: Graphical model of a Gaussian Process state space model. The random variables f represent the GP function values, which are fully connected to each other, as represented by the thick line. The transition of $\boldsymbol{x}_{t-1}$ to $\boldsymbol{x}_t$ is modelled by the GP followed by the addition of process noise $\boldsymbol{\epsilon}_t$

If we start with a prior distribution on the first latent state $\boldsymbol{x}_1$ then we can form the joint distribution between $\boldsymbol{x}$ and $\boldsymbol{f}$ over a complete trajectory of length T by proceeding up the chain of latent variables, $\boldsymbol{x}_{t-1} \;\rightarrow\; \boldsymbol{f}_t \;\rightarrow\; \boldsymbol{x}_t$,

$$p(\boldsymbol{x}_{1:T},\, \boldsymbol{f}_{2:T} \mid \boldsymbol{\theta}) \;=\; p(\boldsymbol{x}_1)\, p(\boldsymbol{f}_2 \mid \boldsymbol{x}_1)\, p(\boldsymbol{x}_2 \mid \boldsymbol{f}_2)\, p(\boldsymbol{f}_3 \mid \boldsymbol{x}_1,\, \boldsymbol{x}_2, \boldsymbol{f}_2)\, p(\boldsymbol{x}_3 \mid \boldsymbol{f}_3)\, p(\boldsymbol{f}_4 \mid \boldsymbol{x}_1,\, \boldsymbol{x}_2,\, \boldsymbol{x}_3,\, \boldsymbol{f}_2,\, \boldsymbol{f}_3) \ldots \tag{3.12}$$

$$\;=\; p(\boldsymbol{x}_1) \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1},\, \boldsymbol{f}_{2:t-1})\, p(\boldsymbol{x}_t \mid \boldsymbol{f}_t) \tag{3.13}$$

where we have dropped $\boldsymbol{\theta}$ from each of the distributions on the right hand side. Note that,

$$\prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1}, \boldsymbol{f}_{2:t-1}, \boldsymbol{\theta}) \;\neq\; p(\boldsymbol{f}_{2:T} \mid \boldsymbol{x}_{1:T}, \boldsymbol{\theta}) \tag{3.14}$$

This is because $p(\boldsymbol{f}_{2:T} \mid \boldsymbol{x}_{1:T}, \boldsymbol{\theta})$ implies Gaussian Process regression where $\boldsymbol{x}_{1:T-1}$ are inputs and $\boldsymbol{x}_{2:T}$ are outputs, whereas the product term in equation 3.14 implies GP regression where $\boldsymbol{f}_{2:T}$ are the outputs (the inputs are still $\boldsymbol{x}_{1:T-1}$). In the first case (right hand side of equation 3.14) the outputs are corrupted by process noise but in the second case (left hand side) they are noiseless — we are conditioning directly on the GP latent function value. Thus we can write,

$$p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1}, \boldsymbol{f}_{2:t-1}, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(\boldsymbol{f}_t;\; \boldsymbol{\mu}_{f_t},\; \Sigma_{f_t}\right) \tag{3.15}$$

where

$$\boldsymbol{\mu}_{f_t} \;=\; \begin{bmatrix} \boldsymbol{k}_1(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{1:t-2})\, K_1(\boldsymbol{x}_{1:t-2}, \boldsymbol{x}_{1:t-2})^{-1} \\ \vdots \\ \boldsymbol{k}_D(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{1:t-2})\, K_D(\boldsymbol{x}_{1:t-2}, \boldsymbol{x}_{1:t-2})^{-1} \end{bmatrix} \boldsymbol{f}_{2:t-1} \tag{3.16}$$

$$\Sigma_{f_t} \;=\; \operatorname{diag}\left\{k_d(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-1}) \;-\; \boldsymbol{k}_d(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{1:t-2})\, K_d(\boldsymbol{x}_{1:t-2}, \boldsymbol{x}_{1:t-2})^{-1} \boldsymbol{k}_d(\boldsymbol{x}_{1:t-2}, \boldsymbol{x}_{t-1})\right\}_{d=1}^{D} \tag{3.17}$$

We can now find a representation for the product term in equation 3.13. We can see from equations 3.15 to 3.17, and considering just one output dimension, that the first two terms in the product are,

$$p(f_2 \mid \boldsymbol{x}_1, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(f_2;\; 0,\; k(\boldsymbol{x}_1, \boldsymbol{x}_1)\right) \tag{3.18}$$

$$p(f_3 \mid \boldsymbol{x}_{1:2}, f_2, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(f_3;\; k(\boldsymbol{x}_2, \boldsymbol{x}_1)\, k(\boldsymbol{x}_1, \boldsymbol{x}_1)^{-1} f_2,\; k(\boldsymbol{x}_2, \boldsymbol{x}_2) - k(\boldsymbol{x}_2, \boldsymbol{x}_1)\, k(\boldsymbol{x}_1, \boldsymbol{x}_1)^{-1} k(\boldsymbol{x}_1, \boldsymbol{x}_2)\right) \tag{3.19}$$

from which it is clear that the distributions are linked by previous fs appearing in the mean of the distribution for the current f. If,

$$a \;\sim\; \mathcal{N}(a;\; \mu_a,\; \Sigma_a) \qquad \text{and} \qquad b \;\sim\; \mathcal{N}(b;\; c\,a,\; \Sigma_b) \tag{3.20}$$

then,

$$\mathbb{E}_{a,b}[b] \;=\; c\,\mu_a, \qquad \mathbb{V}_{a,b}[b] \;=\; \Sigma_b \;+\; c^2\,\Sigma_a, \qquad \mathbb{C}_{a,b}[a, b] \;=\; c\,\Sigma_a \tag{3.21}$$

from which we can immediately realise,

$$\mathbb{E}_{f_{2:t}}\left[\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1}, \boldsymbol{f}_{2:t-1}, \boldsymbol{\theta}\right] \;=\; \mathbf{0} \tag{3.22}$$

We can also see that,

$$\begin{aligned} \mathbb{V}_{f_{2:3}}[f_3 \mid \boldsymbol{x}_{1:2}, f_2, \boldsymbol{\theta}] \;&=\; k(\boldsymbol{x}_2, \boldsymbol{x}_2) - k(\boldsymbol{x}_2, \boldsymbol{x}_1)\, k(\boldsymbol{x}_1, \boldsymbol{x}_1)^{-1} k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &\qquad + \; k(\boldsymbol{x}_2, \boldsymbol{x}_1)\, k(\boldsymbol{x}_1, \boldsymbol{x}_1)^{-1} k(\boldsymbol{x}_1, \boldsymbol{x}_1)\, k(\boldsymbol{x}_1, \boldsymbol{x}_1)^{-1} k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &=\; k(\boldsymbol{x}_2, \boldsymbol{x}_2) \end{aligned} \tag{3.23}$$

and

$$\begin{aligned}\mathbb{C}_{f_{2:3}}[f_3 \mid \boldsymbol{x}_{1:2},\, f_2,\, \boldsymbol{\theta};\; f_2 \mid \boldsymbol{x}_1,\, \boldsymbol{\theta}] &= k(\boldsymbol{x}_2, \boldsymbol{x}_1)\, k(\boldsymbol{x}_1, \boldsymbol{x}_1)^{-1} k(\boldsymbol{x}_1, \boldsymbol{x}_1) \\ &= k(\boldsymbol{x}_2, \boldsymbol{x}_1) \end{aligned} \tag{3.24}$$

which shows that,

$$\prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1},\, \boldsymbol{f}_{2:t-1},\, \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{f}_{2:T};\; \mathbf{0},\; K(\boldsymbol{x}_{1:T-1}, \boldsymbol{x}_{1:T-1})\right) \tag{3.25}$$

where we've assumed that each element of $K(\boldsymbol{x}_{1:T-1}, \boldsymbol{x}_{1:T-1})$ is actually a diagonal matrix with one entry for each output dimension. Thus,

$$p(\boldsymbol{x}_{1:T},\, \boldsymbol{f}_{2:T} \mid \boldsymbol{\theta}) = p(\boldsymbol{x}_1)\, \mathcal{N}\left(\boldsymbol{x}_{2:T};\; \boldsymbol{f}_{2:T},\; I_D \otimes \Sigma_\epsilon\right) \mathcal{N}\left(\boldsymbol{f}_{2:T},\; \mathbf{0},\; K(\boldsymbol{x}_{1:T-1}, \boldsymbol{x}_{1:T-1})\right) \tag{3.26}$$

where $I_D \otimes \Sigma_\epsilon$ produces a block diagonal matrix with each block being a copy of the process noise covariance matrix. We can integrate out the GP latent variables $\boldsymbol{f}_{2:T}$ to get,

$$p(\boldsymbol{x}_{1:T} \mid \boldsymbol{\theta}) = p(\boldsymbol{x}_1)\, \text{“}\mathcal{N}\text{”}\left(\boldsymbol{x}_{2:T};\; \mathbf{0},\; K(\boldsymbol{x}_{1:T-1}, \boldsymbol{x}_{1:T-1}) + I_D \otimes \Sigma_\epsilon\right) \tag{3.27}$$

Although this appears to have a Gaussian form $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{\theta})$ is **not** Gaussian (hence the quotes) due to the presence of $\boldsymbol{x}$ inside the kernel which makes up the covariance matrix of equation 3.27. For example, although for a stationary covariance function such as the Gaussian kernel,

$$\begin{aligned} p(\boldsymbol{x}_2 \mid \boldsymbol{x}_1, \boldsymbol{\theta}) &= \mathcal{N}\left(\boldsymbol{x}_2;\; \mathbf{0},\; k(\boldsymbol{x}_1, \boldsymbol{x}_1)\right) \\ &= \mathcal{N}\left(\boldsymbol{x}_2;\; \mathbf{0},\; \sigma_f^2\right) \end{aligned} \tag{3.28}$$

$$\begin{aligned} p(\boldsymbol{x}_3 \mid \boldsymbol{x}_2, \boldsymbol{x}_1, \boldsymbol{\theta}) &= \mathcal{N}\left(\boldsymbol{x}_3;\; \mathbf{0},\; k(\boldsymbol{x}_1, \boldsymbol{x}_1) - k(\boldsymbol{x}_2, \boldsymbol{x}_1)\, k(\boldsymbol{x}_1, \boldsymbol{x}_1)^{-1} k(\boldsymbol{x}_1, \boldsymbol{x}_2)\right) \\ &= \mathcal{N}\left(\boldsymbol{x}_3;\; \mathbf{0},\; \sigma_f^2 - \sigma_f^{-2} k(\boldsymbol{x}_1, \boldsymbol{x}_2)^2\right) \end{aligned} \tag{3.29}$$

are both Gaussian, it is clear that the marginal $p(\boldsymbol{x}_3 \mid \boldsymbol{\theta})$ is not due to the position of $\boldsymbol{x}_2$ and $\boldsymbol{x}_1$ in equation 3.29. Therefore, for the moment, we will refrain from integrating out f. We can write the joint probability of all the variables in our model,

$$\begin{aligned} p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T} \mid \boldsymbol{\theta}) &= p(\boldsymbol{y}_{1:T} \mid \boldsymbol{x}_{1:T}, \boldsymbol{\theta})\, p(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T} \mid \boldsymbol{\theta}) \\ &= \mathcal{N}\left(\boldsymbol{y}_{1:T};\; \hat{C}\, \boldsymbol{x}_{1:T},\; I_D \otimes \Sigma_\nu\right) p(\boldsymbol{x}_1 \mid \boldsymbol{\theta}) \mathcal{N}\left(\boldsymbol{x}_{2:T};\; \boldsymbol{f}_{2:T},\; I_D \otimes \Sigma_\epsilon\right) \mathcal{N}\left(\boldsymbol{f}_{2:T},\; \mathbf{0},\; K(\boldsymbol{x}_{1:T-1}, \boldsymbol{x}_{1:T-1})\right) \end{aligned} \tag{3.30}$$

where $\hat{C}$ is a $TE \times D$ matrix of the linear weights of the observation function ($g(\boldsymbol{x}) = C\boldsymbol{x}$) replicated T times.

3.4.1 Learning in Gaussian Process State Space Models

In a Bayesian setting, learning in the Gaussian Process state space model (GP-SSM) means finding a posterior on the parameters given the observed data, $p(\boldsymbol{\theta} \mid \boldsymbol{y}_{1:T})$. Unfortunately, but somewhat unsurprisingly, this is intractable, although a sampling based method to do this does exist (Frigola et al., 2013), which is discussed in section 3.4.3. An alternative is to apply maximum marginal

likelihood optimisation to find a set of parameters. To do this analytically we need an expression for the marginal likelihood, $p(\boldsymbol{y}_{1:T} \mid \boldsymbol{\theta})$, which involves integrating out $\boldsymbol{f}_{2:T}$ and $\boldsymbol{x}_{1:T}$ from the joint distribution in equation 3.30. As we stated in the previous section, we can integrate out $\boldsymbol{f}_{2:T}$, however, the resulting non-Gaussian distribution on $\boldsymbol{x}_{1:T}$ means that we cannot integrate out $\boldsymbol{x}$ analytically. This means we cannot find a closed form expression for the marginal likelihood and thus cannot perform exact maximum marginal likelihood optimisation for the parameters either. This suggests that if we want to find an analytic method to fit the model we need to make some approximations.

Taking a step back for a moment, a very basic approach to the problem is to dispense with the time series structure and cast the task of learning the GP transition function as a regression problem between pairs of sequential states $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$. Of course we don't have access to the latent states but we could approximate $\boldsymbol{x}_t$ with $\boldsymbol{y}_t$, and find the mapping from $\boldsymbol{y}_{t-1}$ (and controls) to $\boldsymbol{y}_t$. This can be seen as a type of nonlinear, first order autoregressive model. The difficulty is that there is noise in both the inputs and the outputs, a problem sometimes called *errors-in-the-variables*. This is the situation discussed in chapter 2 and we could use the methods presented in that chapter in this context. These methods are much simpler than the approaches discussed in the rest of this section and are very wasteful of the information inherent in the structure shown in the graphical model. Still, in some cases they may work well and are likely to be much quicker to train than the methods discussed here. We test this out by including NIGP (see chapter 2), in our comparisons at the end of this chapter.

Perhaps the simplest method for tractable learning in the GP-SSM which does take into account the structure in the state space model, is to treat the latent states as extra parameters to be optimised. This is the basis for a number of methods based on the GP-LVM, as discussed in section 3.4.2. However, this approach is very undesirable as there are as many latent states as observed data points and so the number of parameters will always be larger than size of the training data set (once we factor in the other parameters such as those in the GP covariance function). Without careful regularisation these models are highly likely to overfit and lead to poor modelling performance.

Rather than trying to gain tractability by applying non-Bayesian methods to the exact model, a better set of solutions can be found by instead considering an approximate version of the state space model: by introducing some conditional independencies we can make the model tractable. To do this, we must first make a rather unintuitive step and augment the GP training data with some auxiliary variables or pseudo-points, $\{\tilde{X}, \tilde{F}\}$, in a similar manner to the FITC sparse GP approximation of Snelson and Ghahramani (2006a). We term the pseudo-targets $\tilde{F}$ to indicate that we do not consider them to be corrupted by process noise. The graphical model for this setup is shown in figure 3.4.

As an augmentation of the GP training set, the pseudo-data shares the same GP prior as the latent transitions,

$$p(\boldsymbol{f}_t, \tilde{F} \mid \boldsymbol{x}_{t-1}, \tilde{X}, \boldsymbol{\theta}) = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-1}) & k(\boldsymbol{x}_{t-1}, \tilde{X}) \\ k(\tilde{X}, \boldsymbol{x}_{t-1}) & k(\tilde{X}, \tilde{X}) \end{bmatrix}\right) \tag{3.31}$$

and thus the distribution on a latent transition variable, f_t, given the pseudo-data has the form of a GP posterior. Combining this with the results from the previous section (e.g. equation 3.15) we can

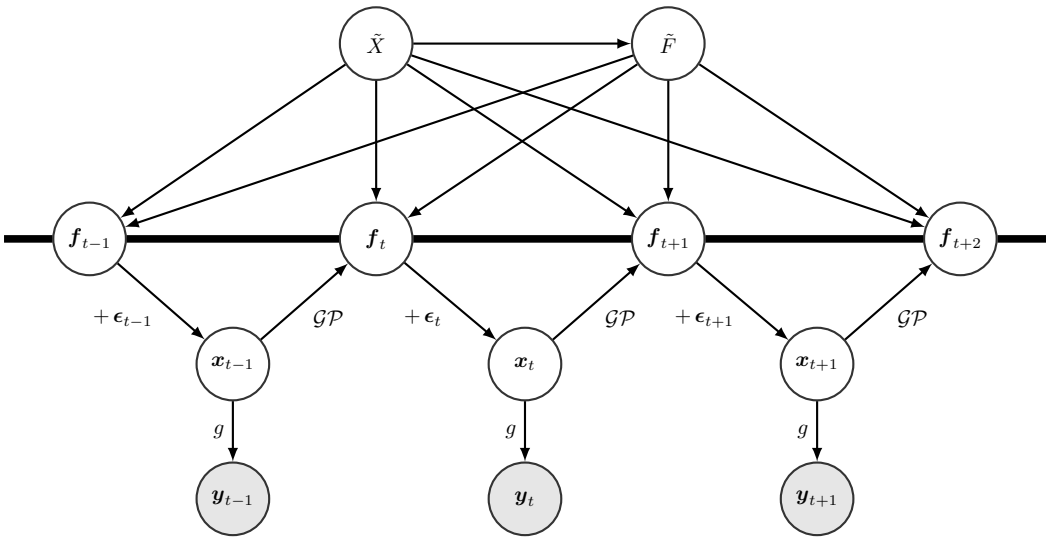


Figure 3.4: Graphical model of a Gaussian Process state space model with a set of pseudo points $\{\tilde{X}, \tilde{F}\}$. If we know the latent transitions, $f_{2:T}$ then the latent states, $\boldsymbol{x}_{2:T}$ are independent of each other and the pseudo-data as they are just process noise corrupted versions of the transition variables f. The pseudo-data, $\{\tilde{X}, \tilde{Y}\}$, induces a distribution over the latent transitions by means of the joint GP prior.

see that,

$$p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1}, \boldsymbol{f}_{2:t-1}, \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \;=\; \mathcal{N}(\mathfrak{m}(\boldsymbol{x}_{t-1}), \mathfrak{s}(\boldsymbol{x}_{t-1})) \tag{3.32}$$

where $\mathfrak{m}$ and $\mathfrak{s}$ are the GP posterior moments and we set the training set to be $\{[\tilde{X},\ \boldsymbol{x}_{1:t-2}], [\tilde{F},\ \boldsymbol{f}_{2:t-1}]\}$. We now write the joint distribution, conditioned on the pseudo-inputs,

$$p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \tilde{F} \mid \tilde{X}, \boldsymbol{\theta}) \;=\; p(\boldsymbol{y}_{1:T} \mid \boldsymbol{x}_{1:T}, \boldsymbol{\theta})\, p(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T} \mid \tilde{F}, \tilde{X}, \boldsymbol{\theta})\, p(\tilde{F} \mid \tilde{X}, \boldsymbol{\theta}) \tag{3.33}$$

with,

$$\begin{aligned} p(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T} \mid \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \;&=\; p(\boldsymbol{x}_1) \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1}, \boldsymbol{f}_{2:t-1}, \tilde{F}, \tilde{X}, \boldsymbol{\theta})\, p(\boldsymbol{x}_t \mid \boldsymbol{f}_t) \\ &=\; p(\boldsymbol{x}_1)\, \mathcal{N}\left(\boldsymbol{x}_{2:T};\ \boldsymbol{f}_{2:T},\ I_D \otimes \Sigma_\epsilon\right) \mathcal{N}\left(\boldsymbol{f}_{2:T};\ \boldsymbol{\mu}_{f|\tilde{F}},\ \Sigma_{f|\tilde{F}}\right) \end{aligned} \tag{3.34}$$

and,

$$\boldsymbol{\mu}_{f|\tilde{F}} \;=\; K(\boldsymbol{x}_{1:T-1}, \tilde{X})\, K(\tilde{X}, \tilde{X})^{-1} \tilde{F} \tag{3.35}$$

$$\Sigma_{f|\tilde{F}} \;=\; K(\boldsymbol{x}_{1:T-1}, \boldsymbol{x}_{1:T-1}) - K(\boldsymbol{x}_{1:T-1}, \tilde{X})\, K(\tilde{X}, \tilde{X})^{-1} K(\tilde{X}, \boldsymbol{x}_{1:T-1}) \tag{3.36}$$

The form of the moments in equations 3.35 and 3.36 shows how the pseudo-points play the role of extra training points, which induce a GP posterior on $\boldsymbol{f}_{2:T}$ when conditioned upon. Unfortunately, it is still intractable to integrate out $\boldsymbol{x}_{1:T}$ from equation 3.34, although we could take a variational approach to form a lower bound on the marginal likelihood from which we can integrate out all the latent variables. This approach is discussed in section 3.4.4. The difficulty for learning in equation 3.34 is that the transitions from $\boldsymbol{x}_{1:t-2}$ to $\boldsymbol{f}_{2:t-1}$ form part of the GP training set for the prediction on the transition from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{f}_t$, as shown in figure 3.5. It is intractable to integrate out the latent

states once they are acting as training data for a GP. However, the pseudo-points we have introduced are also acting as training data and can be used to specify the GP. Therefore, we could choose to approximate the GP prediction on the transition at $\boldsymbol{x}_{t-1}$ by only using the distribution induced by the pseudo-points,

$$p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1}, \boldsymbol{f}_{2:t-2}, \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \approx p(\boldsymbol{f}_t \mid \boldsymbol{x}_{t-1}, \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \tag{3.37}$$

$$= \mathcal{N}(\boldsymbol{f}_t;\ \mathfrak{m}(\boldsymbol{x}_{t-1}),\ \mathfrak{s}(\boldsymbol{x}_{t-1})) \tag{3.38}$$

where $\mathfrak{m}$ and $\mathfrak{s}$ are the GP posterior predictive mean and variance functions (see equations 1.15 and 1.16), with $\{\tilde{X},\ \tilde{F}\}$ as the training set. The approximate distribution only depends on the latent state $\boldsymbol{x}_{t-1}$, which, crucially, only acts *as a test input* and not as training data. This means that we have a Gaussian (approximate) marginal distribution for $\boldsymbol{f}_t$. By using this approximate predictive distribution we are making a conditional independence approximation,

$$p(\boldsymbol{f}_{2:T} \mid \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \approx \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \tag{3.39}$$

this 'cuts' the thick lines in figure 3.4. That is, given the pseudo training set the individual GP latent variables $\boldsymbol{f}_t$ are independent of each other. This is equivalent to saying that previous state transitions no longer affect the belief over future state transitions. From a learning point of view we can say that the GP transition function is now only determined by the values of the pseudo data set — after all, if previous transitions have no effect on the future then we cannot learn from them. Figure 3.6 shows an example of the effect of this approximation and figure 3.7 shows the new graphical model. With this approximation in place the Gaussian distribution on $\boldsymbol{f}_{2:T}$ in equation 3.34 becomes diagonal. This means that when we integrate out the fs, although the resulting joint distribution on $\boldsymbol{x}_{1:T}$ is still non-Gaussian, we can factorise it into a product of conditionals where $\boldsymbol{x}_t$ only depends on $\boldsymbol{x}_{t-1}$ and so on,

$$\begin{aligned} p(\boldsymbol{x}_{1:T} \mid \tilde{F}, \tilde{X}, \boldsymbol{\theta}) &\approx p(\boldsymbol{x}_1) \int \prod_{t=2}^{T} \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{f}_t, \boldsymbol{\theta})}_{\text{Process noise}} \underbrace{p(\boldsymbol{f}_t \mid \boldsymbol{x}_{t-1}, \tilde{F}, \tilde{X}, \boldsymbol{\theta})}_{\mathcal{GP} \text{ prediction eq. 3.38}} d\boldsymbol{f}_{2:T} \\ &= p(\boldsymbol{x}_1) \prod_{t=2}^{T} \int p(\boldsymbol{x}_t \mid \boldsymbol{f}_t, \boldsymbol{\theta})\, p(\boldsymbol{f}_t \mid \boldsymbol{x}_{t-1}, \tilde{F}, \tilde{X}, \boldsymbol{\theta})\ d\boldsymbol{f}_t \\ &= p(\boldsymbol{x}_1) \prod_{t=2}^{T} \mathcal{N}\Big(\boldsymbol{x}_t;\ k(\boldsymbol{x}_{t-1}, \tilde{X})\, K(\tilde{X}, \tilde{X})^{-1} \tilde{F}, \\ &\qquad\qquad k(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-1}) - k(\boldsymbol{x}_{t-1}, \tilde{X})\, K(\tilde{X}, \tilde{X})^{-1} k(\tilde{X}, \boldsymbol{x}_{t-1}) + \Sigma_\epsilon\Big) \end{aligned} \tag{3.40}$$

$$= p(\boldsymbol{x}_1) \prod_{t=2}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \tag{3.41}$$

Considering the conditional in equation 3.40, we can recognise that this is a standard GP prediction with the process noise variance added to the predictive variance,

$$p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1},\ \boldsymbol{y}_{1:T}) \approx p\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1},\ \tilde{X},\ \tilde{F},\ \boldsymbol{\theta}\right) = \mathcal{N}(\boldsymbol{x}_t;\ \mathfrak{m}(\boldsymbol{x}_{t-1}),\ \mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon) \tag{3.42}$$

This leads to the following approximation for the joint distribution having integrated out the GP

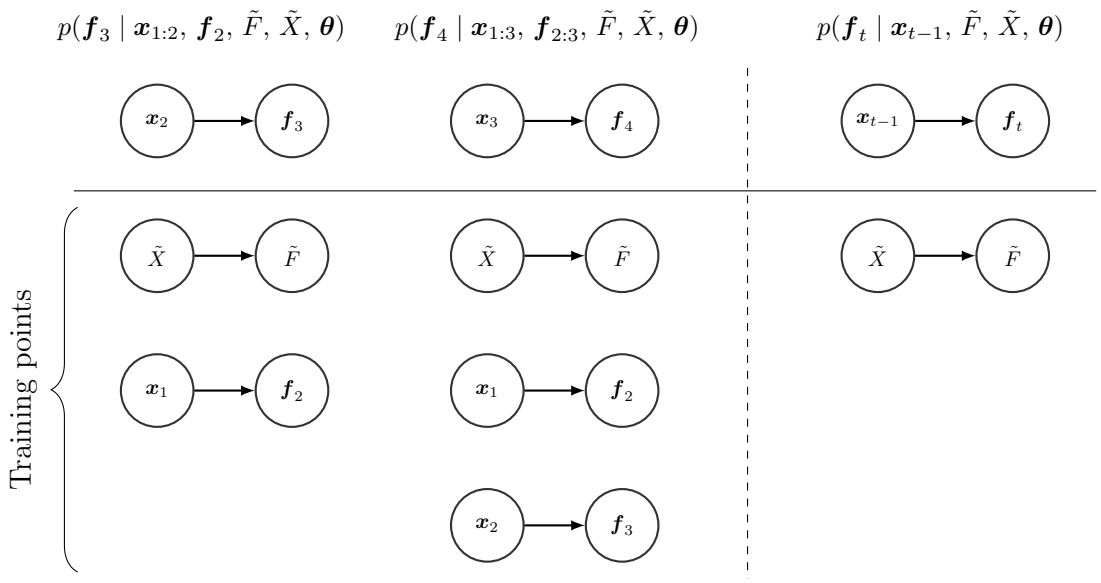


Figure 3.5: Illustration of the training set for the GP transition function prediction as we move forward in time in an online manner. Predictions are conditioned on previous transitions $\boldsymbol{x}_{1:t-2}$ to $\boldsymbol{f}_{2:t-1}$, as well as the pseudo-points $\{\tilde{X}, \tilde{F}\}$. It is intractable to integrate out the $\boldsymbol{x}$ and $\boldsymbol{f}$ variables from the training set, which poses a problem for learning in the GP-SSM. One solution is to only use the pseudo-points to specify the GP transition function, as shown on the right of the figure.

latent function values,

$$p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \tilde{F} \mid \tilde{X}, \boldsymbol{\theta}) \;\approx\; p(\tilde{F} \mid \tilde{X}, \boldsymbol{\theta})\, p(\boldsymbol{x}_1) \prod_{t=2}^{T} p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \tilde{F}, \tilde{X}, \boldsymbol{\theta}) \tag{3.43}$$

There are still two sets of unknown variables in the joint of equation 3.43, the latent states $\boldsymbol{x}_{1:T}$ and the pseudo-targets $\tilde{F}$ (we are treating the pseudo-inputs as parameters here), which we would like to integrate out. We cannot immediately see how to marginalise over the latent states as both $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t-1}$ appear inside each term of the product in equation 3.43, with $\boldsymbol{x}_{t-1}$ appearing in a complex way: inside the GP covariance function.

In FITC (Snelson and Ghahramani, 2006a), the pseudo-targets are integrated out against the prior $p(\tilde{F} \mid \tilde{X}, \boldsymbol{\theta})$ which is set to be the same GP prior as was placed on f (the intuition being that the pseudo-points should behave like the latent points). This is the same situation as we have in equation 3.43. However we cannot do that here as the pseudo-targets are required to specify the transition function as the $\boldsymbol{x}$ variables are unknown (whereas they are known in FITC): if we integrate $\tilde{F}$ out then we are only left with $\tilde{X}$ to specify the transition model. The ideal situation is to find the posterior on the pseudo-targets given the observed data, $p(\tilde{F} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, however this is intractable. The most obvious solution is to treat the pseudo-points as parameters to be optimised. Note that this situation is much more preferable to the case where we treat $\boldsymbol{x}_{1:T}$ as parameters as here we can set the number of pseudo-points to be much less than the number of data points. Not only does this greatly help with overfitting but it also reduces the computational burden as well.

Treating the pseudo-points as parameters effectively creates a 'parametric GP'. However, it is important to note that despite using a pseudo-training set, the GP is not degenerate and this is a different model to a Bayesian RBF. This is most clearly seen by considering the variance far away from the pseudo-inputs: for a Bayesian RBF the variance drops away to zero (assuming we are not integrating

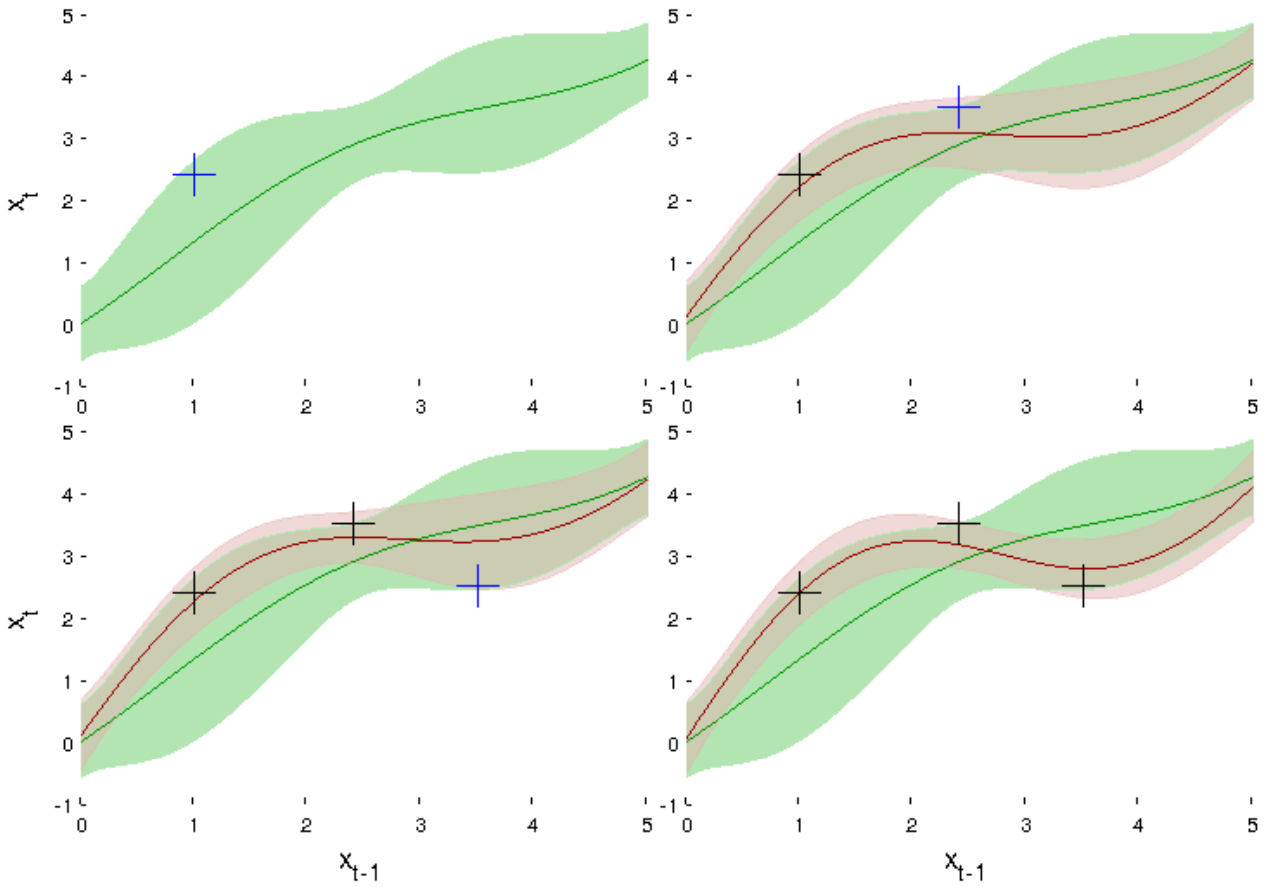


Figure 3.6: The effect past transitions can have on future transitions in a GP state space model with fully connected latent function values (figure 3.3). The green plot shows the posterior over a transition function given by the pseudo-data set. Starting in the top left plot, we sample a transition as shown by the blue cross. The red function in the top right plot shows the new GP posterior if we also condition on this sampled transition, now shown in black. We now sample a second transition from this new posterior, again shown by the blue cross, before also conditioning on it. These steps are repeated in the bottom left plot. The bottom right plot shows the two different posteriors over transitions, the green posterior does not take into account the sampled transitions, whereas the red posterior does.

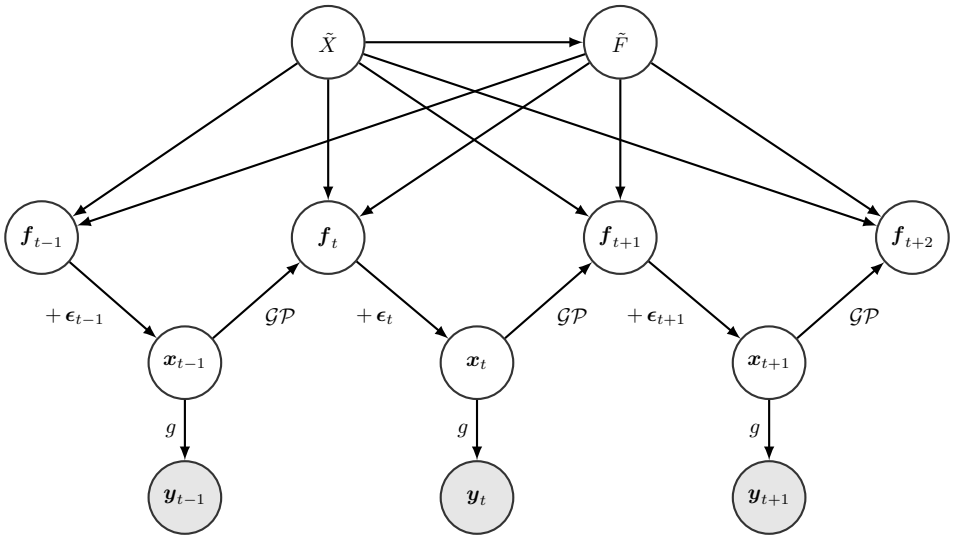


Figure 3.7: Graphical model of a Gaussian Process state space model with independent transitions and with a set of pseudo points $\{\tilde{X}, \tilde{F}\}$.

over the basis function locations), with a GP, parametric or otherwise, the predictive variance returns to the prior variance. There are some significant drawbacks to using a parametric GP approach: the largest of which is the possibility of overfitting. Given that we cannot marginalise out the pseudo training set analytically (for the same reasons as why we introduced it in the first place) we must consider $\tilde{X}$ and $\tilde{F}$ as extra parameters to be optimised. Although we lose many of the desirable properties of a Bayesian, nonparametric model, using a parametric GP allows us to build models which retain analytic tractability. We will show that such models can be very useful for modelling nonlinear dynamical systems.

If $g(\boldsymbol{x}_t) = \boldsymbol{x}_t$, we can pre-select the pseudo-inputs to be a subset of the observed data via the variational metric of Titsias (2009) applied to a GP fitted to the observed data. This is much more efficient than training the pseudo-inputs alongside the other parameters. The GP posterior over f will be most certain around the pseudo-inputs, hence restricting these to be observed data reduces overfitting. The pseudo-targets are added to the parameter set, giving,

$$\boldsymbol{\theta} \;=\; \left[\sigma_f^2,\; \Lambda,\; \tilde{F},\; \Sigma_\epsilon,\; \Sigma_\nu\right] \tag{3.44}$$

Unless we can observe a very long trajectory in one go, a single observed trajectory is unlikely to be sufficient to train the model. We therefore need to be able to make use of several separate trajectories, of potentially different lengths, to fit the parameters $\boldsymbol{\theta}$. All the methods discussed in the rest of this chapter can handle this situation.

3.4.2 Previous Work with Gaussian Process State Space Models

Early GP state space models were focussed around extending the GP-LVM Lawrence (2004) to dynamical systems Lawrence and Moore (2007); Ferris et al. (2007); Wang et al. (2008); Ko and Fox (2011). The graphical model for the GP-LVM is shown in figure 3.8. There is no concept of time or state dynamics in the GP-LVM, hence the only connection between the $\boldsymbol{x}$ variables is via the joint distribution on the observations. In the original GP-LVM the latent states $\boldsymbol{x}$ are optimised along with the hyperparameters of the GP.

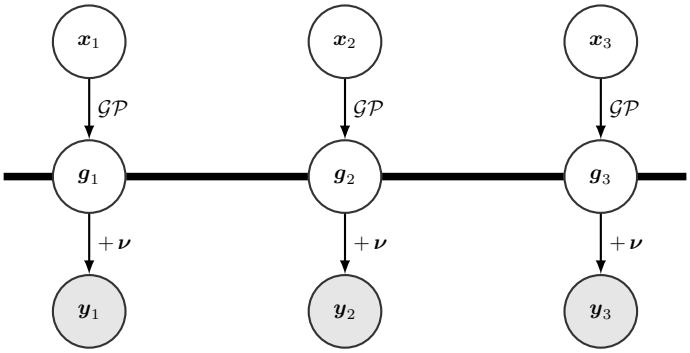


Figure 3.8: The graphical model for the Gaussian Process Latent Variable Model (Lawrence, 2004). A Gaussian Process is placed on the observations $\boldsymbol{y}_i$ given a set of hidden variables $\boldsymbol{x}_i$. The $\boldsymbol{g}_i$ represent the GP latent function values, termed g as the GP plays the role of an observation model.

Ferris et al. (2007) extended the GP-LVM to handle state dynamics for the application of WiFi-SLAM, by introducing a set of constraints on the latent variables, for example that adjacent measurements

should have latent states which are 'close' to one another. In this setup the GP is still modelling the observation function rather than the transition function. One of the first attempts at including a GP transition function in the GP-LVM was the Gaussian Process Dynamical Model (GPDM) (Wang et al., 2008). Figure 3.9 shows the graphical model of the GPDM, which is identical similar to the GP state space model of figure 3.3 except that the GPDM specifies a GP observation model. This is because of the specific application the authors were aiming for (motion capture data), where the observed space is very large and they desired a dimensionality reducing observation model. Both the WiFi-SLAM GP-LVM model and the GPDM set the latent states $\boldsymbol{x}$ to directly be parameters of the model. This is similar to the proposed GP-SSM setup described in section 3.4.1 except that here there are as many pseudo-points as observed points. This model is thus only really appropriate for the case where the size of the observation dimension is significantly greater than the size of the latent dimension, otherwise it will strongly overfit. This model is likely to also be very slow to train for larger data sets due to the large number of parameters to fit. By way of comparison, the framework of the models discussed in the following sections of this chapter specifies a small number of pseudo-points for optimisation and then (approximately) integrates out the latent states conditioned on these. To counter some of these problems Wang et al. (2008) suggest a sampling algorithm based on Monte Carlo EM and Hamiltonian Monte Carlo. In practice this has proved to be very difficult to implement and slow to run.

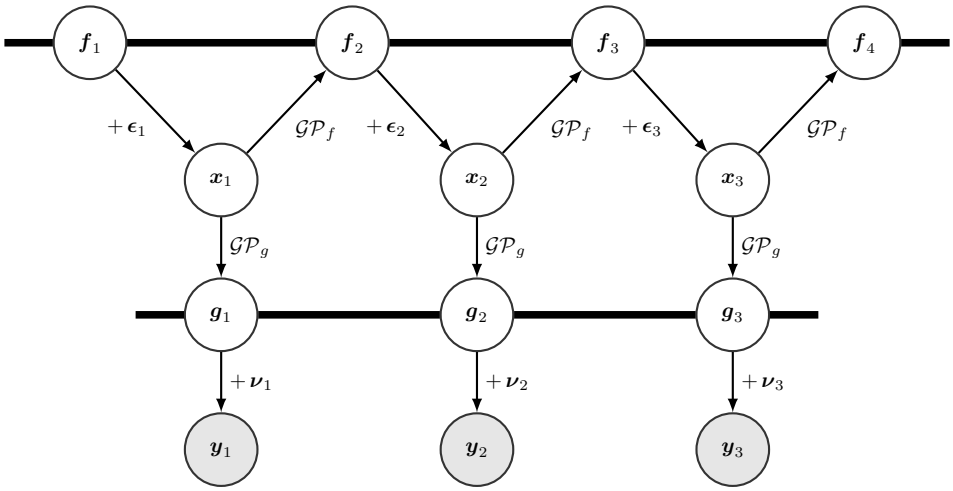


Figure 3.9: The graphical model for the Gaussian Process Dynamical Model (Wang et al., 2008). Building on the GP-LVM, a second Gaussian Process is placed on the transitions between latent states, $\boldsymbol{x}_{t-1}$ to $\boldsymbol{x}_t$.

An extension to the GPDM to overcome some of these problems is seen in the GP-BayesFilterLearn set of algorithms (Ko and Fox, 2011). This method first uses a nearly identical setup to the GPDM (the authors extend the setup to allow partial latent state labels and controls) to find a point estimate of the latent states $\boldsymbol{x}$. These latent states are then used to train one of the authors' BayesFilter algorithms (Ko and Fox, 2009). These filter models differ from the latent variable model trained by the GPDM in that they make the same first order Markov assumption on the transitions as we do (as depicted in figure 3.7). Thus this method can be seen as solving a similar problem to those considered here except that they use a GP-LVM variant to first provide a point estimate of the latent states.

Using point estimates of the latent states is very undesirable as it completely ignores the uncertainty in latent states which will lead to overfitting and overconfident predictions. One attempt to avoid

making point estimates of $\boldsymbol{x}$ is the 'variational GP dynamical system' (Damianou et al., 2011). In this model the authors use two GPs again, one for the observation model as before, and one for the latent state dynamics. However, instead of using the GP to model the transitions $\boldsymbol{x}_{t-1}$ to $\boldsymbol{x}_t$, the authors use the GP to model the latent states as a function of time. This setup is shown in figure 3.10. By using a GP with time as the input rather than the previous state the latent states can be approximately integrated out using a variational approach. This is a big advantage of the approach and significantly reduces overfitting. However, implementing the latent state dynamics using a GP based on time makes this a considerably different model to the ones we have so far considered. On the one hand it can handle time-varying state dynamics, on the other if the transition dynamics are time-invariant then this will be a much less powerful approach than using a GP transition model. This is because it is not uncommon for a system to return to the same or a similar state to one it visited a number of time steps before. If the time gap between visits is large enough then there will be no correlation between the subsequent latent state transitions. This problem is magnified if we consider taking several observed trajectories from a system with time-invariant dynamics. The approach of Damianou et al. (2011) is to use a separate GP to model the temporal dynamics of the latent states of each observed trajectory: that is, no information is shared about the latent state transitions between trajectories.

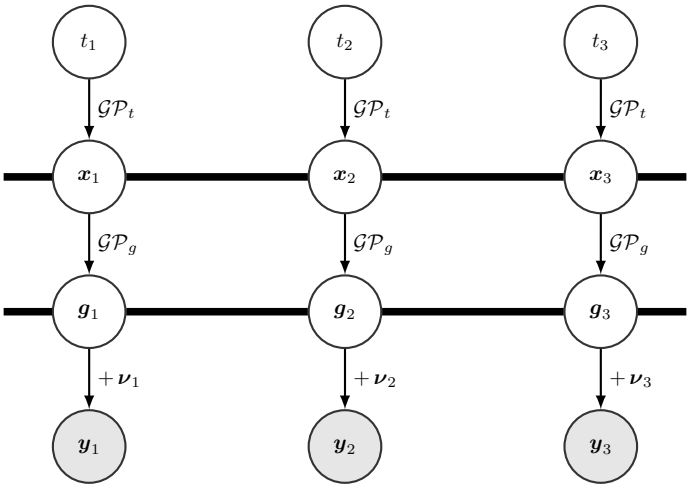


Figure 3.10: The graphical model for the Variational Gaussian Process Dynamical System (Damianou et al., 2011). A Gaussian Process is placed on the observations $\boldsymbol{y}_i$ given a set of hidden variables $\boldsymbol{x}_i$. The $\boldsymbol{g}_i$ represent the GP latent function values, termed g as the GP plays the role of an observation model.

A different approach introduced by Turner et al. (2010) and termed 'GPIL' allowed learning GP transition and observation models based on full posterior distributions of the latent states via the EM algorithm. They use exactly the same model structure as we presented in the previous section (shown in figure 3.7), choosing a small set of pseudo-points as parameters and approximately integrating out the latent states based on these. They use the GP assumed density filter in the E step to compute approximate smoothing posteriors and then optimise an approximation to the log likelihood bound in the M step. Their E step is presented in section 3.6.1 and their M step, which we extend by solving for the pseudo-targets and observation noise rather than optimising, in section 3.6.4. As we shall show in those sections, the E step of Turner et al. (2010) is potentially problematic. Deisenroth and Mohamed (2012) introduced a different method for computing approximate smoothing posteriors in the same model by using Expectation Propagation. We present their method in section 3.6.2

The methods discussed so far are all based on analytic approaches, although Wang et al. (2008) discuss a sampling extension to their model. The analytic methods discussed in sections 3.5 and 3.6 are forced to make a number of approximations in order to make the required computations tractable. The most common approximation is to model the latent state smoothing posteriors with Gaussian distributions as these readily lend themselves to tractable computations. For many applications these approximate distributions capture well the true distribution on the latent states and thus the models built using the approximations perform well. However, it is not hard to construct a system where a Gaussian distribution would be a poor approximation to the latent state distribution. In these cases a sample based method may well outperform the analytic method as they typically require far fewer approximations—in fact usually only the approximation of representing a distribution with samples. There is an extremely large body of work discussing nonlinear dynamical models from a sampling based approach. Most of these are based around Sequential Monte Carlo (SMC) methods, often termed 'Particle Filters'. There are many excellent review articles and books covering these methods, for example Doucet (2001); Kantas et al. (2009); Cappé et al. (2005); Doucet and Johansen (2009). Somewhat surprisingly there has been little attention in the SMC community to applying these methods to Gaussian Process state space models, although this is beginning to change: a key approach is discussed in the next section and further SMC approaches are presented in sections 3.7 and 3.8.

3.4.3 The Fully Bayesian Approach

Section 3.4.1 introduced the maximum likelihood approach to learning a GP-SSM based on observed data. This approach lends itself to approximate-analytic algorithms as well as sampling based ones, and some of these methods are discussed in this chapter. However, these approaches are still using maximum likelihood and are thus vulnerable to overfitting; in particular, optimising the pseudo-targets is fraught with danger. We can attempt to reduce the problem by keeping the number of pseudo-points low, however, this limits the flexibility of the GP transition function; furthermore, as the pseudo-points are local in nature (they only affect the GP posterior around the pseudo-input locations) we need to ensure we have sufficient points to cover the required area of the state space. We can also introduce regularisation although this can be hard to do in a principled manner and introduces further parameters to set.

All of these problems can be avoided by using a fully Bayesian approach to the task, where we average over many parameter settings. Although we cannot do this analytically, it can be done by using a sampling method, as introduced by Frigola et al. (2013). Their approach uses a Particle MCMC (PMCMC) algorithm (see Andrieu et al. (2010) and section 3.8) to draw complete sampled trajectories from the joint latent state smoothing posteriors as well as drawing GP hyperparameters from their smoothing posteriors,

$$\boldsymbol{x}^i_{1:T}, \boldsymbol{\theta}^i \;\sim\; p(\boldsymbol{x}_{1:T}, \boldsymbol{\theta}^i \mid \boldsymbol{y}_{1:T}) \tag{3.45}$$

where i indexes the samples. In fact, these samples are drawn from the fully connected transition model, shown in figure 3.3, rather than the independent transition model of figure 3.7. Each set of samples $\{\boldsymbol{x}^i_{1:T}, \boldsymbol{\theta}^i\}$ defines a Gaussian Process and thus if we draw N samples we have a mixture of N

GPs. This mixture model can be used to make predictions at new inputs,

$$p(\boldsymbol{f}^* \mid \boldsymbol{x}^*, \boldsymbol{y}_{1:T}) \;\approx\; \sum_{i=1}^{N} p(\boldsymbol{f}^* \mid \boldsymbol{x}^*, \boldsymbol{x}^i_{1:T}, \boldsymbol{\theta}^i) \tag{3.46}$$

where,

$$p(\boldsymbol{f}^* \mid \boldsymbol{x}^*, \boldsymbol{x}^i_{1:T}, \boldsymbol{\theta}^i) \;=\; \mathcal{N}\left(\boldsymbol{f}^*;\; \mathfrak{m}(\boldsymbol{x}^*, \boldsymbol{x}^i_{1:T}, \boldsymbol{\theta}^i),\; \mathfrak{s}(\boldsymbol{x}^*, \boldsymbol{x}^i_{1:T}, \boldsymbol{\theta}^i)\right) \tag{3.47}$$

By coupling the sampling algorithm with an inducing point method the complexity of drawing a single sampled trajectory and corresponding hyperparameters is $\mathcal{O}(M^2\,T)$, where M is the number of inducing points being used and T is the length of the observed trajectory. Thus if we draw N_s samples then the complexity is $\mathcal{O}(N_s\,M^2\,T)$. As noted previously, we may need to use several observed trajectories in order to gain sufficient data to accurately model the system. This can be done by applying the method described above to each observed trajectory individually. If we have N_{traj} observed trajectories then the total computational complexity will be $\mathcal{O}(N_{\text{traj}}\,N_s\,M^2\,T)$. Making predictions with this model is also considerably more expensive than in the parametric GP model discussed earlier: predictions cost $\mathcal{O}(N_{\text{traj}}\,N_s\,M^2)$ in comparison to $\mathcal{O}(M^2)$ for the parametric GP approach. For a complex system the fully Bayesian approach could get very computationally heavy, although samples from different observed trajectories can be drawn in parallel.

3.4.4 The Variational Approach

The parametric GP approach outlined in section 3.4.1 has the undesirable requirement of requiring training of the pseudo targets, which can lead to overfitting. Although we cannot treat the pseudo targets probabilistically with direct approximations to the marginal likelihood, we can if we use a variational method to lower bound the marginal likelihood. Variational methods are recapped in section 1.5. Current variational methods in Gaussian Processes are largely built upon Titsias (2009) and Titsias and Lawrence (2010). The second of these papers demonstrated how the combination of a variational approach with the introduction of pseudo-points allowed the latent inputs in the GPLVM to be integrated out. Frigola et al. (2014) applied the same principles to derive a variational lower bound on the marginal likelihood for a GP state space model. We will look at their approach here.

The graphical model with the pseudo points and fully connected GP latent variables was shown in figure 3.4. In order to integrate out the pseudo targets we include them in the same GP prior as is used for the latent function values f (as is done in FITC (Snelson and Ghahramani, 2006a)). We will continue to treat the pseudo inputs as parameters, although the variational approach gives us a slightly different viewpoint on them. In order to keep the equations as clear as possible we will drop the explicit conditioning on $\tilde{X}$ and $\boldsymbol{\theta}$. In section 3.4.1 we found the complete joint probability distribution to be,

$$p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \tilde{F}) \;=\; p(\boldsymbol{y}_{1:T} \mid \boldsymbol{x}_{1:T})\, p(\boldsymbol{x}_1)\, p(\tilde{F})\, p(\boldsymbol{x}_{1:T} \mid \boldsymbol{f}_{1:T}) \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1}, \boldsymbol{f}_{2:t-1}, \tilde{F}) \tag{3.48}$$

To find the marginal likelihood therefore we need to integrate the joint density over all the latent

variables,

$$p(\boldsymbol{y}_{1:T}) = \int p(\boldsymbol{y}_{1:T} \mid \boldsymbol{x}_{1:T})\, p(\boldsymbol{x}_1)\, p(\tilde{F})\, p(\boldsymbol{x}_{1:T} \mid \boldsymbol{f}_{1:T}) \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1},\, \boldsymbol{f}_{2:t-1},\, \tilde{F})\; d\boldsymbol{f}_{1:T}\; d\boldsymbol{x}_{1:T}\; d\tilde{F} \tag{3.49}$$

As we have stated previously, we can solve the integral over $\boldsymbol{f}_{1:T}$, however this leads to a non-Gaussian joint distribution on $\boldsymbol{x}_{1:T}$, and we can proceed no further with the integration. The problematic term in equation 3.49 is $p(\boldsymbol{f}_t \mid \boldsymbol{f}_{2:t-1}, \boldsymbol{x}_{1:t-1}, \tilde{F})$ and so Frigola et al. (2014) take the same strategy as Titsias and Lawrence (2010) and choose a variational distribution on the latent variables which removes this difficult term. Let,

$$q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \tilde{F}) = q(\boldsymbol{x}_{1:T})\, q(\tilde{F}) \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1},\, \boldsymbol{f}_{2:t-1},\, \tilde{F}) \tag{3.50}$$

which factorises so that $\boldsymbol{x}_t$ has no dependence on $\boldsymbol{f}_t$. The approximating family is the same model as that shown in figure 3.4 except that the link between $\boldsymbol{f}_t$ and $\boldsymbol{x}_t$ is removed. This leads to a simpler distribution q, which will serve as a surrogate for the true distribution. We will then attempt to find the q within the approximating family which provides the closest match (in the KL sense) to the true posterior $p(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \tilde{F} \mid \boldsymbol{y}_{1:T})$. Recall from equation 1.40 that the variational lower bound is given by,

$$\begin{aligned}
&\log p(\boldsymbol{y}_{1:T}) \\
&\geq \int q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{1:T}, \tilde{F}) \log \frac{p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \tilde{F})}{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{1:T}, \tilde{F})}\; d\boldsymbol{f}_{1:T}\; d\boldsymbol{x}_{1:T}\; d\tilde{F} \\
&= \int q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{1:T}, \tilde{F}) \log \frac{p(\boldsymbol{y}_{1:T} \mid \boldsymbol{x}_{1:T})\, p(\boldsymbol{x}_1)\, p(\tilde{F})\, p(\boldsymbol{x}_{1:T} \mid \boldsymbol{f}_{1:T}) \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1},\, \boldsymbol{f}_{2:t-1},\, \tilde{F})}{q(\boldsymbol{x}_{1:T})\, q(\tilde{F}) \prod_{t=2}^{T} p(\boldsymbol{f}_t \mid \boldsymbol{x}_{1:t-1},\, \boldsymbol{f}_{2:t-1},\, \tilde{F})}\; d\boldsymbol{f}_{1:T}\; d\boldsymbol{x}_{1:T}\; d\tilde{F} \\
&= \int q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{1:T}, \tilde{F}) \log \frac{p(\boldsymbol{y}_{1:T} \mid \boldsymbol{x}_{1:T})\, p(\boldsymbol{x}_1)\, p(\tilde{F})\, p(\boldsymbol{x}_{1:T} \mid \boldsymbol{f}_{1:T})}{q(\boldsymbol{x}_{1:T})\, q(\tilde{F})}\; d\boldsymbol{f}_{1:T}\; d\boldsymbol{x}_{1:T}\; d\tilde{F} \\
&= \int q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{1:T}, \tilde{F}) \left[\log \frac{p(\tilde{F})}{q(\tilde{F})} - \log q(\boldsymbol{x}_{1:T}) + \log p(\boldsymbol{y}_{1:T} \mid \boldsymbol{x}_{1:T})\, p(\boldsymbol{x}_1)\, p(\boldsymbol{x}_{1:T} \mid \boldsymbol{f}_{1:T})\right] d\boldsymbol{f}_{1:T}\; d\boldsymbol{x}_{1:T}\; d\tilde{F} \\
&= -\mathrm{KL}\left(q(\tilde{F}) \,||\, p(\tilde{F})\right) + \mathrm{H}(q(\boldsymbol{x})) + \int q(\boldsymbol{x}_t) \log p(\boldsymbol{x}_1)\; d\boldsymbol{x}_1 + \sum_{t=1}^{T} \int q(\boldsymbol{x}_t) \log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)\; d\boldsymbol{x}_t \\
&\qquad + \sum_{t=1}^{T} \int q(\boldsymbol{x}_t)\, q(\tilde{F})\, p(\boldsymbol{f}_t \mid \boldsymbol{x}_{t-1}, \tilde{F}) \log p(\boldsymbol{x}_t \mid \boldsymbol{f}_t)\; d\boldsymbol{f}_t\; d\boldsymbol{x}_t\; d\tilde{F}
\end{aligned} \tag{3.51}$$

$$\triangleq \mathcal{L}\big(q(\boldsymbol{x}_{1:T}),\, q(\tilde{F})\big) \tag{3.52}$$

Note how by deleting the term which chains the GP latent variables together we once again only have the probability distribution on the transition at time t depending on $\boldsymbol{x}_{t-1}$ and not on all previous latent states. This is exactly the same position as we were in when we introduced the conditional independence of the GP latent variables given the pseudo-points. Indeed, Frigola et al. (2014) have shown that the same marginal likelihood lower bound is found if you start from the independent

transition model. We can solve the final integral over $\boldsymbol{f}_t$ in equation 3.51 to find,

$$q(\boldsymbol{x}_t)\, q(\tilde{F}) \int p(\boldsymbol{f}_t \mid \boldsymbol{x}_{t-1},\, \tilde{F}) \log p(\boldsymbol{x}_t \mid \boldsymbol{f}_t)\; d\boldsymbol{f}_t \;=\; \log \mathcal{N}\left(\boldsymbol{x}_t;\; A(\boldsymbol{x}_{t-1}),\; \Sigma_\epsilon\right) \;-\; \frac{1}{2}\mathrm{tr}\left\{\Sigma_\epsilon^{-1}\, B(\boldsymbol{x}_{t-1})\right\} \quad (3.53)$$

with $A(\boldsymbol{x}_{t-1})$ being a $M \times D$ matrix (where M is the number of pseudo-points we are using) with columns $\{\boldsymbol{a}_i\}_{i=1}^D$ and $B(\boldsymbol{x}_{t-1})$ being a $D \times D$ diagonal matrix with diagonal entries $\{b_i\}_{i=1}^D$:

$$\boldsymbol{a}_i(\boldsymbol{x}_{t-1}) \;=\; k_i(\boldsymbol{x}_{t-1}, \tilde{X})\, K_i(\tilde{X}, \tilde{X})^{-1}\, \tilde{F}_i \quad (3.54)$$

$$b_i(\boldsymbol{x}_{t-1}) \;=\; k_i(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-1}) \;-\; k_i(\boldsymbol{x}_{t-1}, \tilde{X})\, K_i(\tilde{X}, \tilde{X})^{-1}\, k_i(\tilde{X}, \boldsymbol{x}_{t-1}) \quad (3.55)$$

where $\tilde{F}_i$ are the pseudo targets for the ith output dimension and the subscript on the covariance function indicates that we are using the hyperparameters for the ith output dimension. The forthcoming derivations are difficult to represent for multidimensional states, although the mathematics works out the same, and so for clarity of exposition we will restrict ourself to a one dimensional state. We also introduce the shorthand,

$$K(\tilde{X},\, \tilde{X}) \;=\; K_{\tilde{X},\tilde{X}} \quad (3.56)$$

The advantage of the variational approach is that we do not need to optimise the pseudo-targets as we can find an approximate posterior over them. Frigola et al. (2014) show that using variational calculus one can find that the optimal variational distribution for the inducing points is a multivariate Gaussian,

$$\begin{aligned} q^*\left(\tilde{F}\right) \;&=\; \underset{q(\tilde{F})}{\arg\max}\; \mathcal{L}\big(q(x_{1:T}),\, q(\tilde{F})\big) \\ &=\; \mathcal{N}\left(\tilde{F};\; -\frac{1}{2}\boldsymbol{\eta}_2^{-1}\,\boldsymbol{\eta}_1,\; -\frac{1}{2}\boldsymbol{\eta}_2^{-1}\right) \;=\; \mathcal{N}\left(\tilde{F};\; \boldsymbol{\mu}_{\tilde{F}},\; \Sigma_{\tilde{F}}\right) \end{aligned} \quad (3.57)$$

$$\boldsymbol{\eta}_1 \;=\; \Sigma_\epsilon^{-1} \sum_{t=1}^{T} \mathbb{E}_{q(x_{t-1},\, x_t)}\left[x_t\; A(x_{t-1})\right] \qquad (M \times 1) \quad (3.58)$$

$$\boldsymbol{\eta}_2 \;=\; -\frac{1}{2}K_{\tilde{X},\,\tilde{X}}^{-1} \;-\; \frac{1}{2}\Sigma_\epsilon^{-1} \sum_{t=1}^{T} \mathbb{E}_{q(x_{t-1})}\left[A(x_{t-1})\, A(x_{t-1})^T\right] \qquad (M \times M) \quad (3.59)$$

Optimal here means that this is the distribution which minimizes the KL divergence between $q(x_{1:T},\, f_{2:T},\, \tilde{F})$ and the true posterior $p(x_{1:T},\, f_{2:T},\, \tilde{F} \mid y_{1:T})$. We can also find the optimal distribution for $q(x_{1:T})$, however there is no closed form for this, instead it takes the form of another dynamical system, albeit a more simple one,

$$\begin{aligned} q^*(x_{1:T}) \;&=\; \underset{q(x_{1:T})}{\arg\max}\; \mathcal{L}\big(q(x_{1:T}),\, q(\tilde{F})\big) \\ &\propto\; p(x_1) \prod_{t=2}^{T} p(y_t \mid x_t) \exp\left(-\frac{1}{2}\Sigma_\epsilon^{-1} B(x_{t-1})\right) \mathcal{N}\left(x_t;\; A(x_{t-1})^T \boldsymbol{\mu}_{\tilde{F}}\right) \end{aligned} \quad (3.60)$$

One could use sampling methods or other approximations to find a close fit to $q^*(x_{1:T})$ although this adds extra complexity and makes training the GP hyperparameters more difficult (especially with a sampling approach). As $q(x_{1:T})$ is a variational distribution we are free to choose it as we wish and optimise any parameters it has without fear of overfitting. We therefore take a simple approach and

choose $q(x_{1:T})$ to have a Markov decomposition,

$$q(x_{1:T}) \;=\; q(x_1) \prod_{t=2}^{T} q(x_t \mid x_{t-1}) \tag{3.61}$$

and choose each conditional distribution be Gaussian. It turns out that with this choice of variational distribution we only need to define the pairwise joint distributions, as these contain the only variables which affect the variational bound — note that in the lower bound (equation 3.51) $q(x)$ only appears in either a marginal form, $q(x_t)$ or inside the entropy term $H(q(x_{1:T}))$ for which there is a special formulation (see equation 3.71). Thus we define,

$$q(x_{t-1},\, x_t) \;=\; \mathcal{N}\left(\begin{bmatrix} x_{t-1} \\ x_t \end{bmatrix};\; \begin{bmatrix} \mu_{t-1} \\ \mu_t \end{bmatrix},\; \begin{bmatrix} \Sigma_{t-1} & C_{t-1,t} \\ C_{t,t-1} & \Sigma_t \end{bmatrix} \right) \tag{3.62}$$

for $t = 2$ to T. $C_{t-1,t}$ is the pairwise covariance between x_{t-1} and x_t, which makes it a $D \times D$ matrix (although here we are just considering $D = 1$) and it should not be confused with the observation function, $y_t = C\, x_t + \nu_t$ from equation 3.4. This Markov formulation leads to a tri-diagonal precision matrix for $q(x_{1:T})$. The means, variances, and pairwise covariances are treated as variational parameters and are added to the parameter set to be optimised. We will need to make a number of expectations over the GP covariance function when its inputs are Gaussian. As discussed before, this limits the choice of covariance function we can use. As with the other methods discussed in this chapter, here we will use the Gaussian covariance function (a.k.a. squared exponential/exponentiated quadratic), shown in equation 3.8. We have derived a number of results with this covariance function: expectations, variances, and derivatives, and present them in Appendix A. Rather than repeating all those results here, the reader is directed to the appendix to find the various results needed. With a Gaussian $q(x)$ we can solve the required expectations over x for both $q^*(\tilde{F})$ and the lower bound:

The expectation of $A(x_{t-1})$, a $M \times 1$ vector,

$$\mathbb{E}_{q(x_{t-1})}\left[A(x_{t-1})\right] \;=\; \mathbb{E}_{q(x_{t-1})}\left[\boldsymbol{k}(x_{t-1},\, \tilde{X})\right] K^{-1}_{\tilde{X},\tilde{X}} \tag{3.63}$$

The variance of $A(x_{t-1})$, a $M \times M$ matrix,

$$\mathbb{V}_{q(x_{t-1})}\left[A(\boldsymbol{x}_{t-1})\right] \;=\; K^{-1}_{\tilde{X},\tilde{X}}\, \mathbb{V}_{q(x_{t-1})}\left[\boldsymbol{k}(x_{t-1},\, \tilde{X})\right] K^{-1}_{\tilde{X},\tilde{X}} \tag{3.64}$$

The covariance of x_t with $A(x_{t-1})$, a $1 \times M$ vector,

$$\mathbb{C}_{q(x_{t-1},x_t)}[x_t,\, A(x_{t-1})] \;=\; \mathbb{C}_{q(x_{t-1},x_t)}[x_t,\, \boldsymbol{k}(x_{t-1}, \tilde{X})]\, K^{-1}_{\tilde{X},\tilde{X}} \tag{3.65}$$

The expectation of $B(x_{t-1})$, a scalar,

$$\begin{aligned} \mathbb{E}_{q(x_{t-1})}\left[B(x_{t-1})\right] \;&=\; \mathbb{E}_{q(x_{t-1})}\left[k(x_{t-1},\, x_{t-1}) \;-\; \boldsymbol{k}(x_{t-1},\, \tilde{X})\, K^{-1}_{\tilde{X},\tilde{X}}\, \boldsymbol{k}(\tilde{X},\, x_{t-1})\right] \\ &=\; \sigma_f^2 \;-\; \mathbb{E}\left[\boldsymbol{k}(x_{t-1},\, \tilde{X})\right] K^{-1}_{\tilde{X},\tilde{X}}\, \mathbb{E}\left[\boldsymbol{k}(\tilde{X},\, x_{t-1})\right] \;-\; \mathrm{tr}\left\{ K^{-1}_{\tilde{X},\tilde{X}}\, \mathbb{V}\left[\boldsymbol{k}(\boldsymbol{x}_{t-1},\, \tilde{X})\right] \right\} \end{aligned} \tag{3.66}$$

The expectation of x_t times $A(x_{t-1})$, a $1 \times M$ vector,

$$\begin{aligned}\mathbb{E}_{q(x_{t-1},x_t)}\left[x_t\, A(x_{t-1})\right] &= \mathbb{E}_{q(x_{t-1},x_t)}\left[x_t\, \boldsymbol{k}(x_{t-1},\, \tilde{X})\right] K^{-1}_{\tilde{X},\tilde{X}} \\ &= \left(\mu_t\, \mathbb{E}_{q(x_{t-1})}\left[\boldsymbol{k}(x_{t-1},\, \tilde{X})\right] \;+\; \mathbb{C}_{q(x_{t-1},x_t)}\left[x_t,\; \boldsymbol{k}(x_{t-1},\, \tilde{X})\right]\right) K^{-1}_{\tilde{X},\tilde{X}} \end{aligned} \tag{3.67}$$

The expectation of the outer product of $A(x_{t-1})$ with itself, a $M \times M$ matrix,

$$\mathbb{E}_{q(x_{t-1})}\left[A(x_{t-1})\, A(x_{t-1})^T\right] \;=\; K^{-1}_{\tilde{X},\tilde{X}}\, \mathbb{E}_{q(x_{t-1},x_t)}\left[\boldsymbol{k}(\tilde{X},\, x_{t-1})\, \boldsymbol{k}(x_{t-1},\, \tilde{X})\right] K^{-1}_{\tilde{X},\tilde{X}} \tag{3.68}$$

Using these results we can write out the various terms in the lower bound from equation 3.51: firstly, the Kullback-Leibler divergence term,

$$\begin{aligned}-\mathrm{KL}\left(q(\tilde{F}) \;||\; p(\tilde{F})\right) &= -\mathrm{KL}\left(\mathcal{N}(\boldsymbol{\mu}_{\tilde{F}},\; \Sigma_{\tilde{F}}) \;||\; \mathcal{N}(\mathbf{0},\; K_{\tilde{X},\tilde{X}})\right) \\ &= -\frac{1}{2}\left(\mathrm{tr}\left\{K^{-1}_{\tilde{X},\tilde{X}}\, \Sigma_{\tilde{F}}\right\} \;+\; \boldsymbol{\mu}^T_{\tilde{F}}\, K^{-1}_{\tilde{X},\tilde{X}}\, \boldsymbol{\mu}_{\tilde{F}} \;-\; D \;-\; \log|\Sigma_{\tilde{F}}| \;+\; \log|K_{\tilde{X},\tilde{X}}|\right) \end{aligned} \tag{3.69}$$

Next, the likelihood term,

$$\begin{aligned}\sum_{t=1}^{T} \int q(x_t) \log p(y_t \mid x_t)\; dx_t &= \sum_{t=1}^{T} \int q(x_t) \left(-\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_\nu| - \frac{1}{2}(y_t - x_t)^T \Sigma_\nu^{-1}(y_t - x_t)\right)\; dx_t \\ &= -\frac{TD}{2}\log 2\pi - \frac{T}{2}\log|\Sigma_\nu| - \frac{1}{2}\sum_{t=1}^{T}\left[(y_t - \mu_t)^T \Sigma_\nu^{-1}(y_t - \mu_t) + \mathrm{tr}\left\{\Sigma_\nu^{-1}\Sigma_t\right\}\right] \end{aligned} \tag{3.70}$$

The entropy term, where we make use of the fact that the precision matrix $\Sigma_{1:T}$ of our variational distribution $q(x_{1:T})$ is tri-diagonal, and thus has a particular decomposition for its determinant:

$$\begin{aligned}H\left(q(x_{1:T})\right) &= \frac{1}{2}\log|2\pi\, e\, \Sigma_{1:T}| \\ &= \frac{1}{2}DT\log(2\pi e) \;+\; \frac{1}{2}\sum_{t=2}^{T}\log|\Sigma_{t-1:t}| \;-\; \frac{1}{2}\sum_{t=2}^{T-1}\log|\Sigma_t| \end{aligned} \tag{3.71}$$

where,

$$\Sigma_{t-1:t} \;=\; \begin{bmatrix} \Sigma_{t-1} & C_{t-1,t} \\ C_{t-1,t} & \Sigma_t \end{bmatrix} \tag{3.72}$$

the $2D \times 2D$ pairwise covariance matrix.

The next term measures how well the transition function predicts the transitions from x_{t-1} to x_t based

on the current posterior on the pseudo-points, this is the first term in equation 3.53:

$$\begin{aligned}
&\sum_{t=2}^{T} \int q(x_{t-1:t})\, q(\tilde{F})\, \log \mathcal{N}(x_t;\ \tilde{F}^T A(x_{t-1}),\ \Sigma_\epsilon)\ dx_t\ dx_{t-1}\ d\tilde{F} \\
&\quad = -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_\epsilon| - \frac{1}{2}\int q(x_{t-1:t})\, q(\tilde{F})\, \left((x_t - A(x_{t-1})^T\tilde{F})^T\, \Sigma_\epsilon^{-1}(x_t - A(x_{t-1})^T\tilde{F})\right)\ d\tilde{F}\ dx_{t-1:t} \\
&\quad = -\frac{D}{2}\log 2\pi\ -\ \frac{1}{2}\log|\Sigma_\epsilon|\ -\ \frac{1}{2}\,\Sigma_\epsilon\, \mathbb{E}_{x_{t-1}}\left[A(x_{t-1})^T \Sigma_{\tilde{F}}\, A(x_{t-1})\right] \\
&\qquad\qquad -\ \frac{1}{2}\,\mathbb{E}_{x_{t-1},x_t}\left[(x_t - A(x_{t-1})^T\,\boldsymbol{\mu}_{\tilde{F}})^T\, \Sigma_\epsilon^{-1}(x_t - A(x_{t-1})^T\,\boldsymbol{\mu}_{\tilde{F}})\right] \\
&\quad = -\frac{D}{2}\log 2\pi\ -\ \frac{1}{2}\log|\Sigma_\epsilon|\ -\ \frac{1}{2}\,\Sigma_\epsilon\, \mathbb{E}_{x_{t-1}}\left[A(x_{t-1})\right]^T \Sigma_{\tilde{F}}\, \mathbb{E}_{x_{t-1}}\left[A(x_{t-1})\right]\ -\ \frac{1}{2}\mathrm{tr}\left\{\Sigma_\epsilon\, \Sigma_{\tilde{F}}\, \mathbb{V}[A(x_{t-1})]\right\} \\
&\qquad\qquad -\ \frac{1}{2}\left(\mu_t - \mathbb{E}_{x_{t-1}}\left[A(x_{t-1})\right]\,\boldsymbol{\mu}_{\tilde{F}}\right)^T \Sigma_{\tilde{F}}^{-1}\left(\mu_t - \mathbb{E}_{x_{t-1}}\left[A(x_{t-1})\right]\,\boldsymbol{\mu}_{\tilde{F}}\right)\ -\ \frac{1}{2}\mathrm{tr}\left\{\Sigma_\epsilon^{-1}\Sigma_t\right\} \\
&\qquad\qquad -\ \frac{1}{2}\mathrm{tr}\left\{\Sigma_\epsilon^{-1}\boldsymbol{\mu}_{\tilde{F}}^T\, \mathbb{V}[A(x_{t-1})]\,\boldsymbol{\mu}_{\tilde{F}}\right\}\ +\ \mathrm{tr}\left\{\Sigma_\epsilon^{-1}\mathbb{C}[x_t,\, \boldsymbol{\mu}_{\tilde{F}}^T\, A(x_{t-1})]\right\}
\end{aligned} \tag{3.73}$$

The second term in equation 3.53 penalises uncertainty in the GP posterior,

$$-\frac{1}{2}\sum_{t=1}^{T}\int q(x_t)\, q(\tilde{F})\, \mathrm{tr}\left\{\Sigma_\epsilon^{-1}B(x_{t-1})\right\}\ dx_t\ dx_{t-1}\ d\tilde{F}\ =\ -\frac{1}{2}\sum_{t=1}^{T}\mathrm{tr}\left\{\Sigma_\epsilon^{-1}\mathbb{E}_{x_{t-1}}\left[B(x_{t-1})\right]\right\} \tag{3.74}$$

All these terms are computable in closed form. Furthermore, we can take the derivative of the lower bound w.r.t. the GP hyperparameters and the moments of the variational distribution on the latent states, thus we can optimise the lower bound via gradient ascent. To test the variational approach we trained it on a 1D 'kink' transition function, as shown in figure 3.11. Ten trajectories were generated from the transition function for training, with a further twenty used for the test set. Figure 3.11 also shows the resulting GP posterior; whilst the posterior mean matches the true transition function well up to around $x_{t-1} = 4.5$, the variational method has greatly overestimated the amount of process noise. We found this to be a common occurrence — the variational method required, on average, a lot more data to produce a similar fit to other methods with fewer data points. Figure 3.12 shows how the bound on the NLML decreases over optimisation along with how the test likelihood changes. The test performance is measured by computing the (negative log) probability (NLP) of the true observation under the predicted distribution. For each test trajectory we have access to the sequence of latent states, therefore, rather than computing the probability of a single observed point we integrate over the true distribution on the observed point given the latent point. We can compute this test statistic easily for a D-dimensional state and so we write it in terms of this generality,

$$\begin{aligned}
\text{Test NLP}\ &=\ -\log\int p(\boldsymbol{y}_t^*\mid \boldsymbol{\mu}_t^*,\ \Sigma_t^*)\, p(\boldsymbol{y}_t^*\mid \boldsymbol{x}_t^*,\, \boldsymbol{\theta})\ d\boldsymbol{y}_t^* \\
&=\ -\log\int \mathcal{N}(\boldsymbol{y}_t^*;\ \boldsymbol{\mu}_t^*,\ \Sigma_t^*)\,\mathcal{N}(\boldsymbol{y}_t^*;\ \boldsymbol{x}_t^*,\ \Sigma_\nu)\ d\boldsymbol{y}_t^* \\
&=\ \frac{D}{2}\log 2\pi\ +\ \frac{1}{2}\log|\Sigma_t^*\ +\ \Sigma_\nu|\ +\ \frac{1}{2}\left(\boldsymbol{\mu}_t^*\ -\ \boldsymbol{x}_t^*\right)^T\left(\Sigma_t^*\ +\ \Sigma_\nu\right)^{-1}\left(\boldsymbol{\mu}_t^*\ -\ \boldsymbol{x}_t^*\right)
\end{aligned} \tag{3.75}$$

where $\boldsymbol{\mu}_t^*$ and Σ_t^* are the predictive mean and variance calculated by the trained GP model evaluated at the latent point $\boldsymbol{x}_{t-1}^*$. Both training and test metrics show a very rapid convergence, which is somewhat surprising given that there are 627 parameters to fit.

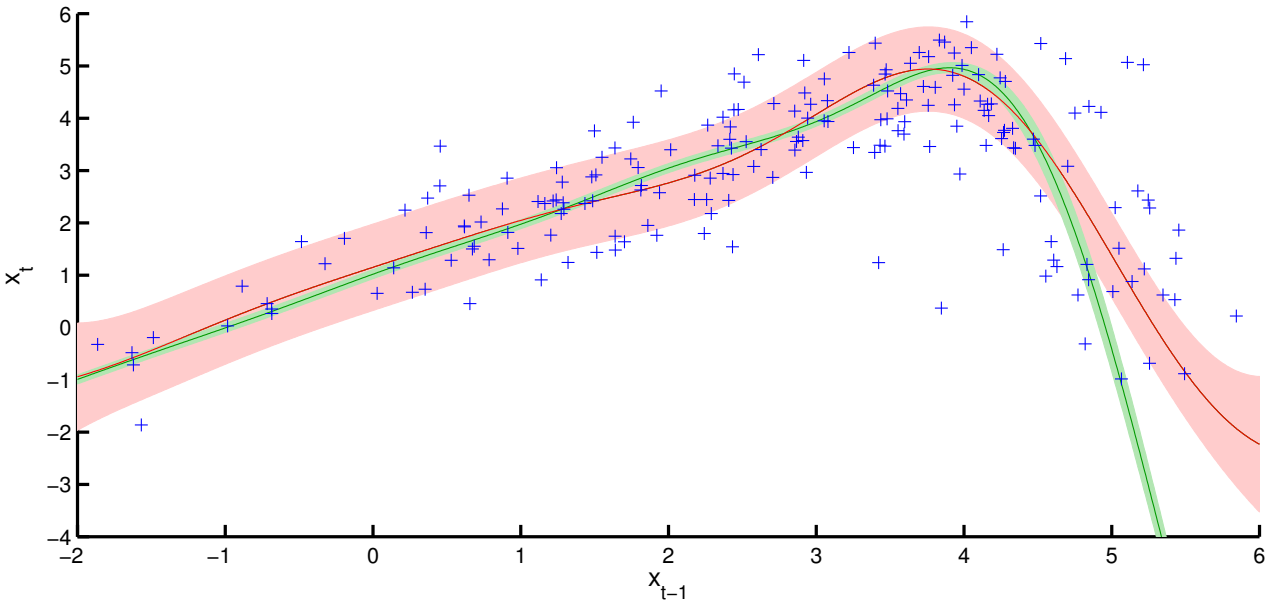


Figure 3.11: The true 1D test transition function is shown in green, where the shaded region shows two standard deviations of process noise. The red distribution is the transition function learnt by the variational approach along with the two standard deviations of process noise. The blue crosses represent the *observed* transitions, y_{t-1} to y_t, and so should not be within the transition function posterior.

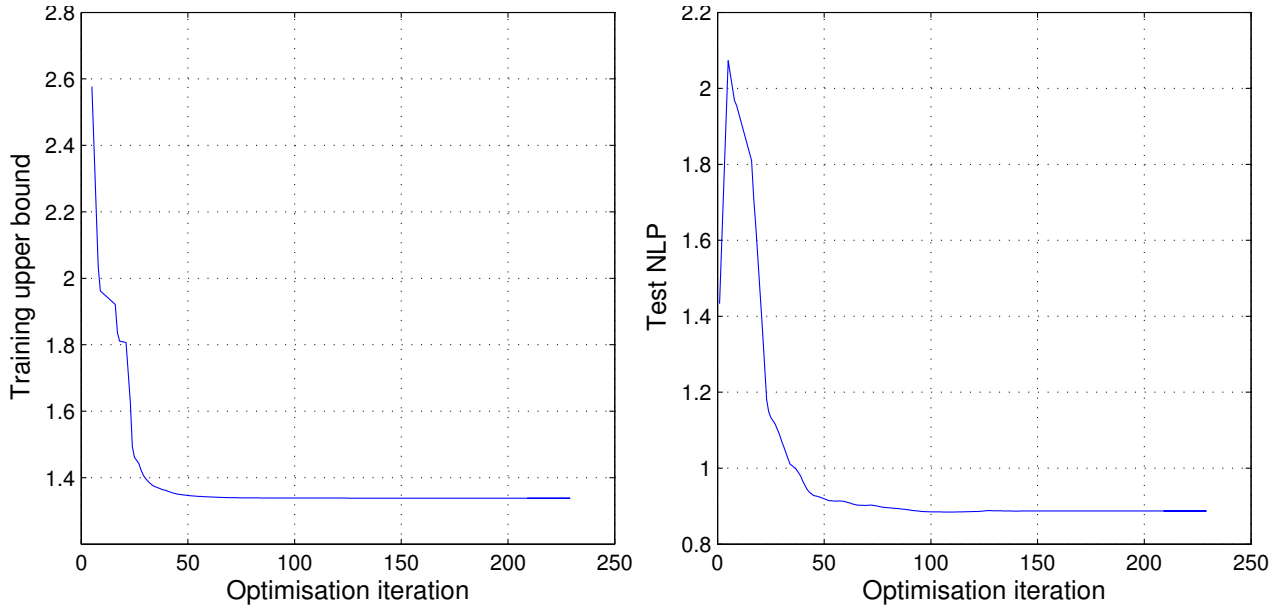


Figure 3.12: The training and test performance for the 1D function as the GP and variational parameters are optimised. The training plot is of the upper bound value and the test plot uses the metric in equation 3.75.

3.5 The Direct Method

We now look at a number of different approaches based on the optimisation of the pseudo-targets $\tilde{F}$, which are included in the parameter vector $\boldsymbol{\theta}$. In this section we present an approximate-analytic approach to learning in GP-SSMs, which only makes use of filtering and attempts to directly optimise the marginal likelihood. The variational approach discussed in the previous section maintained an

explicit distribution over the latent states $\boldsymbol{x}_{1:T}$, which was an approximation to the smoothing posterior $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T})$. In the following approach we do not need to construct such a distribution, in contrast to the variational approach and to the expectation maximisation algorithms encountered in sections 3.6 and 3.8. Our approach will proceed by making a single forward filtering sweep, continually projecting intractable distributions onto the Gaussian family using moment matching. This is often termed assumed density filtering (ADF), although here we extend this to computing the marginal likelihood.

The marginal likelihood can be factored as follows,

$$p(\boldsymbol{y}_{1:T} \mid \boldsymbol{\theta}) \;=\; p(\boldsymbol{y}_1 \mid \boldsymbol{\theta}) \prod_{t=2}^{T} p(\boldsymbol{y}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \tag{3.76}$$

We can expand the term inside the product in equation 3.76, by using the structure inherent in the state space model,

$$p(\boldsymbol{y}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \;=\; \iint \underbrace{p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})}_{\text{Observation}} \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\text{Transition}} \underbrace{p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Filtered state}} \, d\boldsymbol{x}_{t-1} \, d\boldsymbol{x}_t \tag{3.77}$$

where we recognise the three terms as the observation probability, the transition probability, and the filtered state distribution at $t-1$. Here we have already integrated out the GP latent function variable $\boldsymbol{f}_t$ from the transition probability,

$$\begin{aligned} p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) \;&=\; \int \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{f}_t, \boldsymbol{\theta})}_{\text{Gaussian noise}} \underbrace{p(\boldsymbol{f}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\mathcal{GP}\text{ prediction}} \, d\boldsymbol{f}_t \\ &=\; \mathcal{N}\left(\boldsymbol{x}_t;\; \mathfrak{m}(\boldsymbol{x}_{t-1}),\, \mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon\right) \end{aligned} \tag{3.78}$$

To compute equation 3.77 analytically we must approximate the filtered state distribution with a Gaussian.

$$\begin{aligned} p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \;&\approx\; q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \\ q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \;&=\; \mathcal{N}\left(\boldsymbol{x}_{t-1};\; \boldsymbol{\mu}_{t-1|t-1},\; \Sigma_{t-1|t-1}\right) \end{aligned} \tag{3.79}$$

We use the subscript notation '$t-1 \mid t-1$' to indicate that these are moments for the state at time $t-1$ using the information (observations) up to, and including, time $t-1$. We can either solve the integral over $\boldsymbol{x}_{t-1}$ or $\boldsymbol{x}_t$ first. If we could compute all the integrals analytically, clearly it would make no difference which order we solved them in. However, in this case we are solving the integrals approximately, and the approximations we make will differ depending on the order in which we consider the integrals. For now we choose to solve the integral over $\boldsymbol{x}_{t-1}$ first, followed by the integral over $\boldsymbol{x}_t$. The alternative approach, integrating over $\boldsymbol{x}_t$ then over $\boldsymbol{x}_{t-1}$, can be done using importance sampling for the second integral; this is more accurate but computationally slower, especially as we also require derivatives w.r.t. $\boldsymbol{\theta}$ in order to optimise $\boldsymbol{\theta}$. Using the moment-matching approximation of equation 3.79 for the integral over $\boldsymbol{x}_{t-1}$ in equation 3.77 gives,

$$p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \;\approx\; \int \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\mathcal{GP}\text{ prediction + noise; eq. 3.78}} \underbrace{q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Moment-matched Gaussian; eq 3.79}} \, d\boldsymbol{x}_{t-1} \tag{3.80}$$

This integral corresponds to averaging a GP prediction when its test input $\boldsymbol{x}_{t-1}$ is Gaussian. The resulting distribution is generally non-Gaussian (using a GP with a linear kernel is an exception), however, the first two moments can be computed analytically for the SE covariance function (amongst a few others) (Girard et al., 2003), and so we approximate $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ with a moment-matched Gaussian distribution,

$$\begin{aligned} p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) &\approx q_{\text{time}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \\ q_{\text{time}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) &= \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{t|t-1},\ \Sigma_{t|t-1}\right) \end{aligned} \tag{3.81}$$

where,

$$\boldsymbol{\mu}_{t|t-1} = \iint \boldsymbol{x}_t\ p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\ q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})\ d\boldsymbol{x}_{t-1}\ d\boldsymbol{x}_t \tag{3.82}$$

$$\Sigma_{t|t-1} = \iint \boldsymbol{x}_t \boldsymbol{x}_t^T\ p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\ q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})\ d\boldsymbol{x}_{t-1}\ d\boldsymbol{x}_t\ -\ \boldsymbol{\mu}_{t|t-1}\boldsymbol{\mu}_{t|t-1}^T \tag{3.83}$$

See Girard et al. (2003) or Deisenroth (2009) for the solutions to these moment integrals. Note, we have now used the subscript '$t \mid t-1$' as these are the moments of $\boldsymbol{x}_t$ but still only using observations up to time $t-1$. Substituting the approximate result of the integral over $\boldsymbol{x}_{t-1}$ (equation 3.81) into equation 3.77 gives the integral over $\boldsymbol{x}_t$ as

$$p(\boldsymbol{y}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \approx \int \underbrace{p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)}_{\text{Observation probability = Gaussian}} \underbrace{q_{\text{time}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1})}_{\text{Gaussian; eqs: 3.80,3.81}}\ d\boldsymbol{x}_t \tag{3.84}$$

The restrictions we placed on the observation model mean that $\boldsymbol{y}_t$ and $\boldsymbol{x}_t$ have a linear relationship ($\boldsymbol{y}_t = C\boldsymbol{x}_t + \nu_t$ — equation 3.4). Thus the integrand in equation 3.84 equates to a joint Gaussian,

$$p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)\, q_{\text{time}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{y}_t \end{bmatrix};\ \begin{bmatrix} \boldsymbol{\mu}_{t|t-1} \\ C\boldsymbol{\mu}_{t|t-1} \end{bmatrix},\ \begin{bmatrix} \Sigma_{t|t-1} & \Sigma_{t|t-1}C^T \\ C\Sigma_{t|t-1} & C\Sigma_{t|t-1}C^T + \Sigma_\nu \end{bmatrix}\right) \tag{3.85}$$

which we can integrate exactly to yield an approximation to the marginal likelihood term,

$$p(\boldsymbol{y}_t \mid \boldsymbol{y}_{1:t-1}) \approx \mathcal{N}\left(\boldsymbol{y}_t;\ C\boldsymbol{\mu}_{t|t-1},\ C\Sigma_{t|t-1}C^T + \Sigma_\nu\right) \tag{3.86}$$

and, using conditioning, an expression for the filtered state at time t,

$$\begin{aligned} q_{\text{filter}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) &= \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{t|t},\ \Sigma_{t|t}\right) \\ \boldsymbol{\mu}_{t|t} &= \boldsymbol{\mu}_{t|t-1} + \Sigma_{t|t-1}C^T\left(C\,\Sigma_{t|t-1}C^T + \Sigma_\nu\right)^{-1}(\boldsymbol{y}_t - C\boldsymbol{\mu}_{t|t-1}) \\ \Sigma_{t|t} &= \Sigma_{t|t-1} - \Sigma_{t|t-1}C^T\left(C\,\Sigma_{t|t-1}C^T + \Sigma_\nu\right)^{-1}C\Sigma_{t|t-1} \end{aligned} \tag{3.87}$$

This gives us a complete method for computing an approximation to the marginal likelihood by using a single forward sweep over the data. The method, as outlined above, uses the GP assumed density filter (GP-ADF) (Deisenroth et al., 2009) to infer an approximation to the filter distribution on the latent states. We then make use of the linear observation model to compute the marginal likelihood without further approximation. Furthermore, we can find the derivatives of each of the steps outlined above w.r.t. the parameters $\boldsymbol{\theta}$ and the previous moments; thus we can fit the whole model using gradient descent on the approximate marginal likelihood. The key required derivatives are presented at the

end of the chapter. A summary of the algorithm, in the future referred to as 'the direct method', is shown in algorithm 1.

Algorithm 1 Direct Optimisation with Moment Matching

- Pre-select pseudo-inputs $\tilde{X}$ to span data and initialise parameters, $\boldsymbol{\theta}$
- Run gradient descent optimisation until termination criteria are met

  One optimisation step:
  - Compute negative log marginal likelihood contribution for $t = 1$
  - For $t = 2 : T$:
    - Compute moments of GP prediction at t given Gaussian on $t-1$, $\boldsymbol{\mu}_{t|t-1}$, $\Sigma_{t|t-1}$
    - Compute negative log marginal likelihood contribution (equation 3.86),
      $$-\log \mathcal{N}\left(\boldsymbol{y}_t;\ C\boldsymbol{\mu}_{t|t-1},\ C\Sigma_{t|t-1}C^T + \Sigma_\nu\right)$$
    - Compute approximate filter distribution (equation 3.87),
      $$q_{\text{filter}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{t|t},\ \Sigma_{t|t}\right)$$
    - Compute derivatives of both terms w.r.t. $\boldsymbol{\theta}$ and previous filter moments $\boldsymbol{\mu}_{t-1|t-1}$, $\Sigma_{t-1|t-1}$
    - Using the chain rule combine derivatives from previous steps to obtain
      $$\frac{\partial -\log p(\boldsymbol{y}_{1:t} \mid \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$
- Return optimised parameters

3.5.1 Analysis

Figure 3.13 shows optimisation performance for the Direct algorithm on three systems: the 1D kink function, the 4D cart & pendulum system, and the 10 unicycle system. In all cases the dynamics model was initialised using the hyperparameters from an initial GP trained on the observed data in the standard (non-latent variable) manner. The figure shows how both training and test performance changes with optimisation iteration. The measure of training performance is the approximation to the negative log marginal likelihood calculated by the Direct algorithm (equations 3.76 and 3.86). The test performance is measured using the same statistic as described in the variational approach section.

The figure shows the optimisation converging for around 750 function evaluations or fewer based on the test performance. The number of minimisation steps required for optimisation is strongly dependent on the system being modelled; not just dependent on its dimension but on other features capturing the complexity of the transition function. The one dimensional system shows a section of the optimisation where the test metric experiences some 'turbulence' which does not appear in the training NLML, although the training NLML has not yet converged. The optimisation progress breaks through this phase but this behaviour warns of the dangers of early stopping. On the other hand, the test metric is still mostly decreasing in this section and so perhaps one would not read too much into it. More importantly, none of the systems show any signs of long term over-fitting as we might expect from a maximum (marginal) likelihood method.

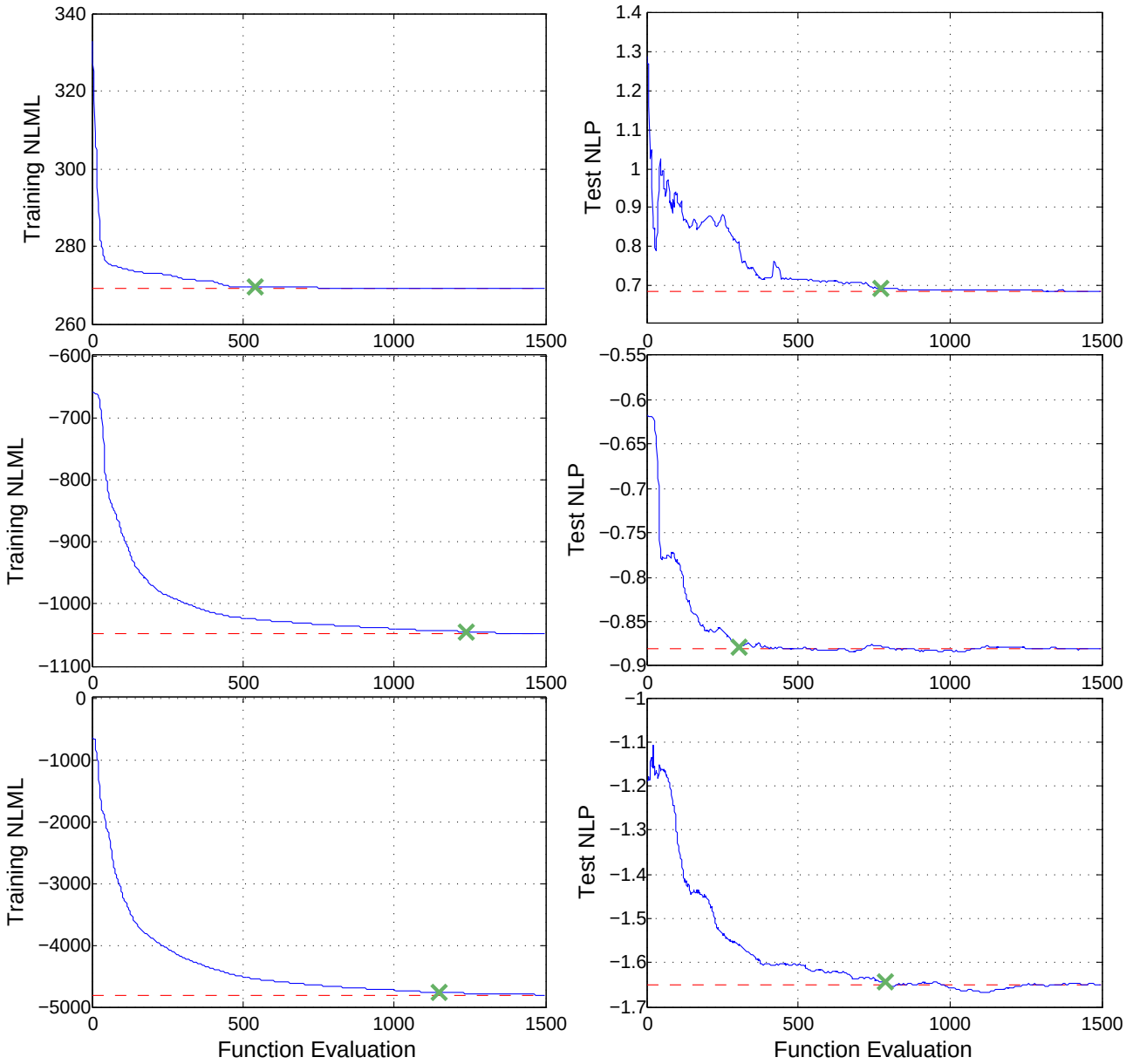


Figure 3.13: Training and test performance for the 1D kink function (top), 4D cart & pendulum system (middle), and the 10D unicycle system (bottom). The y -axis of the left hand plots shows the Direct algorithm's approximation to the training negative log marginal likelihood; the y-axis on the right hand plots shows the negative log probability of a test set under the model. The red dashed line shows the final value reached after 1500 minimisation steps, the green cross shows the point where 99% of the total function value decrease is reached. Note that the training and test scores are measuring different probabilities and thus are not comparable.

Figure 3.14 shows wall clock timings in seconds for the computation of the approximate negative log marginal likelihood and its derivatives w.r.t. the parameters for systems with dimensionality from one to ten. The times are shown for a system of ten pseudo-points and a single observed trajectory of twenty time steps. The figure shows a nearly perfect quadratic dependence on dimension, which is to be expected: at the heart of the algorithm we must make D GP predictions (one for each state dimension) each of which is linear in state dimension. The timings shown in the figure are for a single evaluation of the derivatives but we can estimate the time to train a full model by using around 750 iterations, as seen from figure 3.13. On top of this we would typically require more than a single observed trajectory of twenty time steps to accurately learn the model. If we assume we have twenty

separate observed trajectories, each of twenty steps long, and run training for 750 iterations then total training time runs from around fifty minutes to ten hours for one to ten dimensional systems. These times are also quadratically dependent on the number of pseudo points used; the times shown are based on a model using ten points. It is likely that more than ten points would be required for complex systems, particularly for higher dimensional systems where more points are needed to sufficiently cover the state space.

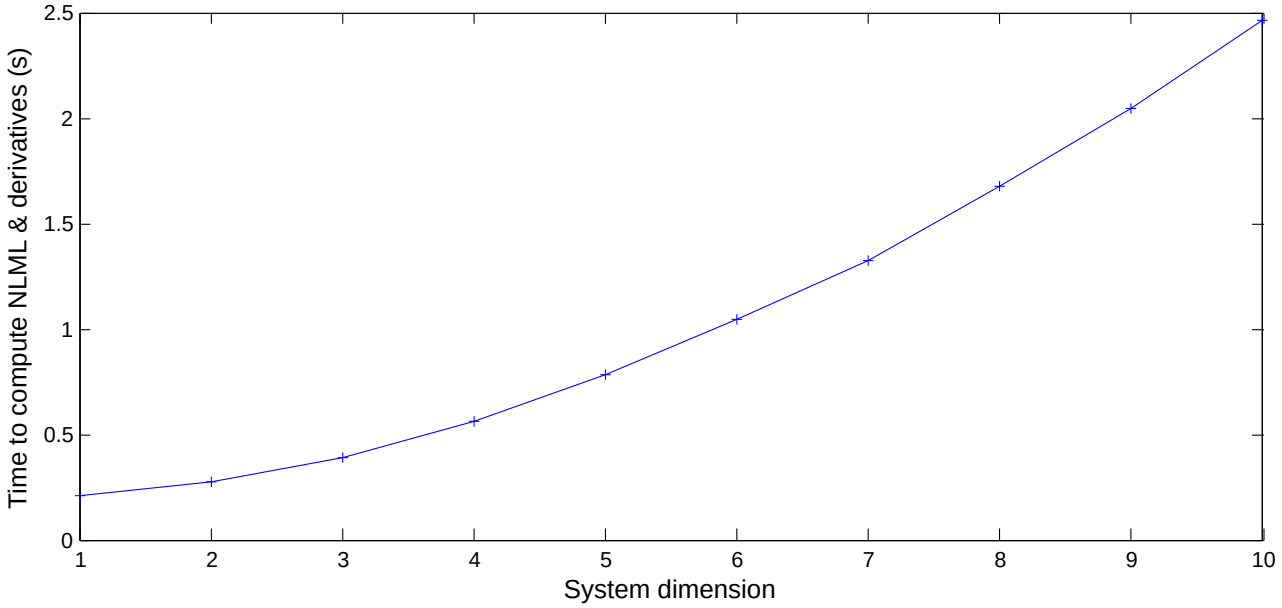


Figure 3.14: Wall clock timings in seconds for the Direct algorithm on systems with varying dimensionality. The timings are how long it takes the algorithm to compute the approximate NLML and its derivatives for an observed trajectory of 20 time steps and with 10 pseudo-points.

3.6 Analytic Expectation Maximisation

In this section we look at another approximate-analytic approach, this time based on the expectation maximisation (EM) algorithm. The EM approach splits the problem into two parts: in the E step we find an approximate posterior distribution over the latent states, $q(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \approx p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, and in the M step we optimise an approximate bound on the marginal likelihood, using the distribution found in the E step. In contrast to the Direct method discussed in the previous section, in the E step here we (approximately) compute the full smoothing posterior, which requires both forward and backward sweeps. This is much more complicated than the filtering-only Direct approach, however, it has an advantage when it comes to optimising the parameters, as a number of them can be solved for rather than optimised by gradient descent.

3.6.1 E step using assumed density smoothing

The E step of Turner et al. (2010) used the GP-ADF (Deisenroth et al., 2009) to run forward-filtering, as we did in algorithm 1, but they then follow this with a new backward-smoothing step. On the forward sweep, joint Gaussian distributions are moment matched to each pair of sequential states, which effectively linearises the transition function about each filtered state. This linearisation is then

reused in the backward sweep to update the state location, despite the fact that the smoothed state might be significantly far away from the filtered state, where the linearisation was made. This weakness also appears in the extended Kalman filter (EKF) and can lead to disastrous results in both cases as we will show.

First we will derive the E step of Turner et al. (2010), which uses the GP-ADF (Deisenroth et al., 2009). The goal of the E step is to find an approximation to the latent state posterior,

$$q(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;\approx\; p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \tag{3.88}$$

This can be achieved using a forward-backward algorithm. In the forward sweep we recursively compute the filter distributions $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t})$ by using the following relation,

$$p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \;\propto\; \underbrace{p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})}_{\text{Observation}} \int \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\mathcal{GP}\text{ Transition}} \underbrace{p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Filtered state at } t-1} d\boldsymbol{x}_{t-1} \tag{3.89}$$

where the normalisation term is $p(\boldsymbol{y}_t \mid \boldsymbol{y}_{1:T-1}, \boldsymbol{\theta})$. Equation 3.89 is typically split into two parts: the time update,

$$p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \;=\; \int \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\mathcal{GP}\text{ Transition}} \underbrace{p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Filtered state at } t-1} d\boldsymbol{x}_{t-1} \tag{3.90}$$

and the measurement correction,

$$p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \;\propto\; p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})\; p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta}) \tag{3.91}$$

Equation 3.89 is the same as the marginal likelihood equation (3.77) without the integral over $\boldsymbol{x}_t$. Thus we can solve equation 3.89 by using the same set of approximations as we did in the direct approach, equations 3.79 to 3.84: first we approximate the filter distribution at $t-1$ with a Gaussian, $q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ (equation 3.79), secondly we find the moments of the time update distribution using the uncertain test-input GP equations. Thirdly, using these moments we approximate the time update $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ with a Gaussian, $q_{\text{time}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ (equation 3.81). Finally, we combine the time-update distribution with the observation at time t and use conditioning to find a Gaussian approximation to the filter distribution at t, $q_{\text{filter}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})$.

The forward sweep only conditions on observed data from the current and previous time steps. Forming a posterior over a latent state which conditions on all observations, i.e. also on future measurements, is known as smoothing. The filter posteriors are updated to smoothing posteriors during the backward step. This backward sweep is much more complicated than the forward sweep as it involves the inversion of the GP dynamics:

$$\underbrace{p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})}_{\text{Smoothing distribution at } t-1} \;=\; \int \underbrace{p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Inverse GP transition}} \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})}_{\text{Smoothing distribution at } t} d\boldsymbol{x}_t \tag{3.92}$$

Note, due to the Markov structure of the state space model (figure 3.3) $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) = p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$. Unfortunately there are no analytic results for the distribution on the test input to a GP for a particular test output. Thus, even if the smoothing distribution at time t is Gaussian, we cannot solve the integral in equation 3.92.

So far we have approximated the marginal filter distributions $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})$ with Gaussians $q_{\text{filter}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})$, however, we could extend the approximation and fit joint Gaussian distributions to pairs of adjacent states, $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$. In step two of the filter step outlined above, we compute the moments of a GP predictive posterior when the test input is Gaussian distributed. We can further compute the input-output covariance term between the filter distribution at time $t-1$ and the time-update distribution $q_{\text{time}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$, giving an approximate joint distribution,

$$\begin{aligned} p(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) &\approx q_{\text{joint}}(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \\ q_{\text{joint}}(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) &= \mathcal{N}\left(\begin{bmatrix} \boldsymbol{x}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix}; \begin{bmatrix} \boldsymbol{\mu}_{t-1|t-1} \\ \boldsymbol{\mu}_{t|t-1} \end{bmatrix}, \begin{bmatrix} \Sigma_{t-1|t-1} & C_{t-1,t|t-1} \\ C_{t-1,t|t-1}^T & \Sigma_{t|t-1} \end{bmatrix} \right) \end{aligned} \tag{3.93}$$

where $C_{t-1,t|t-1}$ is the input-output covariance term for a GP with a Gaussian distributed test point (see equation 1.30 and Deisenroth (2009)). Given that we have a joint Gaussian distribution in equation 3.93 we can condition on $\boldsymbol{x}_t$ which gives us an approximation to the inverse transition term in equation 3.92,

$$\begin{aligned} p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) &\approx q_{\text{bkwd}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \\ q_{\text{bkwd}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) & \\ &= \mathcal{N}\left(\boldsymbol{x}_{t-1};\ \boldsymbol{\mu}_{t-1|t-1} + C_{t-1,t|t-1}\, \Sigma_{t|t-1}^{-1}(\boldsymbol{x}_t - \boldsymbol{\mu}_{t|t}),\ \Sigma_{t-1|t-1} - C_{t-1,t|t-1}\, \Sigma_{t|t-1}^{-1}\, C_{t-1,t|t-1}^T\right) \end{aligned} \tag{3.94}$$

The latent state $\boldsymbol{x}_t$ appears linearly in the mean of equation 3.94 and thus can be integrated against a Gaussian as required in equation 3.92. Therefore, by substituting $q_{\text{bkwd}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ for $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ in equation 3.92 we can solve the integral and find an approximation to the smoothed state distribution,

$$\begin{aligned} p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) &\approx q_{\text{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \\ q_{\text{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) &= \mathcal{N}\left(\boldsymbol{x}_{t-1};\ \boldsymbol{\mu}_{t-1|T}^{\text{ADS}},\ \Sigma_{t-1|T}^{\text{ADS}}\right) \end{aligned} \tag{3.95}$$

with,

$$\begin{aligned} \boldsymbol{\mu}_{t-1|T}^{\text{ADS}} &= \boldsymbol{\mu}_{t-1|t-1} + J_{t-1}\,(\boldsymbol{\mu}_{t|T}^{\text{ADS}} - \boldsymbol{\mu}_{t|t-1}) \\ \Sigma_{t-1|T}^{\text{ADS}} &= \Sigma_{t-1|t-1} + J_{t-1}\,(\Sigma_{t|T}^{\text{ADS}} - \Sigma_{t|t-1})\, J_{t-1}^T \\ J_{t-1} &= C_{t-1,t|t-1}\, \Sigma_{t|t-1}^{-1} \end{aligned} \tag{3.96}$$

Equations 3.95 and 3.96 provide a method for finding an approximation to the smoothing posteriors on the latent states. We will look at a number of other methods for computing an approximation to the smoothing posteriors and so we use the superscript to differentiate them: here, 'ADS' refers to assumed density smoothing.

We can also compute the covariance between pairs of adjacent states, $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$, under the smoothing distributions,

$$\mathbb{C}\left[\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1_T}, \boldsymbol{\theta}\right] = C_{t-1,t|t-1}\, \Sigma_{t|T} \tag{3.97}$$

Equations 3.89 to 3.97 describe the 'assumed density smoothing' (ADS) method for finding an approximation to the latent state posterior; this method is summarised in algorithm 2.

Algorithm 2 Assumed Density Smoothing E Step

Filtering

- Compute first filter posterior $p(\boldsymbol{x}_1 \mid \boldsymbol{y}_1, \boldsymbol{\theta})$
- For $t = 2 : T$:
  - Time update: Compute moments $(\boldsymbol{\mu}_{t|t-1},\ \Sigma_{t|t-1},\ C_{t-1,t|t-1})$ for the pairwise joint Gaussian approximation, equation 3.93, $q_{\text{joint}}(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$, using equations 1.28 to 1.30
  - Measurement correction: Compute approximate filter distribution (equation 3.87),
$$q_{\text{filter}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{t|t},\ \Sigma_{t|t}\right)$$

Smoothing

- Last smoothing posterior $p(\boldsymbol{x}_T \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$ is given by last filter posterior
- For $t = T-1 : 1$:
  - Compute moments $(\boldsymbol{\mu}_{t|T},\ \Sigma_{t|T},\ C_{t,t+1|T})$ for joint smoothing Gaussian approximation, using equations 3.96 and 3.97

It was stated earlier that this method suffers from a serious weakness and we will now demonstrate this. The weakness arises from the use of the joint Gaussian distribution calculated during the forward filtering pass (equation 3.93) in the backward sweep (recall that we cannot compute the distribution on a GP input for a given output distribution, hence we had to reuse the joint distribution). The approximation of a joint Gaussian distribution implies an implicit linearisation of the GP transition function: a linear relationship exists between $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$ under the filter distribution and the time-update distribution. It is straightforward to find the parameters of this linear relationship: let,

$$\begin{aligned} \boldsymbol{x}_{t-1} &= A\,\boldsymbol{x}_t + \boldsymbol{b} + \boldsymbol{\epsilon}'_{t-1} \\ \boldsymbol{\epsilon}'_{t-1} &\sim \mathcal{N}(0,\ \Sigma_{\epsilon'}) \end{aligned} \tag{3.98}$$

then,

$$p(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{x}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix};\ \begin{bmatrix} A\,\boldsymbol{\mu}_{t|t-1} + \boldsymbol{b} \\ \boldsymbol{\mu}_{t|t-1} \end{bmatrix},\ \begin{bmatrix} A\,\Sigma_{t|t-1}\,A^T + \Sigma_{\epsilon'} & A\,\Sigma_{t|t-1} \\ \Sigma_{t|t-1}\,A^T & \Sigma_{t|t-1} \end{bmatrix}\right) \tag{3.99}$$

By comparing the covariance terms in equations 3.93 and 3.99 we can find A,

$$A = C_{t-1,t|t-1}\,\Sigma_{t|t-1}^{-1} = J_{t-1} \tag{3.100}$$

from the means,

$$\boldsymbol{b} = \boldsymbol{\mu}_{t-1|t-1} - J_{t-1}\,\boldsymbol{\mu}_{t|t-1} \tag{3.101}$$

and finally,

$$\Sigma_{\epsilon'} = \Sigma_{t-1|t-1} - J_{t-1}\,\Sigma_{t|t-1}\,J_{t-1}^T \tag{3.102}$$

This linearisation is more complex than that used by the EKF, as it takes into account the whole distribution $q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ rather than just the mean. However, it is still a linearisation *about the filter distribution* $q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$. In the backward sweep we use this linearisation to compute the approximate smoothing distribution $q_{\text{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$ from the smoothing distribution at the

subsequent time step, despite the fact that $q_{\text{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$ maybe very far away from $q_{\text{filter}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$, the site of the linearisation.

Figure 3.15 shows how this reuse of the linearisation can lead to poor results. The top plot shows the time update in a particular filter step; the bottom plot shows the corresponding smoothing step. Reuse of the linearisation (shown by the magenta line) pushes the smoothing distribution at $t-1$ beyond the peak in the transition function. Furthermore, the covariance between the two adjacent smoothing states, $\mathbb{C}[\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}]$, is also based on the linearisation. In the case of figure 3.15 this results in the covariance having the wrong sign. Note that it is the presence of process noise which is causing the backward linearisation to be steeper than one would expect from the shape of the transition function. If the process noise level is reduced then the smoothing distribution is pushed even further along the x-axis.

One simple adaptation we could make to reduce the problem is to iterate the filtering/smoothing sweeps multiple times and to introduce a damping factor. For many cases these steps will mitigate the effect of the problem described above; however, this is not guaranteed and equally, there will be many cases where the problem persists.

3.6.2 E Step using Expectation Propagation

We can completely remove the weakness described in the previous section, and hence greatly improve the GPIL (Gaussian Process Inference and Learning (Turner et al., 2010)) algorithm, by using Expectation Propagation (EP) (Minka, 2001) to compute the approximate posterior. EP has been applied for inference in nonlinear dynamical systems previously, for example (Ypma and Heskes, 2005; Yu et al., 2006). Recent work by Deisenroth and Mohamed (2012) applied EP to Gaussian Processes transition functions, although the authors focus purely on inference and do not tackle parameter learning. Here we modify their EP approach and combine it with the M step from GPIL.

We start by first applying belief propagation to the state space model but show that we cannot compute the required messages exactly. We then explain how EP can be used as an approximation. Belief propagation (BP) (Pearl, 1988) is a message passing algorithm which can be used for inference in Bayesian networks. The messages themselves are (potentially unnormalised) probability distributions and are computed according to a particular formula, explained below. The messages are passed between factors and variables in a factor graph until convergence is reached. For trees, such as the state space model we are considering in this chapter, BP is an exact algorithm, which makes it very attractive. Unfortunately, we cannot compute the true messages (which will be denoted α^* and β^* here) and hence we must use an approximation: expectation propagation.

We can factorise the complete likelihood as,

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:T} \mid \boldsymbol{\theta}) = \prod_{t=1}^{T} h_t \tag{3.103}$$

$$h_t = \begin{cases} p(\boldsymbol{x}_1, \boldsymbol{y}_1 \mid \boldsymbol{\theta}) & \text{for } t = 1 \\ p(\boldsymbol{x}_t, \boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) & \text{otherwise} \end{cases} \tag{3.104}$$

which can be represented as a factor graph, as shown in figure 3.16.

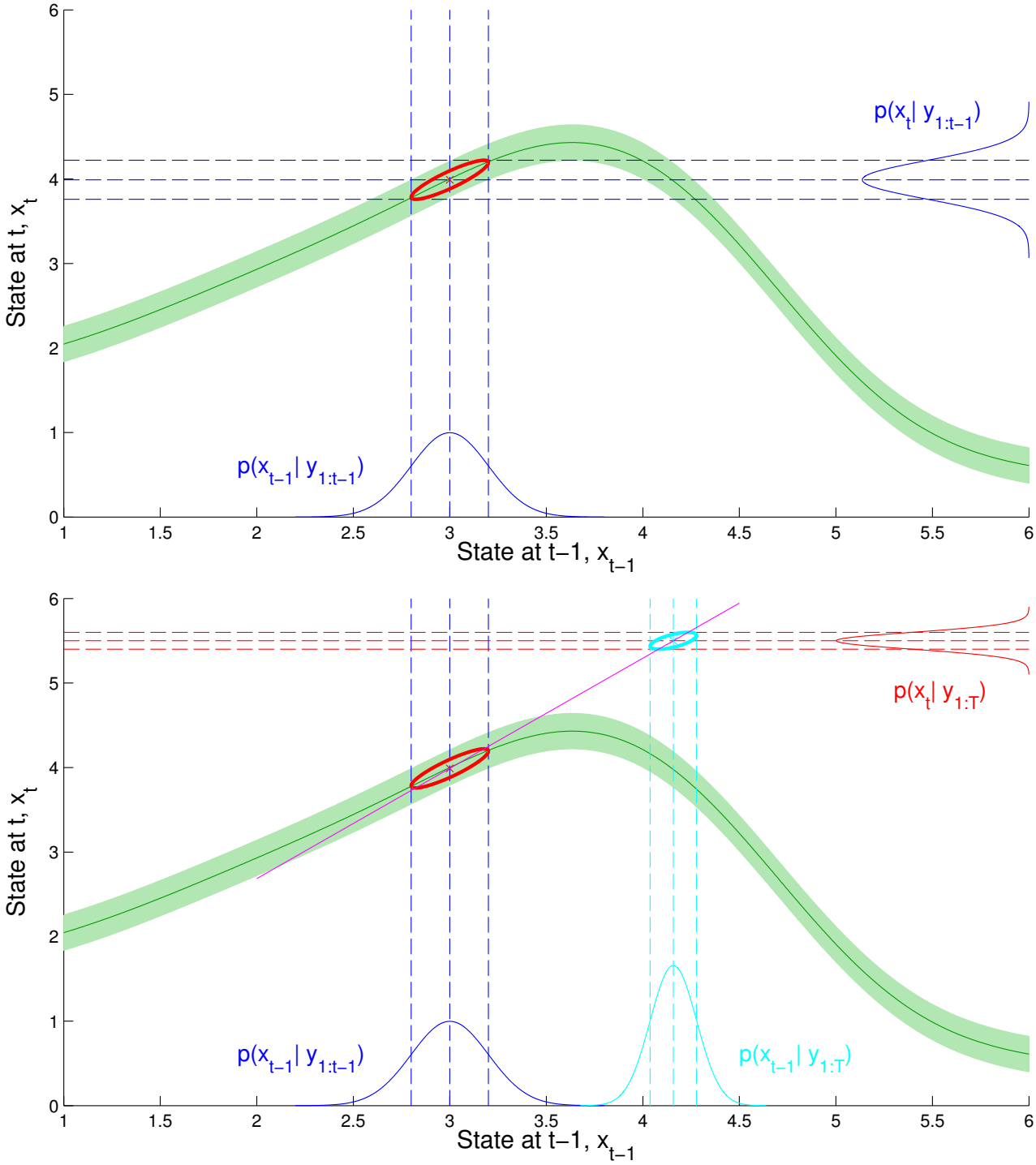


Figure 3.15: Example of poor performance in the GP-ADF based E step due to the reuse of the linearisation from the filtering sweep in the smoothing sweep. The green GP posterior shows the current belief over transition functions for a 1D system. The top plot shows, in blue, the filter distribution at $t-1$, the corresponding predictive distribution at time t, and, in red, their joint distribution. In the bottom plot the red Gaussian shows the smoothed distribution at time t and the cyan Gaussian the smoothed distribution at $t-1$ calculated by equation 3.96. The magenta line shows the backward implicit linearisation.

In belief propagation the message sent from a variable to a factor is the product of messages the variable has received from all the factors except the factor to which the message will be sent. For the case where a variable is connected to two factors it is therefore 'transparent', merely passing a message received from one factor on to the other factor. Thus, for simplicity, we can define messages

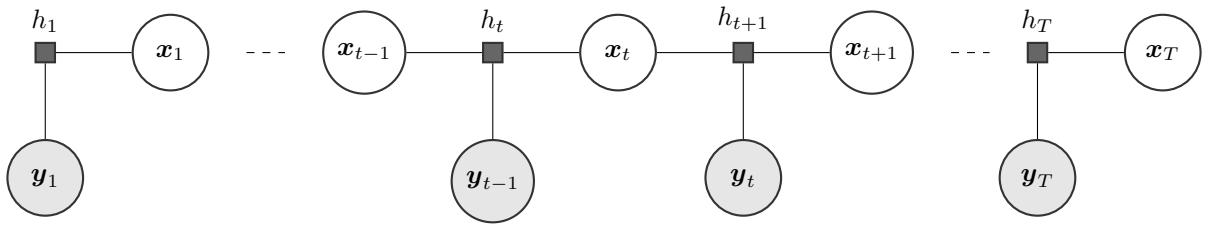


Figure 3.16: Factor graph representation of the state space model. Note, we could equivalently define the factor graph such that the factors are on the right hand side of the corresponding variable node, or even define a symmetric version; in all cases the results are the same.

directly between factors, as shown in figure 3.17. We have termed forward messages, that is those being sent forward in time, α and backward messages β. As the observations are fixed and Gaussian we do not need to consider messages between the observation nodes and the factors: we can treat the observations as constants in the factors where they appear.

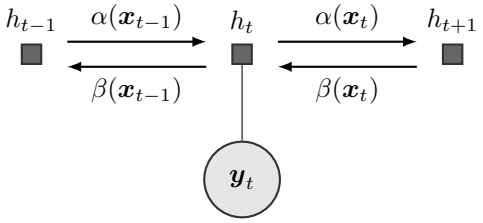


Figure 3.17: Messages in the state space model factor graph. The forward messages are specified by α and the backward by β.

In this setup, the current estimate of the marginal over a hidden state is given by the product of the two messages at a particular time step,

$$q(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}) \;\propto\; \alpha(\boldsymbol{x}_t)\,\beta(\boldsymbol{x}_t) \tag{3.105}$$

The two-slice marginal,

$$\begin{aligned} q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}) \;&\propto\; \alpha(\boldsymbol{x}_{t-1})\, h_t(\boldsymbol{x}_{t-1}, \boldsymbol{y}_t, \boldsymbol{x}_t)\,\beta(\boldsymbol{x}_t) && (3.106)\\ &\propto\; \alpha(\boldsymbol{x}_{t-1})\, p(\boldsymbol{x}_t, \boldsymbol{y}_t \mid \boldsymbol{x}_{t-1})\,\beta(\boldsymbol{x}_t) && (3.107)\\ &\propto\; \alpha(\boldsymbol{x}_{t-1})\, p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})\, p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)\,\beta(\boldsymbol{x}_t) && (3.108) \end{aligned}$$

As we stated earlier, for trees these estimates will converge to the true posteriors if we can compute the correct messages. We will now attempt to derive the form of these messages. We will introduce a superscript asterisk to the message notation to indicate that these are the true messages, as opposed to the approximate version we will introduce shortly. The rules of belief propagation state that the true forward message from h_t to h_{t+1}, denoted $\alpha_t^*(\boldsymbol{x}_t)$, is calculated by taking the product of the message

from h_{t-1} with the factor potential $h_t(\boldsymbol{x}_{t-1}, \boldsymbol{y}_t, \boldsymbol{x}_t)$, and integrating out $\boldsymbol{x}_{t-1}$,

$$\alpha_t^*(\boldsymbol{x}_t) = \int \underbrace{\alpha_{t-1}^*(\boldsymbol{x}_{t-1})}_{\text{message } h_{t-1}\to h_t} \underbrace{h_t(\boldsymbol{x}_{t-1}, \boldsymbol{y}_t, \boldsymbol{x}_t)}_{h_t \text{ factor potential}} d\boldsymbol{x}_{t-1} \tag{3.109}$$

$$= p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta}) \int \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\mathcal{GP} \text{ prediction}} \underbrace{\alpha_{t-1}^*(\boldsymbol{x}_{t-1})}_{\text{message } h_{t-1}\to h_t} \tag{3.110}$$

Equation 3.110 once again involves finding the average GP predictive posterior when the test input is uncertain. As has been stated previously, even if the message $\alpha_{t-1}^*(\boldsymbol{x}_{t-1})$ is Gaussian distributed the integral in equation 3.110 will result in a complex distribution over $\boldsymbol{x}_t$, which is certainly not Gaussian (barring a few exceptional cases, e.g. a linear kernel). This is due to the nonlinearity of the GP transition function. The corresponding message update equation for β_{t-1} is

$$\beta_{t-1}^*(\boldsymbol{x}_{t-1}) = \int \underbrace{\beta_t^*(\boldsymbol{x}_t)}_{\text{message } h_{t+1}\to h_t} \underbrace{h_t(\boldsymbol{x}_{t-1}, \boldsymbol{y}_t, \boldsymbol{x}_t)}_{h_t \text{ factor potential}} d\boldsymbol{x}_t \tag{3.111}$$

$$= \int \beta_t^*(\boldsymbol{x}_t) \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\mathcal{GP} \text{ prediction}} p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, d\boldsymbol{x}_t \tag{3.112}$$

If β_t^* is Gaussian distributed then equation 3.112 can be solved due to the linear observation model: first we combine the transition and observation using their joint Gaussianity,

$$\begin{aligned} p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{y}_t, \boldsymbol{\theta}) &= \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{t|xy},\ \Sigma_{t|xy}\right) \\ \boldsymbol{\mu}_{t|xy} &= \mathfrak{m}(\boldsymbol{x}_{t-1}) + (\mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon) C^T \left(C\left(\mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon\right) C^T + \Sigma_\nu\right)^{-1} (\boldsymbol{y}_t - C\mathfrak{m}(\boldsymbol{x}_{t-1})) \\ \Sigma_{t|xy} &= \mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon - (\mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon) C^T \left(C\left(\mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon\right) C^T + \Sigma_\nu\right)^{-1} C\left(\mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon\right) \\ &= \left(\left(\mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon\right)^{-1} + C^T \Sigma_\nu^{-1} C\right)^{-1} \end{aligned} \tag{3.113}$$

then equation 3.112 is just the normalisation constant of the product of two Gaussians,

$$\beta_{t-1}^*(\boldsymbol{x}_{t-1}) = |\Sigma_{t|xy} + \Sigma_\beta|^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_{t|xy} - \boldsymbol{\mu}_\beta)^T (\Sigma_{t|xy} + \Sigma_\beta)^{-1} (\boldsymbol{\mu}_{t|xy} - \boldsymbol{\mu}_\beta)\right) \tag{3.114}$$

However, once again, this is a complex distribution (this time in $\boldsymbol{x}_{t-1}$) and is certainly not Gaussian – note the presence of $\boldsymbol{x}_{t-1}$ inside the inverse in equation 3.114. Therefore, we cannot solve the required update equations for the α and β messages, even if we start with Gaussian messages; thus we need to use an approximate inference scheme. Here we look at using expectation propagation (EP).

In EP, each of the messages are approximated by an exponential family distribution, here a Gaussian.

$$\begin{aligned} \alpha_t(\boldsymbol{x}_t) &\propto \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{\alpha_t},\ \Sigma_{\alpha_t}\right) \approx \alpha_t^*(\boldsymbol{x}_t) \\ \beta_t(\boldsymbol{x}_t) &\propto \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{\beta_t},\ \Sigma_{\beta_t}\right) \approx \beta_t^*(\boldsymbol{x}_t) \end{aligned} \tag{3.115}$$

Note that the messages do not need to normalise, hence the proportionality in equation 3.115. Furthermore, they do not even need to be 'proper' Gaussians in that they can have non-positive definite covariance matrices so long as any expressions which are distributions (for example the product $\alpha_t(\boldsymbol{x}_t)\,\beta_t(\boldsymbol{x}_t)$) do have valid covariance matrices.

Suppose message m_i is a function of $\boldsymbol{x}_i$. Under expectation propagation, we update message m_i using the following steps,

1. Compute the 'cavity distribution' $q(\boldsymbol{x}_i \mid \boldsymbol{y}_{1:T})^{\backslash m_i}$. This is the distribution at the receiving variable node without using message m_i
2. Compute the first and second moments of the true message, m_i^*, multiplied by the cavity distribution

$$\tilde{\boldsymbol{\mu}}_i \;=\; \mathbb{E}\left[m_i^*\, q(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})^{\backslash m_i}\right], \qquad \tilde{\Sigma}_i \;=\; \mathbb{V}\left[m_i^*\, q(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})^{\backslash m_i}\right] \tag{3.116}$$

3. Using the computed moments, project the distribution onto the exponential family, here as Gaussian. This is done by setting the moments of the approximating Gaussian to the computed values, ignoring the presence of any higher order moments in the true distribution. We will denote this operation using 'proj{.}'
4. Find the updated message by dividing out the cavity distribution

A fundamental step in EP is that we do not project the true message m_i^* directly to an approximating distribution but rather the product of the true message and the cavity distribution. This usually leads to a more accurate approximation as the message is approximated in the context in which it will be used.

Recall that the latent state posteriors are given by,

$$q(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;\propto\; \alpha_t(\boldsymbol{x}_t)\, \beta_t(\boldsymbol{x}_t) \tag{3.117}$$

thus the cavity distribution for message α_t is just β_t and vice versa for β_t,

$$q^{\backslash \alpha_t}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; \beta_t; \qquad q^{\backslash \beta_t}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; \alpha_t \tag{3.118}$$

We can therefore write the EP message update equations as,

$$\alpha_t(\boldsymbol{x}_t) \;\propto\; \frac{\text{proj}\left\{\alpha_t^*(\boldsymbol{x}_t)\; q^{\backslash \alpha_t}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})\right\}}{q^{\backslash \alpha_t}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})} \tag{3.119}$$

$$\propto\; \frac{\text{proj}\left\{p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\; d\boldsymbol{x}_{t-1}\, \beta_t(\boldsymbol{x}_t)\right\}}{\beta_t(\boldsymbol{x}_t)} \tag{3.120}$$

$$\beta_{t-1}(\boldsymbol{x}_{t-1}) \;\propto\; \frac{\text{proj}\left\{\beta_{t-1}^*\; q^{\backslash \beta_{t-1}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})\right\}}{q^{\backslash \beta_{t-1}}(\boldsymbol{x}_{t-1})} \tag{3.121}$$

$$\propto\; \frac{\text{proj}\left\{\int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t)\; d\boldsymbol{x}_t\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\right\}}{\alpha_t(\boldsymbol{x}_{t-1})} \tag{3.122}$$

where we substituted in equations 3.110 and 3.112 for the true messages. For the projection step we need the moments of $p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\; d\boldsymbol{x}_{t-1}$ and $\alpha_{t-1}(\boldsymbol{x}_{t-1}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t)\; d\boldsymbol{x}_t$. From the rules of belief propagation we can recognise these equations as the EP approximations to the smoothing marginals,

$$Z_t\, q_{\text{smooth}}^{\text{EP}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\; \beta(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\; d\boldsymbol{x}_{t-1} \tag{3.123}$$

$$Z_{t-1}\, q^{\text{EP}}_{\text{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; \alpha_{t-1}(\boldsymbol{x}_{t-1}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t)\; d\boldsymbol{x}_t \tag{3.124}$$

where Z is the normalising constant, see equation 3.137. Thus,

$$\alpha_t(\boldsymbol{x}_t) \;\propto\; \frac{\text{proj}\left\{q^{\text{EP}}_{\text{smooth}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})\right\}}{\beta_t(\boldsymbol{x}_t)} \tag{3.125}$$

$$\beta_{t-1}(\boldsymbol{x}_{t-1}) \;\propto\; \frac{\text{proj}\left\{q^{\text{EP}}_{\text{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})\right\}}{\alpha_t(\boldsymbol{x}_{t-1})} \tag{3.126}$$

We will write the desired, first and second moments of equations 3.123 and 3.124 as $\boldsymbol{\mu}^{\text{EP}}_{t|T}$, $\Sigma^{\text{EP}}_{t|T}$, $\boldsymbol{\mu}^{\text{EP}}_{t-1|T}$, and $\Sigma^{\text{EP}}_{t-1|T}$ respectively. Unfortunately, none of these moments can be computed analytically. We can approximate the required moments by either using further Gaussian moment-matching, as was done in Deisenroth and Mohamed (2012), or by using sampling. These approaches are discussed in the following sections.

The final step in equations 3.120 and 3.122 is to divide out the other message. As we have projected the numerator onto a Gaussian distribution this is the ratio of two Gaussians, which is another (unnormalised) Gaussian, the moments of which can be computed in closed form,

$$\Sigma_{\alpha_t} \;=\; \left((\Sigma^{\text{EP}}_{t|T})^{-1} \;-\; \Sigma^{-1}_{\beta_t}\right)^{-1}, \qquad \boldsymbol{\mu}_{\alpha_t} \;=\; \Sigma_{\alpha_t}\left((\Sigma^{\text{EP}}_{t|T})^{-1}\, \boldsymbol{\mu}^{\text{EP}}_{t|T} \;-\; \Sigma^{-1}_{\beta_t}\, \boldsymbol{\mu}_{\beta_t}\right) \tag{3.127}$$

$$\Sigma_{\beta_{t-1}} \;=\; \left((\Sigma^{\text{EP}}_{t-1|T})^{-1} \;-\; \Sigma^{-1}_{\alpha_{t-1}}\right)^{-1}, \qquad \boldsymbol{\mu}_{\beta_{t-1}} \;=\; \Sigma_{\beta_{t-1}}\left((\Sigma^{\text{EP}}_{t-1|T})^{-1}\, \boldsymbol{\mu}^{\text{EP}}_{t-1|T} \;-\; \Sigma^{-1}_{\alpha_{t-1}}\, \boldsymbol{\mu}_{\alpha_{t-1}}\right) \tag{3.128}$$

It is possible that the resulting Gaussian messages can have negative variance. This is not necessarily a problem as the messages need not be proper distributions, but we must ensure that the expressions which are proper distributions, e.g. the smoothing marginal $\alpha(\boldsymbol{x}_t)\beta(\boldsymbol{x}_t)$, have positive variance. If an update would violate this condition, common options are to truncate the message in such a way as to remove the negative variance, to skip the update entirely, or to use a partial update (damping). We generally found the last of these options to be preferable and so use it for our experiments.

EP with moment-matching

In this section we look at solving the EP updates approximately by making further moment-matching approximations, as per Deisenroth and Mohamed (2012). We shall use the letters 'mm' in superscripts to label specific distributions/moments under this framework. The integral in the update for α_t, equation 3.120, is the familiar average of a GP prediction over a Gaussian distributed test input. As we know how to compute the moments of the result of this integral, we could project the integral to a Gaussian as we have done previously (e.g. equation 3.81),

$$q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1}, \boldsymbol{\theta}) \;\approx\; \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\; d\boldsymbol{x}_{t-1} \tag{3.129}$$

where,

$$q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1}, \boldsymbol{\theta}) \;=\; \mathcal{N}(\boldsymbol{x}_t;\; \boldsymbol{\mu}_{t|\alpha_{t-1}},\; \Sigma_{t|\alpha_{t-1}}) \tag{3.130}$$

We then just need to find the moments of our new EP smoothing approximation,

$$\tilde{Z}\, q^{\text{EPmm}}_{\text{smooth}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; \underbrace{p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})}_{\text{Gaussian observation}} \quad \underbrace{\beta_t(\boldsymbol{x}_t)}_{\text{Gaussian message}} \quad \underbrace{q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1}, \boldsymbol{\theta})}_{\text{Moment-matched Gaussian}} \tag{3.131}$$

which has an identical form to the integrand in equation 3.112, and thus is analytically tractable ($\tilde{Z}$ is the new normalisation constant – see equation 3.143). The result of equation 3.131 is another Gaussian,

$$q^{\text{EPmm}}_{\text{smooth}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(\boldsymbol{x}_t;\; \boldsymbol{\mu}^{\text{EPmm}}_{t|T},\; \Sigma^{\text{EPmm}}_{t|T}\right) \tag{3.132}$$

which means the projection step doesn't have any effect. This has an unfortunate consequence when we use the approximation, equation 3.131, in the equation for the forward message (equation 3.120),

$$\begin{aligned} \alpha_t(\boldsymbol{x}_t) \;&\propto\; \frac{\text{proj}\left\{p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t)\; q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1}, \boldsymbol{\theta})\right\}}{\beta_t(\boldsymbol{x}_t)} \;=\; \frac{\beta_t(\boldsymbol{x}_t)\; q(\boldsymbol{x}_t, \boldsymbol{y}_t \mid \alpha_{t-1}, \boldsymbol{\theta})}{\beta_t(\boldsymbol{x}_t)} \\ \Rightarrow \alpha_t(\boldsymbol{x}_t) \;&=\; q(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \alpha_{t-1}, \boldsymbol{\theta}) \end{aligned} \tag{3.133}$$

the backward message β_t cancels out. We can find an expression for the new forward message,

$$\begin{aligned} \alpha_t(\boldsymbol{x}_t) \;&=\; \mathcal{N}\left(\boldsymbol{x}_t;\; \boldsymbol{\mu}_{\alpha_t},\; \Sigma_{\alpha_t}\right) \\ \boldsymbol{\mu}_{\alpha_t} \;&=\; \boldsymbol{\mu}_{t|\alpha_{t-1}} + \Sigma_{t|\alpha_{t-1}} C^T \left(C\, \Sigma_{t|\alpha_{t-1}} C^T + \Sigma_\nu\right)^{-1} \left(\boldsymbol{y}_t - C\boldsymbol{\mu}_{t|\alpha_{t-1}}\right) \\ \Sigma_{\alpha_t} \;&=\; \Sigma_{t|\alpha_{t-1}} - \Sigma_{t|\alpha_{t-1}} C^T \left(C\, \Sigma_{t|\alpha_{t-1}} C^T + \Sigma_\nu\right)^{-1} C \Sigma_{t|\alpha_{t-1}} \end{aligned} \tag{3.134}$$

however, we are no longer approximating the message in the context of the cavity distribution as required by EP. In fact the forward message has no dependence at all on the backward message, which means that the approximation cannot be improved by running several EP steps - the posterior approximation is complete after a single forward and backward pass.

Given that, under this approximation, the forward messages are independent of the backward messages they can be interpreted as the same approximate filter distributions as in the GP-ADF based approach,

$$\alpha_t(\boldsymbol{x}_t) \;=\; q_{\text{filter}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \tag{3.135}$$

This also leads to an equality between the time update distribution and the approximate distribution from equation 3.129,

$$q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1}, \boldsymbol{\theta}) \;=\; q_{\text{time}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \tag{3.136}$$

It is hard to see from equation 3.122 what approximation to make in order to compute the backward message update. An alternative method for computing the required moments is to consider the derivatives of the 'log partition function' - the normalising constant of the term we are trying to project. For the backward message this means integrating the term inside the projection in equation 3.122 over $\boldsymbol{x}_{t-1}$ as well as over $\boldsymbol{x}_t$,

$$Z \;=\; \int \alpha_{t-1}(\boldsymbol{x}_{t-1}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t)\; d\boldsymbol{x}_t\; d\boldsymbol{x}_{t-1} \tag{3.137}$$

We can find the moments that we need for the projection step in the backward message update by

taking the derivative of the log partition function, $\log Z$, which acts as a moment generating function, w.r.t. the moments of the relevant cavity distribution — in this case α_{t-1}. To see this, consider the derivative of Z w.r.t. the mean of α_{t-1},

$$\begin{aligned}
\frac{\partial Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}} &= \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \frac{\partial \alpha_{t-1}(\boldsymbol{x}_{t-1})}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}}\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&= \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \Sigma_{\alpha_{t-1}}^{-1} \left(\boldsymbol{x}_{t-1} - \boldsymbol{\mu}_{\alpha_{t-1}}\right) \alpha_{t-1}(\boldsymbol{x}_{t-1})\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&= \Sigma_{\alpha_{t-1}}^{-1} \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int \boldsymbol{x}_{t-1}\, p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&\quad - \Sigma_{\alpha_{t-1}}^{-1} \boldsymbol{\mu}_{\alpha_{t-1}} \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&= Z\, \Sigma_{\alpha_{t-1}}^{-1} \left(\mathbb{E}\left[\boldsymbol{x}_{t-1}\right] - \boldsymbol{\mu}_{\alpha_{t-1}}\right) \qquad (3.138) \\
\Rightarrow\ \mathbb{E}\left[\boldsymbol{x}_{t-1}\right] &= \frac{1}{Z}\, \Sigma_{\alpha_{t-1}} \frac{\partial Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}} + \boldsymbol{\mu}_{\alpha_{t-1}} \\
&= \Sigma_{\alpha_{t-1}} \frac{\partial \log Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}} + \boldsymbol{\mu}_{\alpha_{t-1}} \qquad (3.139)
\end{aligned}$$

and similarly for the variance,

$$\begin{aligned}
\frac{\partial^2 Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T} &= \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \frac{\partial^2 \alpha_{t-1}(\boldsymbol{x}_{t-1})}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T}\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&= \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) \\
&\qquad \Sigma_{\alpha_{t-1}}^{-1} \left(\boldsymbol{x}_{t-1} - \boldsymbol{\mu}_{\alpha_{t-1}}\right) \left(\boldsymbol{x}_{t-1} - \boldsymbol{\mu}_{\alpha_{t-1}}\right)^T \Sigma_{\alpha_{t-1}}^{-1}\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&\qquad - \Sigma_{\alpha_{t-1}}^{-1} \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&= Z \left(\Sigma_{\alpha_{t-1}}^{-1} \left(\mathbb{V}\left[\boldsymbol{x}_{t-1}\right] + \left(\mathbb{E}[\boldsymbol{x}_{t-1}] - \boldsymbol{\mu}_{\alpha_{t-1}}\right) \left(\mathbb{E}[\boldsymbol{x}_{t-1}] - \boldsymbol{\mu}_{\alpha_{t-1}}\right)^T\right) \Sigma_{\alpha_{t-1}}^{-1} - \Sigma_{\alpha_{t-1}}^{-1}\right) \qquad (3.140) \\
\Rightarrow\ \mathbb{V}\left[\boldsymbol{x}_{t-1}\right] &= \frac{1}{Z}\, \Sigma_{\alpha_{t-1}} \frac{\partial^2 Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T} \Sigma_{\alpha_{t-1}} - \left(\mathbb{E}[\boldsymbol{x}_{t-1}] - \boldsymbol{\mu}_{\alpha_{t-1}}\right) \left(\mathbb{E}[\boldsymbol{x}_{t-1}] - \boldsymbol{\mu}_{\alpha_{t-1}}\right)^T + \Sigma_{\alpha_{t-1}} \\
&= \Sigma_{\alpha_{t-1}} \frac{\partial^2 \log Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T} \Sigma_{\alpha_{t-1}} + \Sigma_{\alpha_{t-1}} \qquad (3.141)
\end{aligned}$$

as,

$$\begin{aligned}
\frac{\partial^2 \log Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T} &= \frac{1}{Z} \frac{\partial^2 Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T} - \frac{\partial Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}} \frac{\partial Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}}^T \\
&= \frac{1}{Z} \frac{\partial^2 Z}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T} - \Sigma_{\alpha_{t-1}}^{-1} \left(\mathbb{E}[\boldsymbol{x}_{t-1}] - \boldsymbol{\mu}_{\alpha_{t-1}}\right) \left(\mathbb{E}[\boldsymbol{x}_{t-1}] - \boldsymbol{\mu}_{\alpha_{t-1}}\right)^T \Sigma_{\alpha_{t-1}}^{-1} \qquad (3.142)
\end{aligned}$$

Thus we need to be able to find the derivatives of $\log Z$. Unfortunately, equation 3.137 is intractable. However, by swapping the order of integration we can make the same moment-matching approximation as before (equation 3.129),

$$\begin{aligned}
Z &= \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\, d\boldsymbol{x}_{t-1}\, d\boldsymbol{x}_t \\
&\approx \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t) \underbrace{q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1}, \boldsymbol{\theta})}_{\text{Moment-matched Gaussian}} d\boldsymbol{x}_t \triangleq \tilde{Z} \qquad (3.143)
\end{aligned}$$

$\tilde{Z}$ is the same normalising constant as in equation 3.131 and it is tractable to compute: first we use Bayes rule,

$$\tilde{Z} \;=\; \int p(\boldsymbol{y}_t \mid \boldsymbol{x}_t,\, \boldsymbol{\theta})\, \beta_t(\boldsymbol{x}_t)\, q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1},\, \boldsymbol{\theta})\; d\boldsymbol{x}_t \;=\; p(\boldsymbol{y}_t \mid \beta_t,\, \boldsymbol{\theta}) \int p(\boldsymbol{x}_t \mid \boldsymbol{y}_t,\, \beta_t,\, \boldsymbol{\theta})\, q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1},\, \boldsymbol{\theta})\; d\boldsymbol{x}_t \tag{3.144}$$

where,

$$p(\boldsymbol{y}_t \mid \beta_t,\, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(\boldsymbol{y}_t;\; C\,\boldsymbol{\mu}_{\beta_t},\; C\,\Sigma_{\beta_t} C^T + \Sigma_\nu\right) \tag{3.145}$$

$$p(\boldsymbol{x}_t \mid \boldsymbol{y}_t,\, \beta_t,\, \boldsymbol{\theta}) \;=\; \mathcal{N}\left(\boldsymbol{y}_t;\; \boldsymbol{\mu}_{t|y\beta},\; \Sigma_{t|y\beta}\right) \tag{3.146}$$

$$\boldsymbol{\mu}_{t|y\beta} \;=\; \boldsymbol{\mu}_{\beta_t} + \Sigma_{\beta_t} C^T (C\,\Sigma_{\beta_t} C^T + \Sigma_\nu)^{-1} (\boldsymbol{y}_t - C\,\boldsymbol{\mu}_{\beta_t}) \tag{3.147}$$

$$\Sigma_{t|y\beta} \;=\; \Sigma_{\beta_t} - \Sigma_{\beta_t} C^T \left(C\,\Sigma_{\beta_t} C^T + \Sigma_\nu\right)^{-1} C\,\Sigma_{\beta_t} \tag{3.148}$$

$\tilde{Z}$ is then given by the product of $p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta})$ with the normalising constant of the product of the two Gaussians in the integral. This gives,

$$\begin{aligned}\log Z \;\approx\; \log \tilde{Z} \;=\; &-\log 2\pi \;-\; \frac{1}{2}\log\left|C\,\Sigma_{\beta_t} C^T + \Sigma_\nu\right| \;-\; \frac{1}{2}(\boldsymbol{y} - C\,\boldsymbol{\mu}_{\beta_t})^T \left(C\,\Sigma_{\beta_t} C^T + \Sigma_\nu\right)^{-1} (\boldsymbol{y} - C\,\boldsymbol{\mu}_{\beta_t}) \\ &-\; \frac{1}{2}\log\left|\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right| \;-\; \frac{1}{2}(\boldsymbol{\mu}_{t|y\beta} - \boldsymbol{\mu}_{t|\alpha_{t-1}})^T \left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} (\boldsymbol{\mu}_{t|y\beta} - \boldsymbol{\mu}_{t|\alpha_{t-1}})\end{aligned} \tag{3.149}$$

Before we can proceed with taking the derivatives of $\log \tilde{Z}$, we repeat the observation from Deisenroth and Mohamed (2012),

> When computing EP updates using the derivatives [equations 3.139 and 3.141], it is crucial to explicitly account for the implicit linearization assumption in the derivatives – otherwise, the EP updates are inconsistent.

This is in reference to the relationship between the α_{t-1} message and the matched Gaussian $q_\alpha(\boldsymbol{x}_t \mid \alpha_{t-1}, \boldsymbol{\theta})$. Specifically,

$$W \;\triangleq\; \frac{\partial \boldsymbol{\mu}_{t|\alpha_{t-1}}}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}}, \qquad \frac{\partial \Sigma_{t|\alpha_{t-1}}}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}} \;=\; 0 \tag{3.150}$$

We can now take the required derivatives of the approximate log partition function,

$$\frac{\partial \log \tilde{Z}}{\partial \boldsymbol{\mu}_{\alpha_{t-1}}} \;=\; W^T \left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} (\boldsymbol{\mu}_{t|y\beta} - \boldsymbol{\mu}_{t|\alpha_{t-1}}) \tag{3.151}$$

$$\frac{\partial^2 \log \tilde{Z}}{\partial \boldsymbol{\mu}_{\alpha_{t-1}} \partial \boldsymbol{\mu}_{\alpha_{t-1}}^T} \;=\; -W^T \left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} W \tag{3.152}$$

Substituting these derivatives into equation 3.139 gives,

$$\boldsymbol{\mu}_{t-1|T}^{\mathrm{EPmm}} \;=\; \boldsymbol{\mu}_{\alpha_{t-1}} \;-\; \Sigma_{\alpha_{t-1}} W^T \left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} (\boldsymbol{\mu}_{t|\alpha_{t-1}} - \boldsymbol{\mu}_{t|y\beta}) \tag{3.153}$$

and into equation 3.141

$$\Sigma_{t-1|T}^{\mathrm{EPmm}} \;=\; \Sigma_{\alpha_{t-1}} \;-\; \Sigma_{\alpha_{t-1}} W^T \left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} W\,\Sigma_{\alpha_{t-1}} \tag{3.154}$$

These equations look very similar to the smoothing equations for the GP-ADF based algorithm (equation 3.96). We can explore this further: if we use the equalities from equations 3.135 and 3.136,

$$\boldsymbol{\mu}_{\alpha_{t-1}} = \boldsymbol{\mu}_{t-1|t-1}, \qquad \Sigma_{\alpha_{t-1}} = \Sigma_{t-1|t-1} \tag{3.155}$$

$$\boldsymbol{\mu}_{t|\alpha_{t-1}} = \boldsymbol{\mu}_{t|t-1}, \qquad \Sigma_{t|\alpha_{t-1}} = \Sigma_{t|t-1} \tag{3.156}$$

Further, if $\boldsymbol{x}$ is Gaussian, $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, then when taking the derivative of an expectation over $\boldsymbol{x}$ w.r.t. $\boldsymbol{\mu}$,

$$\frac{\partial \mathbb{E}_x[f(\boldsymbol{x})]}{\partial \boldsymbol{\mu}} = \Sigma^{-1}\, \mathbb{C}[\boldsymbol{x},\ f(\boldsymbol{x})] \tag{3.157}$$

$$\Rightarrow W = \Sigma_{\alpha_{t-1}}^{-1}\, C_{t-1,t|t-1} \tag{3.158}$$

Finally, note that from the EP approximation to the smoothed marginal in equation 3.131,

$$\begin{aligned} \Sigma_{t|T}^{\text{EPmm}} &= \left(\Sigma_{t|y\beta}^{-1} + \Sigma_{t|\alpha_{t-1}}^{-1}\right)^{-1} \\ \Rightarrow \left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} &= \Sigma_{t|t-1}^{-1}\left(\Sigma_{t|t-1} - \Sigma_{t|T}^{\text{EPmm}}\right)\Sigma_{t|t-1}^{-1} \\ \boldsymbol{\mu}_{t|T}^{\text{EPmm}} &= \Sigma_{t|T}^{\text{EPmm}}\left(\Sigma_{t|y\beta}^{-1}\,\boldsymbol{\mu}_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}^{-1}\,\boldsymbol{\mu}_{t|\alpha_{t-1}}\right) \end{aligned} \tag{3.159}$$

This gives,

$$\begin{aligned} \Sigma_{t-1|T}^{\text{EPmm}} &= \Sigma_{\alpha_{t-1}} - \Sigma_{\alpha_{t-1}} W^T \left(\Sigma_{y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} W\, \Sigma_{\alpha_{t-1}} \\ &= \Sigma_{t-1|t-1} + C_{t-1,t|t-1}\,\Sigma_{t|t-1}^{-1}\left(\Sigma_{t|T}^{\text{EPmm}} - \Sigma_{t|t-1}\right)\Sigma_{t|t-1}^{-1} C_{t-1,t|t-1}^T \\ &= \Sigma_{t-1|t-1} + J_{t-1}\left(\Sigma_{t|T}^{\text{EPmm}} - \Sigma_{t|t-1}\right) J_{t-1}^T \end{aligned} \tag{3.160}$$

$$\begin{aligned} \boldsymbol{\mu}_{t-1|T}^{\text{EPmm}} &= \boldsymbol{\mu}_{\alpha_{t-1}} - \Sigma_{\alpha_{t-1}} W^T \left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} (\boldsymbol{\mu}_{t|\alpha_{t-1}} - \boldsymbol{\mu}_{t|y\beta}) \\ &= \boldsymbol{\mu}_{t-1|t-1} + C_{t-1,t|t-1}\Sigma_{t|t-1}^{-1}\,\Sigma_{t|t-1}\left(\Sigma_{t|y\beta} + \Sigma_{t|\alpha_{t-1}}\right)^{-1} (\Sigma_{t|y\beta}\,\Sigma_{t|y\beta}^{-1}\,\boldsymbol{\mu}_{t|y\beta} - \Sigma_{t|t-1}\,\Sigma_{t|t-1}^{-1}\,\boldsymbol{\mu}_{t|\alpha_{t-1}}) \\ &= \boldsymbol{\mu}_{t-1|t-1} + J_{t-1}\Big[\Sigma_{t|t-1}\left(\Sigma_{t|y\beta} + \Sigma_{t|t-1}\right)^{-1}\Sigma_{t|y\beta}\,\Sigma_{t|y\beta}^{-1}\,\boldsymbol{\mu}_{t|y\beta} \\ &\qquad + \left(\Sigma_{t|t-1} - \Sigma_{t|t-1}\left(\Sigma_{t|y\beta} + \Sigma_{t|t-1}\right)^{-1}\Sigma_{t|t-1}\right)\Sigma_{t|t-1}^{-1}\boldsymbol{\mu}_{t|t-1} - \boldsymbol{\mu}_{t|t-1})\Big] \\ &= \boldsymbol{\mu}_{t-1|t-1} + J_{t-1}\left[\Sigma_{t|T}^{\text{EPmm}}\left(\Sigma_{t|y\beta}^{-1}\,\boldsymbol{\mu}_{t|y\beta} + \Sigma_{t|t-1}^{-1}\,\boldsymbol{\mu}_{t|t-1}\right) - \boldsymbol{\mu}_{t|t-1})\right] \\ &= \boldsymbol{\mu}_{t-1|t-1} + J_{t-1}\left(\boldsymbol{\mu}_{t|T}^{\text{EPmm}} - \boldsymbol{\mu}_{t|t-1}\right) \end{aligned} \tag{3.161}$$

We showed earlier that the forward sweep of the 'EP' algorithm of Deisenroth and Mohamed (2012) led to an identical filter distribution as the GP-ADF based filter of Turner et al. (2010). We have now shown that the smoothing updates are also identical to the the GP-ADF based smoothing equations. Thus we have shown that the two seemingly different algorithms actually give rise to an identical approximation to the latent state posterior. This means that use of the algorithm in Deisenroth and Mohamed (2012) does not mitigate the problems with the linearisation described earlier. However, this is not a property of EP but rather the moment-matching approximation that was used to solve the EP updates. If we used a different method for updating the messages this problem could be

avoided.

EP using sampling

Rather than use moment-matching to find approximate moments of the smoothing distributions required for EP, we could use sampling approaches instead. These are likely to give a much more accurate approximation to the required moments, although they may be computationally slower. That said, in the E step of the EM algorithm we do not require any derivatives and so sampling is much more computationally feasible here than it was in the direct approach. In this section we present three different sampling schemes for finding the moments: simple Monte Carlo (designated by 'mc' in superscripts), importance sampling ('is'), and Markov Chain Monte Carlo ('mcmc').

Recall from equations 3.125 and 3.126 that, to update the EP messages, we need to find the first two moments of the smoothing marginals, which we can write,

$$Z_t \, q^{\text{EP}}_{\text{smooth}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; \underbrace{p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta})}_{\text{Gaussian; eq. 3.145}} \; \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta_t, \boldsymbol{\theta})}_{\text{Gaussian; eq. 3.146}} \int \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\mathcal{GP}\text{ prediction}} \; \underbrace{\alpha_{t-1}(\boldsymbol{x}_{t-1})}_{\text{Gaussian message}} \; d\boldsymbol{x}_{t-1} \tag{3.162}$$

$$Z_{t-1} \, q^{\text{EP}}_{\text{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \;=\; p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \, \alpha_{t-1}(\boldsymbol{x}_{t-1}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) \, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta_t, \boldsymbol{\theta}) \; d\boldsymbol{x}_t \tag{3.163}$$

Once we have these moments we can run EP by plugging them straight into equations 3.127 and 3.128. We can't find the moments of the distributions in equations 3.162 and 3.163 analytically due to the GP prediction term and so we use sampling. We start with a simple Monte Carlo scheme.

Simple Monte Carlo

To compute the moments of the first smoothing distribution, equation 3.162, note that for a given $\boldsymbol{x}_{t-1}$ every term is either a constant or (exactly) a Gaussian in $\boldsymbol{x}_t$, excluding α_{t-1}, which is a Gaussian in $\boldsymbol{x}_{t-1}$. As moments of products of Gaussians can be computed analytically, a straightforward sampling approach is to sample multiple values of $\boldsymbol{x}_{t-1}$ from $\alpha_{t-1}(\boldsymbol{x}_{t-1})$ and then average together the resulting moments. That is, first we sample from α_{t-1},

$$\boldsymbol{x}^i_{t-1} \;\sim\; \alpha_{t-1}(\boldsymbol{x}_{t-1}) \;=\; \mathcal{N}\left(\boldsymbol{x}_{t-1};\; \boldsymbol{\mu}_{\alpha_{t-1}},\; \Sigma_{\alpha_{t-1}}\right) \qquad \text{for } i = 1, \ldots, N \tag{3.164}$$

then we define,

$$p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta) \, p(\boldsymbol{x}_t \mid \boldsymbol{x}^i_{t-1}, \boldsymbol{\theta}) \;=\; z^i \, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}^i_{t-1}, \beta_t, \boldsymbol{\theta}) \;=\; z^i \, \mathcal{N}\left(\boldsymbol{x}_t;\; \boldsymbol{\mu}^i_t,\; \Sigma^i_t\right) \tag{3.165}$$

with,

$$\boldsymbol{\mu}^i_t \;=\; \left(\Sigma^{-1}_{t|y\beta} \;+\; \mathfrak{s}(\boldsymbol{x}^i_{t-1})^{-1}\right)^{-1} \left(\Sigma^{-1}_{t|y\beta} \, \boldsymbol{\mu}_{t|y\beta} \;+\; \mathfrak{s}(\boldsymbol{x}^i_{t-1})^{-1} \, \mathfrak{m}(\boldsymbol{x}^i_{t-1})\right) \tag{3.166}$$

$$\Sigma^i_t \;=\; \left(\Sigma^{-1}_{t|y\beta} \;+\; \mathfrak{s}(\boldsymbol{x}^i_{t-1})^{-1}\right)^{-1} \tag{3.167}$$

and z^i as the normalising constant (integrating equation 3.165 over $\boldsymbol{x}_t$ for a given $\boldsymbol{x}_{t-1}^i$ rather than integrating over both $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t-1}$ as for Z),

$$\begin{aligned} z^i &= p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \int p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta)\, p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\ d\boldsymbol{x}_t \\ &= p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta})\, (2\pi)^{-D/2} \left|\Sigma_{t|y\beta} + \mathfrak{s}(\boldsymbol{x}_{t-1}^i)\right|^{-1/2} \\ &\qquad \exp\left(-\frac{1}{2}\left(\boldsymbol{\mu}_{t|y\beta} - \mathfrak{m}(\boldsymbol{x}_{t-1}^i)\right)^T \left(\Sigma_{t|y\beta} + \mathfrak{s}(\boldsymbol{x}_{t-1}^i)\right)^{-1} \left(\boldsymbol{\mu}_{t|y\beta} - \mathfrak{m}(\boldsymbol{x}_{t-1}^i)\right)\right) \end{aligned} \tag{3.168}$$

where $p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta})$, $\boldsymbol{\mu}_{t|y\beta}$, and $\Sigma_{t|y\beta}$ are defined in equations 3.145-3.148. We can interpret z^i as the value of $p(\boldsymbol{x}_{t-1}^i \mid \boldsymbol{y}_t, \beta, \boldsymbol{\theta})$, which is a measure of how well $\boldsymbol{x}_{t-1}^i$ 'agrees' with the observed data points from times t to T. In other words, we sample a value of $\boldsymbol{x}_{t-1}$ based on the information from time 1 to $t-1$, and then weight these samples using the information from the remaining observations. With this setup we can compute a sample-based approximation to the first moment of the smoothing marginal in equation 3.162 using,

$$\begin{aligned} \boldsymbol{\mu}_{t|T}^{\mathrm{EP}} &= p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \int \boldsymbol{x}_t\ p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\ d\boldsymbol{x}_{t-1}\ d\boldsymbol{x}_t \\ &= p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \int \alpha_{t-1}(\boldsymbol{x}_{t-1}) \int \boldsymbol{x}_t\ p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta)\, p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\ d\boldsymbol{x}_t\ d\boldsymbol{x}_{t-1} \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \int \boldsymbol{x}_t\ z^i\, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\ d\boldsymbol{x}_t \\ &= \frac{1}{N} \sum_{i=1}^{N} z^i\, \boldsymbol{\mu}_t^i \ \triangleq\ \boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}} \end{aligned} \tag{3.169}$$

For the variance,

$$\begin{aligned} \Sigma_{t|T}^{\mathrm{EP}} &= p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \int \left(\boldsymbol{x}_t - \boldsymbol{\mu}_{t|T}^{\mathrm{EP}}\right)\left(\boldsymbol{x}_t - \boldsymbol{\mu}_{t|T}^{\mathrm{EP}}\right)^T p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta_t, \boldsymbol{\theta}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, \alpha_{t-1}(\boldsymbol{x}_{t-1})\ d\boldsymbol{x}_{t-1}\ d\boldsymbol{x}_t \\ &\approx \frac{1}{N} \sum_{i=1}^{N} z^i \int \left(\boldsymbol{x}_t - \boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}}\right)\left(\boldsymbol{x}_t - \boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}}\right)^T p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\ d\boldsymbol{x}_t \\ &= \frac{1}{N} \sum_{i=1}^{N} z^i \int \boldsymbol{x}_t\, \boldsymbol{x}_t^T p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\ d\boldsymbol{x}_t \ -\ \frac{1}{N} \sum_{i=1}^{N} z^i \int \boldsymbol{x}_t\, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\ d\boldsymbol{x}_t\, (\boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}})^T \\ &\qquad - \boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}} \frac{1}{N} \sum_{i=1}^{N} z^i \int \boldsymbol{x}_t^T\, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\ d\boldsymbol{x}_t \ +\ \boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}} (\boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}})^T \frac{1}{N} \sum_{i=1}^{N} z^i \int p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\ d\boldsymbol{x}_t \\ &= \frac{1}{N} \sum_{i=1}^{N} z^i \left(\Sigma_t^i \ +\ \boldsymbol{\mu}_t^i\, (\boldsymbol{\mu}_t^i)^T\right) \ -\ \boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}} (\boldsymbol{\mu}_{t|T}^{\mathrm{EPmc}})^T \ \triangleq\ \Sigma_{t|T}^{\mathrm{EPmc}} \end{aligned} \tag{3.170}$$

Similarly for the moments at $t-1$

$$\begin{aligned} \boldsymbol{\mu}_{t-1|T}^{\mathrm{EP}} &= p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \int \boldsymbol{x}_{t-1}\, \alpha_{t-1}(\boldsymbol{x}_{t-1}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta)\ d\boldsymbol{x}_t\ d\boldsymbol{x}_{t-1} \\ &\approx \frac{1}{N} \sum_{i=1}^{N} z^i\, \boldsymbol{x}_{t-1}^i \ \triangleq\ \boldsymbol{\mu}_{t-1|T}^{\mathrm{EPmc}} \end{aligned} \tag{3.171}$$

$$\Sigma^{\mathrm{EP}}_{t-1|T} = p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \int \left(\boldsymbol{x}_{t-1} - \boldsymbol{\mu}^{\mathrm{EP}}_{t-1|T}\right) \left(\boldsymbol{x}_{t-1} - \boldsymbol{\mu}^{\mathrm{EP}}_{t-1|T}\right)^T \alpha_{t-1}(\boldsymbol{x}_{t-1}) \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\, p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta)\; d\boldsymbol{x}_t\; d\boldsymbol{x}_{t-1}$$
$$\approx \frac{1}{N} \sum_{i=1}^{N} z^i \left(\boldsymbol{x}^i_{t-1} - \boldsymbol{\mu}^{\mathrm{EPmc}}_{t-1|T}\right) \left(\boldsymbol{x}^i_{t-1} - \boldsymbol{\mu}^{\mathrm{EPmc}}_{t-1|T}\right)^T \triangleq \Sigma^{\mathrm{EPmc}}_{t-1|T} \tag{3.172}$$

Using these moments we can compute the message updates as before: by plugging the computed moments straight into equations 3.127 and 3.128. These Monte Carlo moments will be good approximations so long as the sampling distribution $\alpha_{t-1}(\boldsymbol{x}_{t-1})$ is a good approximation to the true smoothing marginal on $\boldsymbol{x}_{t-1}$, $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, which is the region where z will take on large values. In other words, if $\alpha_{t-1}(\boldsymbol{x}_{t-1})$ is far from $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$ then the majority of $\{z^i\}$ will have negligible values and the moment approximations will be dominated by only a handful of samples, which will lead to a poor approximation.

Importance sampling

Recall that the approximate single-slice smoothing marginals are given by the product of the messages, $q^{\mathrm{EP}}_{\mathrm{smooth}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) \propto \alpha_{t-1}(\boldsymbol{x}_{t-1})\, \beta_{t-1}(\boldsymbol{x}_{t-1})$. Therefore, rather than sampling $\boldsymbol{x}_{t-1}$ directly from α_{t-1}, a better sample approximation to the smoothing moments is given by using the current values of both α_{t-1} and β_{t-1} to compute the sampling distribution. In this way we draw samples from the current estimate of the true smoothing marginal, rather than just from the estimated filter marginal. As this new sampling distribution does not naturally appear in equations 3.162 and 3.163 we must use a corrective weight to ensure we still compute an unbiased estimate of the true moments. This is an application of importance sampling. Of course, this approach is only different from the previous simple Monte Carlo method if we already have a setting for both the forward α and backward β messages. This usually means that we must have already completed at least one forward and backward sweep.

Redefining the samples $\boldsymbol{x}^i_{t-1}$ (and 'recomputing' z^i, $\boldsymbol{\mu}^i_t$, and Σ^i_t equivalently),

$$\boldsymbol{x}^i_{t-1} \sim \mathcal{N}\left(\boldsymbol{x}_{t-1};\; \boldsymbol{\mu}_{\alpha\beta_{t-1}},\; \Sigma_{\alpha\beta_{t-1}}\right) \propto \alpha_{t-1}(\boldsymbol{x}_{t-1})\, \beta_{t-1}(\boldsymbol{x}_{t-1}) \qquad \text{for } i = 1, \ldots, N \tag{3.173}$$
$$\boldsymbol{\mu}_{\alpha\beta_{t-1}} = \Sigma_{\beta_{t-1}} \left(\Sigma_{\alpha_{t-1}} + \Sigma_{\beta_{t-1}}\right)^{-1} \boldsymbol{\mu}_{\alpha_{t-1}} + \Sigma_{\alpha_{t-1}} \left(\Sigma_{\alpha_{t-1}} + \Sigma_{\beta_{t-1}}\right)^{-1} \boldsymbol{\mu}_{\beta_{t-1}} \tag{3.174}$$
$$\Sigma_{\alpha\beta_{t-1}} = \Sigma_{\beta_{t-1}} \left(\Sigma_{\alpha_{t-1}} + \Sigma_{\beta_{t-1}}\right)^{-1} \Sigma_{\alpha_{t-1}} \tag{3.175}$$

we have that,

$$\boldsymbol{\mu}^{\mathrm{EP}}_{t|T} = p(\boldsymbol{y}_t \mid \beta_t, \boldsymbol{\theta}) \int \frac{\overbrace{\alpha_{t-1}(\boldsymbol{x}_{t-1})\, \beta_{t-1}(\boldsymbol{x}_{t-1})}^{\text{Sampling distribution}}}{\underbrace{\beta_{t-1}(\boldsymbol{x}_{t-1})}_{\text{Correction factor}}} \int \boldsymbol{x}_t\; p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \beta)\, p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\; d\boldsymbol{x}_t\; d\boldsymbol{x}_{t-1}$$
$$\approx \frac{1}{N} \sum_{i=1}^{N} \frac{z^i}{\beta_{t-1}(\boldsymbol{x}^i_{t-1})}\, \boldsymbol{\mu}^i_t \triangleq \boldsymbol{\mu}^{\mathrm{EPis}}_{t|T} \tag{3.176}$$

Notice that we have effectively multiplied by $\frac{\beta_{t-1}(\boldsymbol{x}_{t-1})}{\beta_{t-1}(\boldsymbol{x}_{t-1})}$. Similarly, for the other moments,

$$\Sigma_{t|T}^{\text{EPis}} \triangleq \frac{1}{N}\sum_{i=1}^{N}\frac{z^i}{\beta_{t-1}(\boldsymbol{x}_{t-1}^i)}\left(\Sigma_t^i + \boldsymbol{\mu}_t^i(\boldsymbol{\mu}_t^i)^T\right) - \boldsymbol{\mu}_{t|T}^{\text{EPis}}(\boldsymbol{\mu}_{t|T}^{\text{EPis}})^T \tag{3.177}$$

$$\boldsymbol{\mu}_{t-1|T}^{\text{EPis}} \triangleq \frac{1}{N}\sum_{i=1}^{N}\frac{z^i}{\beta_{t-1}(\boldsymbol{x}_{t-1}^i)}\boldsymbol{x}_{t-1}^i \tag{3.178}$$

$$\Sigma_{t-1|T}^{\text{EPis}} \triangleq \frac{1}{N}\sum_{i=1}^{N}\frac{z^i}{\beta_{t-1}(\boldsymbol{x}_{t-1}^i)}\left(\boldsymbol{x}_{t-1}^i - \boldsymbol{\mu}_{t-1|T}^{\text{EPis}}\right)\left(\boldsymbol{x}_{t-1}^i - \boldsymbol{\mu}_{t-1|T}^{\text{EPis}}\right)^T \tag{3.179}$$

Note that as both the messages are Gaussian the moments of the importance sampling distribution, $\boldsymbol{\mu}_{\alpha\beta_{t-1}}$, $\Sigma_{\alpha\beta_{t-1}}$, are given by the standard equations for the moments of a product of Gaussians. This importance sampling scheme requires a previous forward and backward sweep to set the values of the messages; in the first sweep we must still use the simple Monte Carlo scheme outline previously. If this sweep leads to messages which poorly approximate the smoothing marginal then using importance sampling may not provide a significant benefit. To combat this we could use a MCMC sampling scheme, which will actively seek areas of high probability.

Markov Chain Monte Carlo

In MCMC we do not have a fixed sampling distribution, rather we draw the samples one-by-one, using the last sample drawn to determine where to sample from next — hence forming a Markov chain. At each step a new sample is first proposed and then either accepted or rejected based on its relative probability under the distribution we are sampling from compared to the probability of the previously accepted point. In this way the Markov chain seeks out areas of high probability, which can be very far away from the initial distribution. Thus we can use MCMC to draw samples which cover the support of a distribution, even if we have a very poor initial estimate of where the relevant probability mass is. To define a MCMC sampler we need to choose how to draw the first sample and also what the proposal distribution is, that is, the distribution from which potential new samples are drawn, and which is parametrised by the last sample. A sensible choice for the first sample is the mean of the current approximation to the smoothing marginal,

$$\boldsymbol{x}_{t-1}^1 = \boldsymbol{\mu}_{\alpha\beta_{t-1}} \tag{3.180}$$

We can further use the current smoothing approximation by setting the proposal density to be a Gaussian centred on the current sample and with a variance equal to a scaled version of $\Sigma_{\alpha\beta_{t-1}}$,

$$p(\boldsymbol{x}_{t-1}^{i+1} \mid \boldsymbol{x}_{t-1}^i) = \mathcal{N}\left(\boldsymbol{x}_{t-1}^{i+1};\ \boldsymbol{x}_{t-1}^i,\ c\,\Sigma_{\alpha\beta_{t-1}}\right) \tag{3.181}$$

this can help the MCMC algorithm by using the current estimate of the covariance structure in the proposal density. The acceptance probability is given by,

$$p(\text{accept} \mid \boldsymbol{x}_{t-1}^{i+1}, \boldsymbol{x}_{t-1}^i) = \frac{p(\boldsymbol{x}_{t-1}^{i+1} \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{y}_t, \boldsymbol{\theta})}{p(\boldsymbol{x}_{t-1}^i \mid \boldsymbol{x}_{t-1}^{i-1}, \boldsymbol{y}_t, \boldsymbol{\theta})} = \frac{\alpha_{t-1}(\boldsymbol{x}_{t-1}^{i+1})\,z^{i+1}}{\alpha_{t-1}(\boldsymbol{x}_{t-1}^i)\,z^i} \tag{3.182}$$

This leads to an MCMC approximation to the moments,

$$\boldsymbol{\mu}_{t|T}^{\text{EPmcmc}} \triangleq \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\mu}_t^i \tag{3.183}$$

$$\Sigma_{t|T}^{\text{EPmcmc}} \triangleq \frac{1}{N}\sum_{i=1}^{N} \left(\Sigma_t^i + \boldsymbol{\mu}_t^i (\boldsymbol{\mu}_t^i)^T\right) - \boldsymbol{\mu}_{t|T}^{\text{EPmcmc}} (\boldsymbol{\mu}_{t|T}^{\text{EPmcmc}})^T \tag{3.184}$$

$$\boldsymbol{\mu}_{t-1|T}^{\text{EPmcmc}} \triangleq \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_{t-1}^i \tag{3.185}$$

$$\Sigma_{t-1|T}^{\text{EPmcmc}} \triangleq \frac{1}{N}\sum_{i=1}^{N} \left(\boldsymbol{x}_{t-1}^i - \boldsymbol{\mu}_{t-1|T}^{\text{EPmcmc}}\right)\left(\boldsymbol{x}_{t-1}^i - \boldsymbol{\mu}_{t-1|T}^{\text{EPmcmc}}\right)^T \tag{3.186}$$

Both the simple Monte Carlo scheme and the importance sampling scheme allow all the samples to be drawn at once. In MCMC however the samples are drawn sequentially. As we must make a GP prediction for each sample drawn the MCMC scheme is likely to be a lot slower than the other two schemes as it cannot make use of parallel computation as the other schemes can. On the other hand, it should be able to find a good approximation to the moments even if the current approximation to the marginals is poor.

Algorithm 3 Expectation Propagation with Sampling E Step

- Iterate EP until convergence or for a fixed number of steps
  - Forwards
    - For $t = 1:T$:
      - Compute moments of current EP approximation to smoothing posterior at time t using either simple MC (equations 3.169 & 3.170), IS (equations 3.176 & 3.177), or MCMC (equations 3.183 & 3.184)
        $$q_{\text{smooth}}^{\text{EP}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_{t|T},\ \Sigma_{t|T}\right)$$
      - Divide out the message α_{t-1} to find moments of new message α_t
        $$\alpha_t \propto \frac{q_{\text{smooth}}^{\text{EP}}(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})}{\alpha_{t-1}}$$
  - Backwards
    - For $t = T:2$, compute β_{t-1} messages:
      - Compute moments of current EP approximation to smoothing posterior at time $t-1$ using either simple MC (equations 3.171 & 3.172), IS (equations 3.178 & 3.179), or MCMC (equations 3.185 & 3.186)
        $$q_{\text{smooth}}^{\text{EP}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{x}_{t-1};\ \boldsymbol{\mu}_{t-1|T},\ \Sigma_{t-1|T}\right)$$
      - Divide out the message β_t to find moments of new message β_{t-1}
        $$\beta_{t-1} \propto \frac{q_{\text{smooth}}^{\text{EP}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})}{\beta_t}$$
      - On last backward pass also compute moments of $q_{\text{smooth}}^{\text{EP}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$ and the cross-covariances $\mathbb{C}[\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta}]$ for the M step

3.6.3 Comparison of E Steps

In the previous section we presented five different methods for computing an approximation to the smoothing posterior distribution on the latent states. In this section we compare the five methods

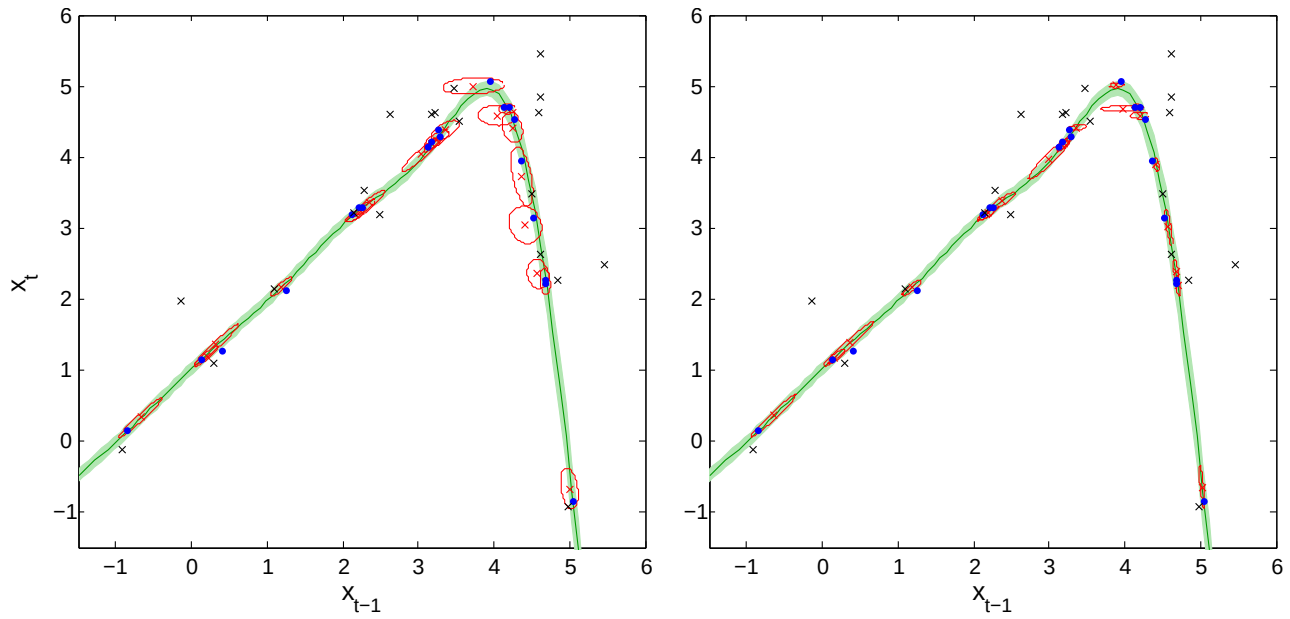


Figure 3.18: Example comparison of the approximate-analytic EP E step methods (left plot, both analytic methods give the same results as predicted earlier) with a sample based EP E step (right plot). The green line shows the true transition function, the shaded region shows two standard deviations of process noise. The black crosses are the observed transitions, i.e. from $\boldsymbol{y}_{t-1}$ to $\boldsymbol{y}_t$, and the blue dots are the latent transitions. The red circles show the Gaussian latent state posteriors: the red cross shows the location of the mean and the red circle shows the one standard deviation contour. The blue, true latent state, dots can be seen to have much higher probability under the red latent state posteriors in the right hand plot; this agrees with the results in table 3.1.

on a number of data sets. First we look at a one dimensional kink function. Figure 3.18 shows the true transition function and the latent state posterior distributions found by the analytic methods and the Monte Carlo sampling method. The posterior distributions from the sampling method are visibly tighter than the posteriors inferred by the approximate-analytic methods. We can find a quantitative comparison by computing the log probability of the true latent points (blue dots in figure 3.18) under each of the inferred latent posteriors. The results for doing so for all methods on this dataset are shown in table 3.1,

Table 3.1: Log probability per data point of the true latent points under the latent state posteriors inferred by each method. ADS is the assumed density smoothing E step of Turner et al. (2010), A-EP is the analytic EP algorithm of Deisenroth and Mohamed (2012). We showed that these two methods are identical and thus we only list one result for them. The remaining three methods are the sample-based EP approaches, using a simple Monte Carlo scheme, importance sampling, and MCMC respectively.

ADS/A-EP	MC-EP	IS-EP	MCMC-EP
0.6765	0.9644	0.9790	0.9651

,

The sampling methods significantly outperform the approximate-analytic methods. All three sampling schemes have a similar performance, with the importance sampling scheme just out-performing the other two methods. The margin there is small though, compared to the increase in performance relative to the analytic methods. However, the sampling methods are much less stable than the approximate-analytic methods and have a number of failure modes. This is highlighted by figure 3.19.

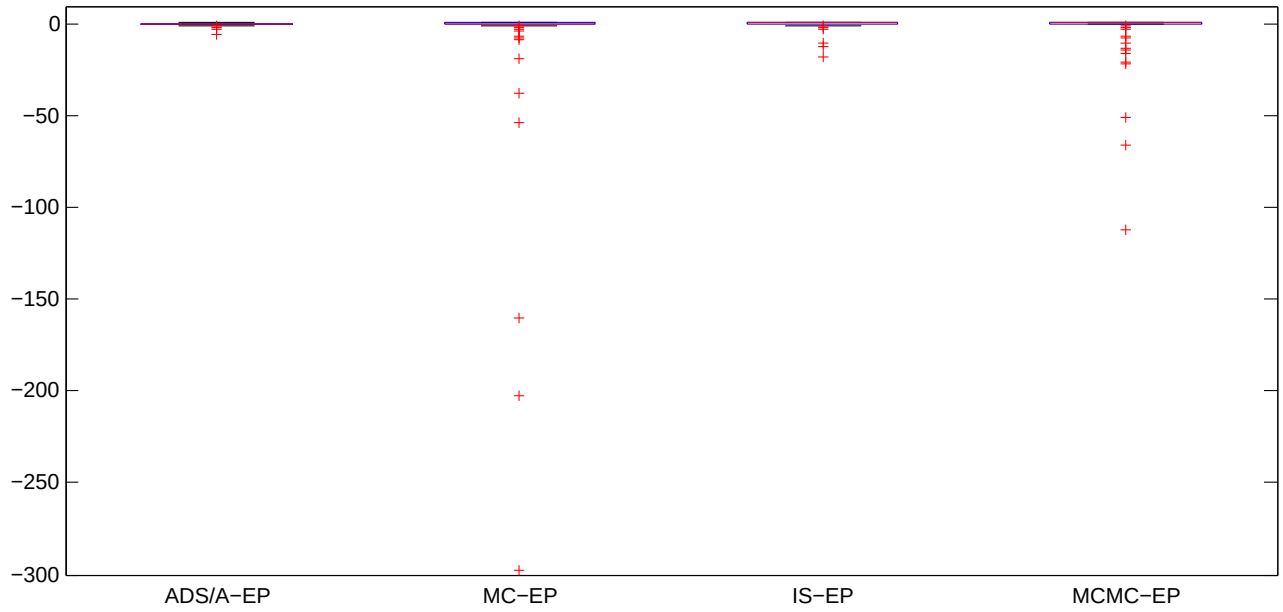


Figure 3.19: Boxplots of the performance of the analytic and sampling E step methods over 100 test trajectories. The y-axis is the log likelihood per data point of the true latent states under the inferred posterior. In over 75% of the test trajectories the sampling methods outperform the analytic methods, an example of this was shown in table 3.1. However, the figure clearly shows that there are a significant number of cases where the sampling methods fail and cause the latent states have extremely low log likelihood.

The first failure mode, which particularly affects the basic MC approach, is where very few (if any) of the samples drawn have a high weight, that is, a significant value of z^i in equation 3.169. In the sampling literature this is often referred to as a low effective sample size. In the application we are considering here, it typically occurs when a data point is corrupted by a large amount of observation noise and thus is far away from where the latent state lies. For example, consider the observation (black cross) near the point $(0, 2)$ in figure 3.18. The corresponding latent state is the blue dot near $(0, 1)$. However, the observed value $\boldsymbol{y}_t$ suggests that $\boldsymbol{x}_{t-1}$ should have a value closer to 1 (cast the value of $\boldsymbol{y}_t$ onto the transition function and then down onto the x-axis). For the sake of this argument let us assume that at the end of the forward sweep the filter posterior on $\boldsymbol{x}_{t-1}$ has a mean of 0 and a standard deviation of 0.1. In the MC smoothing step we will draw samples from this filter posterior and then compute $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_t, \beta_t, \boldsymbol{\theta})$. But, as we just stated, this will only have high values around 1, which is ten standard deviations away from the mean of the sampling distribution (how close a sample must get to the projected observation in order to have a 'high' value depends on how large the observation noise is). In this case, sufficient samples achieve a significant weight but there are many examples where they do not, as the results in figure 3.19 show. Figure 3.20 quantifies this intuition and shows that the points which are least well predicted by the inferred latent state posteriors are typically those with smaller effective sample sizes. We can reduce the effect that this has somewhat by testing the effective sample size and only proceeding with the EP message update if it exceeds a certain threshold. However, skipping update steps in this way can slow down EP's convergence, although this is preferable to accepting a completely erroneous update.

We introduced the importance sampling method to counter this possibility and figure 3.19 shows that using importance sampling does indeed improve the performance of the sampling algorithm: there are fewer outliers and those present are of a smaller magnitude. It does not remove the problem

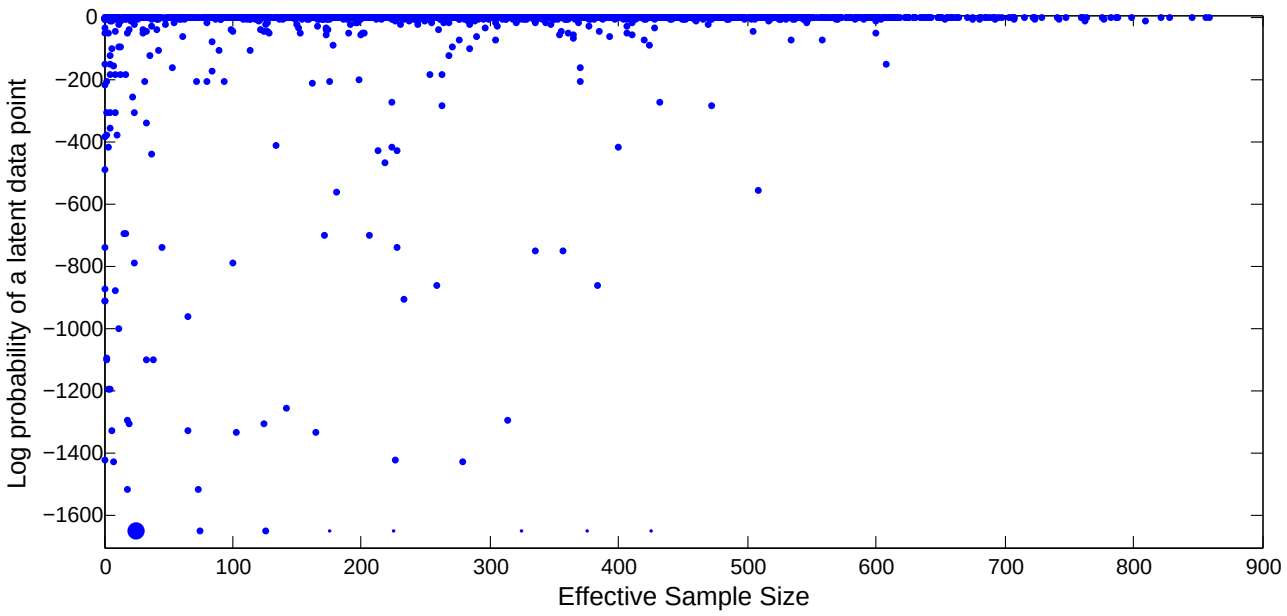


Figure 3.20: This plot shows how the E step performance (likelihood of true latent point under inferred posterior) corresponds to the effective sample size of the MC-EP algorithm. There are thirty data points off the bottom of the plot, which are not directly shown for the sake of clarity. Instead, the line of blue circles at the bottom of the plot indicate, by their size, the number of data points not shown, summarised over equally spaced x-axis intervals of width 50.

completely however: during the first forward and backward pass the importance sampling algorithm the is the same as the simple Monte Carlo algorithm and thus can get off to a poor start. Furthermore, the importance sampling EP step will only improve over the basic MC step if the current estimate of the latent state posterior is a good approximation to the true posterior. There is an inevitable 'chicken and egg' problem here, which can lead to poor performance in the importance sampling algorithm. The third sampling EP algorithm discussed above used MCMC to try and find the areas of high probability. This algorithm is not beholden to the accuracy of the current estimate of latent state posterior: the chain can potentially move some distance from its initialisation point. However, setting the proposal distribution is fraught with difficulty. The method we presented used a scaled version of the current latent state posterior estimate, centred on the previous sample. Unfortunately, choosing the scale value is hard and it often requires tuning to specific problems and data sets. If the proposal distribution is too wide then the MCMC algorithm accepts very few points, on the other hand if the proposal is too narrow then the MCMC chain does very little exploration. Both of these cases usually lead to a poor sample estimate of the latent state posterior: in particular the variance is usually dramatically underestimated due to the poor sample diversity. One could perhaps design a method to adapt the scale of the proposal distribution over multiple EP iterations, which may help to alleviate this problem.

We now move to a broader test of the E step algorithms. We simulate a number of different nonlinear dynamical systems, which allows us to record both the true latent states as well as their 'observed' versions. We generate a random D-dimensional nonlinear system by drawing ten D-dimensional input and output points from a zero-mean Gaussian with variance $2I$ (I is the $D \times D$ identity matrix). The transition function was then set to be a GP with these input/output points as the training matrix. Strictly, we use D GPs, each mapping from the full input space to a single output dimension. Each of these GPs is independent of the others and has a separate set of hyperparameters. We also select the

variance of the process and observation noise in a random manner. The system parameters, for the eth output dimension, were set as follows,

$$\begin{aligned}
\text{process noise variance, } \sigma^2_{\epsilon,e} &= \exp(-2 - 2\mathcal{U}) \\
\text{observation noise variance, } \sigma^2_{\nu,e} &= \exp(-1 - 2\mathcal{U}) \\
\text{squared kernel lengthscales, } l^2_e &= D \\
\text{kernel signal variance, } \sigma^2_{f,e} &= \exp(-2) \\
\text{kernel noise variance, } \sigma^2_{n,e} &= \exp(-4 - 2\mathcal{U})
\end{aligned}$$

where each use of $\mathcal{U}$ indicates a separate draw from a uniform distribution on the interval $[0, 1]$. These values were chosen to generate nonlinear systems with 'interesting' dynamics, i.e. those with strong nonlinearities and non-trivial trajectories. Figure 3.21 shows some sample one dimensional transition functions generated in this manner and an observed trajectory generated from each system. As we are using a GP transition function, the transitions contain uncertainty over-and-above the process noise — the green shaded region in figure 3.21 shows two standard deviations either side of the transition function mean. This extra uncertainty makes the E step more difficult.

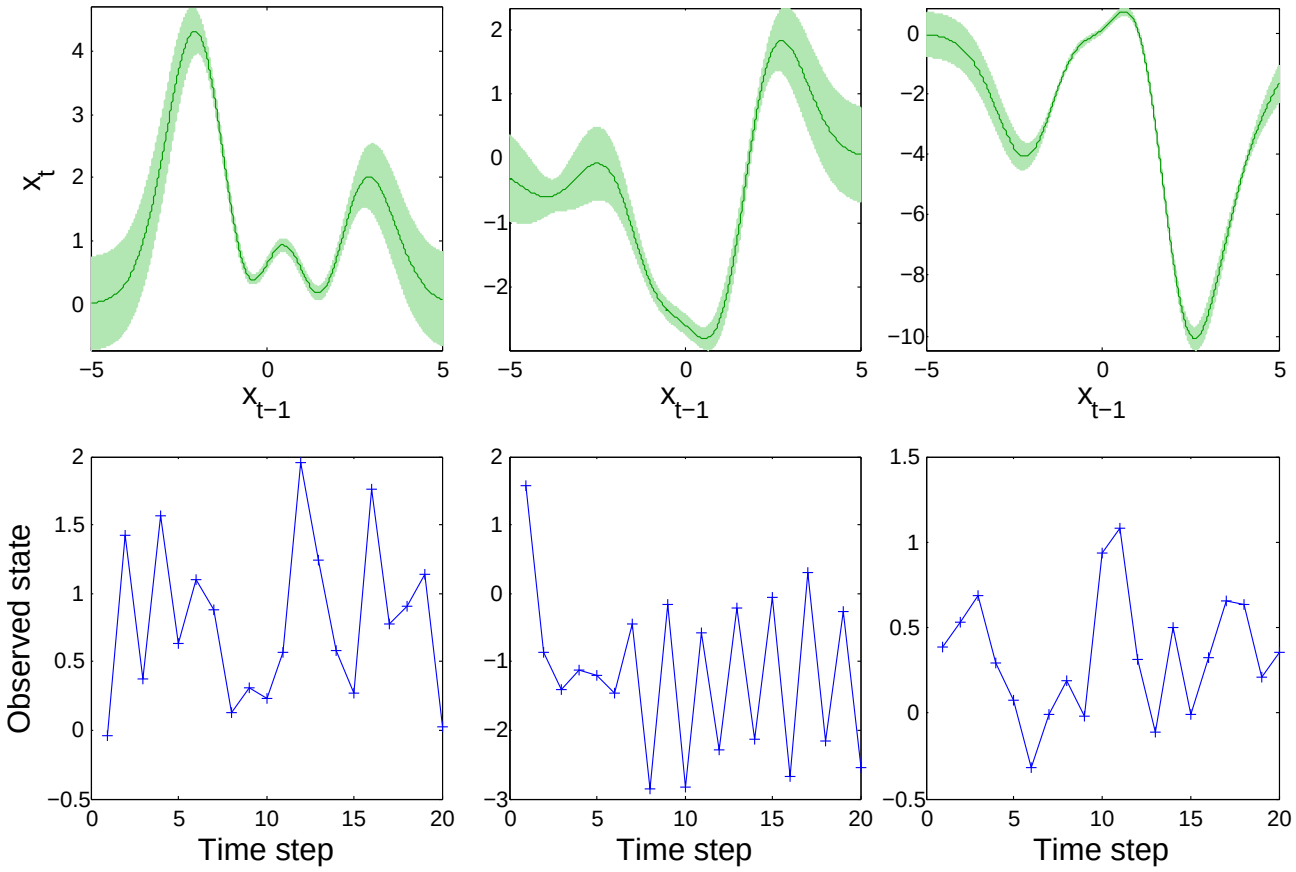


Figure 3.21: Example randomly generated GP transition functions (top) and some resulting, noisy trajectories (bottom). The GP transition functions were constructed according to the steps laid out above.

Each transition function, along with the corresponding observed trajectory was passed to each of the E step algorithms to perform inference. The performance of sampling methods is often dependent on the dimension of the state. To test this we generated systems with dimension ranging from one to

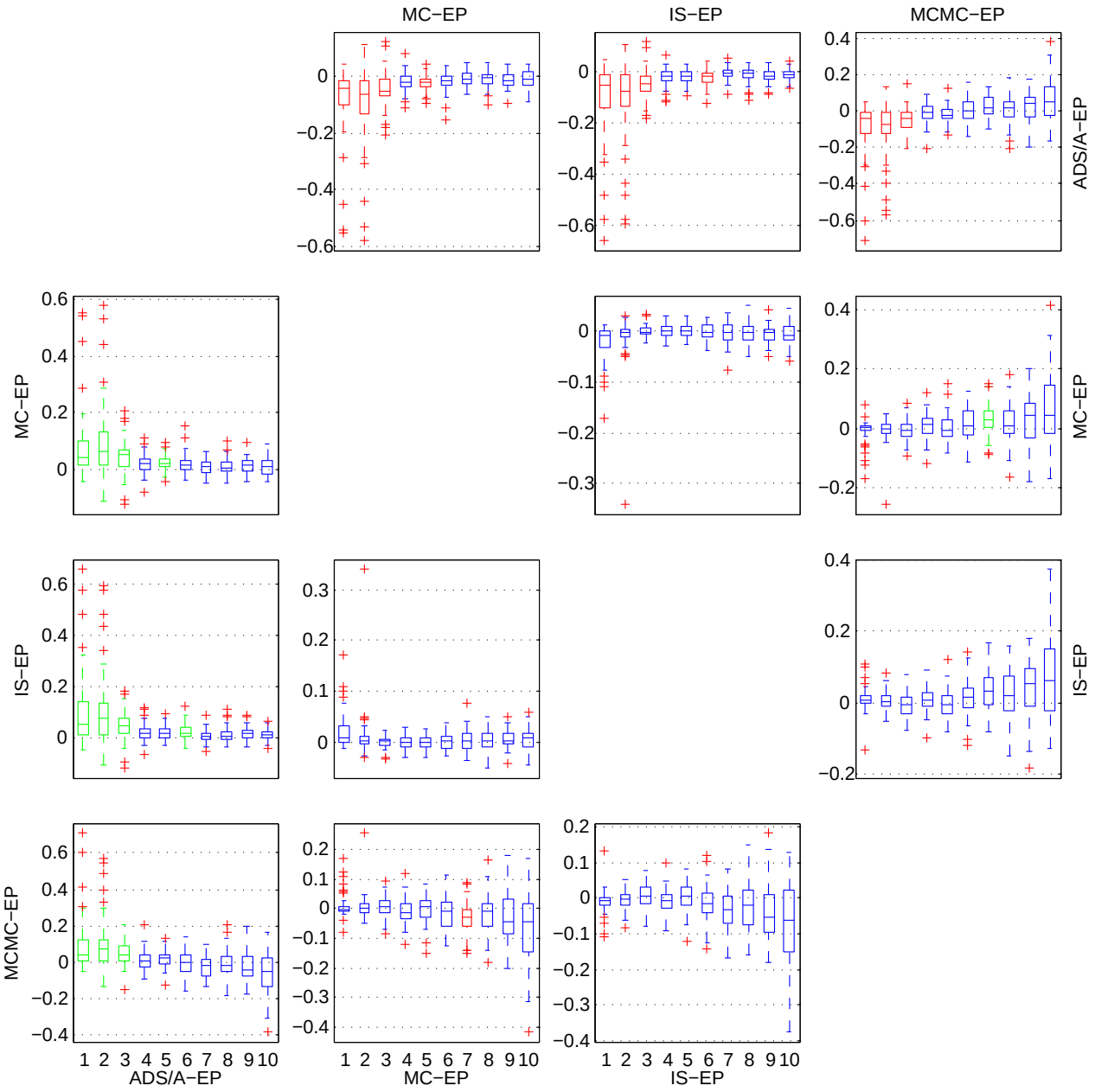


Figure 3.22: Box plots comparing approximate-analytic E-step algorithms on randomly generated nonlinear systems. The x-axis of each plot is the dimension of the state, varying from a 1D system to a 10D system. The y-axis is the difference in the log likelihood per data point of the true latent states between two different algorithms. The difference is found by subtracting the log likelihood per data point of the columns from the rows; thus a positive number on the y-axis indicates the algorithm for the row is outperforming the algorithm for the column. Boxes are coloured green/red when 75% of the data lies above/below zero.

ten. Further, for each dimension we generated ten different random nonlinear systems on which to test the algorithms. The performance of each algorithm was tested by computing the log likelihood of the actual latent trajectory under each of the latent state posteriors. Figure 3.22 shows the results of this experiment.

As one might expect the sampling methods outperform the moment-matching approximation, particularly for lower dimensional systems. The MCMC sampling version does not in general perform as well as the other two sampling methods, indicating that the Markov chain might be getting stuck.

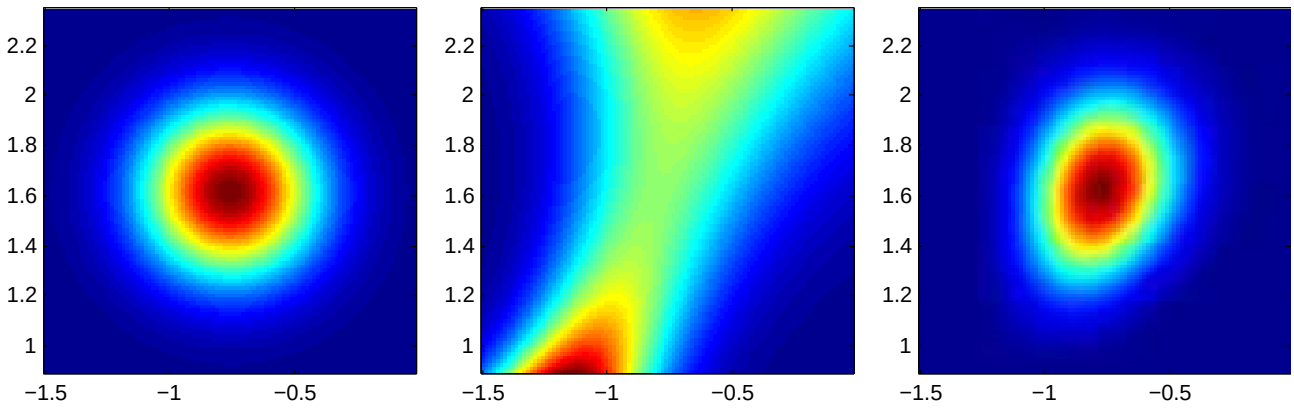


Figure 3.23: The first backwards smoothing step using sampling inside EP for an example 2D system. The left plot shows the filter distribution on $\boldsymbol{x}_{T-1}$; the middle plot shows the distribution on $\boldsymbol{x}_{T-1}$ given by conditioning on $\boldsymbol{y}_T$. The third plot shows the result of the smoothing step: the combination of the previous two plots. The nonlinear GP causes the second and third plots to be non-Gaussian.

However, there is a fundamental problem with using sampling within EP, which is illustrated in figure 3.23. This figure shows the first backwards smoothing update for a two dimensional system. The first plot shows the current Gaussian distribution on $\boldsymbol{x}_{T-1}$ given by $\alpha_{T-1}(\boldsymbol{x}_{T-1})$. The second plot shows the probability distribution on $\boldsymbol{x}_{T-1}$ conditioning on the final observation $\boldsymbol{y}_T$. It is clear to see that this is a very non-Gaussian distribution, which is a result of the nonlinear GP transition function. The third plot shows the combination of the first two plots to give the smoothing posterior $p(\boldsymbol{x}_{T-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, which again is non-Gaussian. In EP we now project the final distribution to a Gaussian by using moment matching. In the example of figure 3.23, the variance of the first distribution, $\Sigma_{\alpha_{T-1}}$ and the empirical variance of the samples in the smoothing distribution are,

$$\Sigma_{\alpha_{T-1}} = \begin{bmatrix} 0.0605 & -0.0005 \\ -0.0005 & 0.0580 \end{bmatrix}, \qquad \tilde{\Sigma}_{T-1|T} \begin{bmatrix} 0.0384 & 0.0098 \\ 0.0098 & 0.0622 \end{bmatrix} \tag{3.187}$$

The marginal variance of the first state dimension has decreased due to including the information from the final observation $\boldsymbol{y}_T$. However, the marginal variance of the second dimension has increased, due to the non-Gaussianity of the distribution. This is incompatible with Gaussian distributions—new information must always decrease the variance. This means that when we calculate the next backward message $\beta_{T-1}(\boldsymbol{x}_{T-1})$ there is no real information to send (the message will have a nonsensical covariance matrix, not positive-definite and with possibly negative variances). Because of the Markov nature of the state space model this breaks the smoothing chain: information from the final observation $\boldsymbol{y}_T$ will not have any major effect on the posterior distributions for previous time steps, t less than $T-1$. This limits the improvement we can gain by using sampling within EP.

It is useful to consider what the approximation the analytic EP method is making looks like in the context of this example; this is shown in figure 3.24. The top left plot shows the true result of passing the α_{T-1} message through the GP transition function. It is striking how close to Gaussian this is (the top middle plot shows the corresponding moment-matched Gaussian) compared to passing $p(\boldsymbol{x}_T \mid \boldsymbol{y}_T, \beta_T, \boldsymbol{\theta})$ back through the transition function (middle plot of figure 3.23). There are two reasons for this: the first is that the filter distribution on $\boldsymbol{x}_{T-1}$ given by the α_{T-1} message is much tighter than the distribution on $\boldsymbol{x}_T$ given by the observation. This can be seen if we compare the

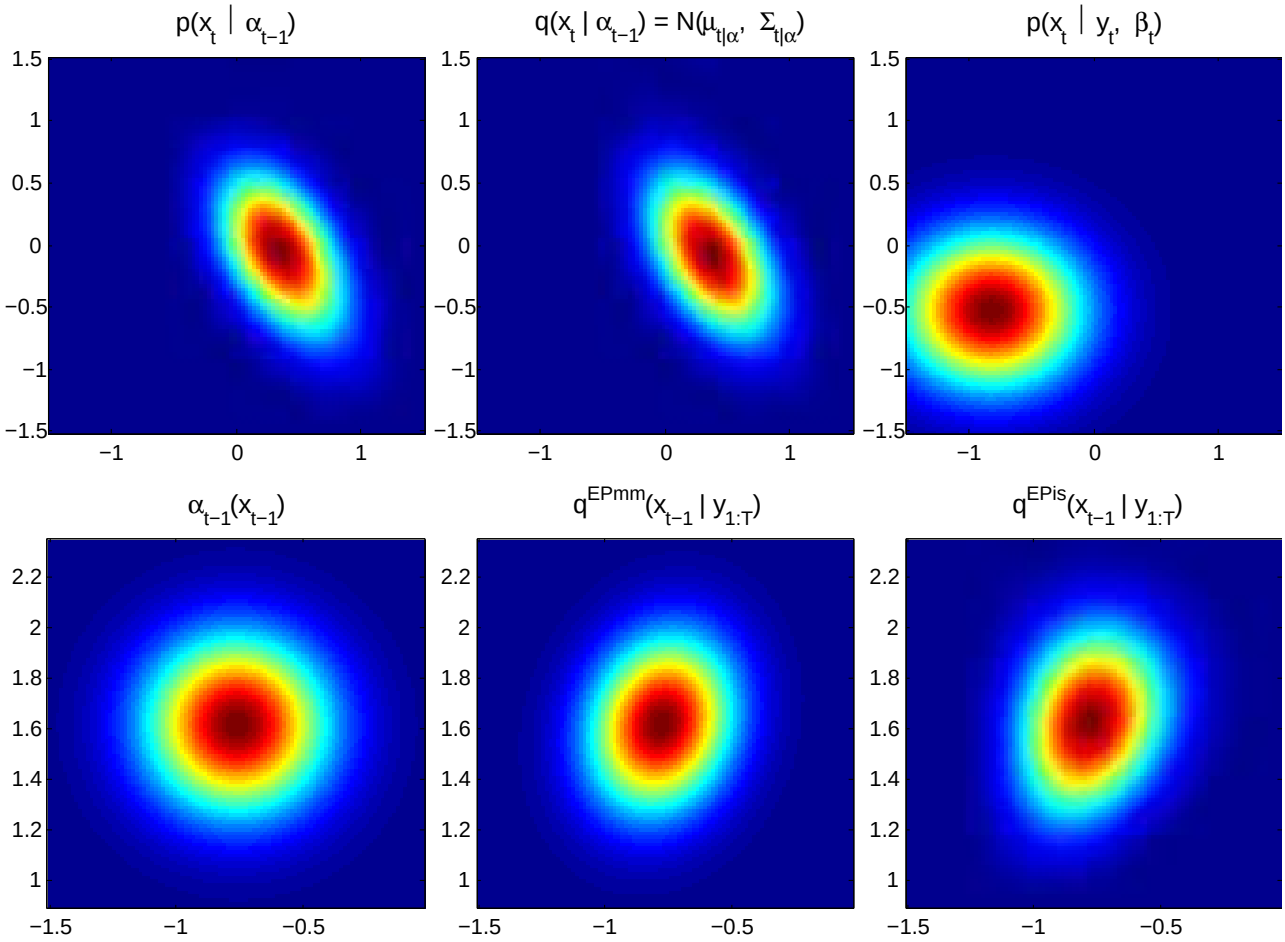


Figure 3.24: The first backwards smoothing step using EP with moment matching for an example 2D system. The top left plot shows the result of the integral $\int p(\boldsymbol{x}_T \mid \boldsymbol{x}_{T-1}, \boldsymbol{\theta})\, \alpha_{T-1}(\boldsymbol{x}_{T-1})\, d\boldsymbol{x}_{T-1}$ using sampling, the top middle plot shows the moment-matched Gaussian approximation (equation 3.129). The top right plot shows the Gaussian distribution on $\boldsymbol{x}_T$ resulting from the final observation $\boldsymbol{y}_T$. The bottom left plot shows the α message on $\boldsymbol{x}_{T-1}$, the middle bottom shows the resulting approximate smoothing posterior on $\boldsymbol{x}_{T-1}$, $q(\boldsymbol{x}_{T-1} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$ obtained by combining the top middle and right plots and passing them back through the linearised transition model. For comparison the bottom right plot shows the posterior obtained with sampling (before the EP moment-matching).

corresponding variances,

$$\Sigma_{\alpha_{T-1}} = \begin{bmatrix} 0.0605 & -0.0005 \\ -0.0005 & 0.0580 \end{bmatrix}, \qquad \Sigma_{T|T} \begin{bmatrix} 0.2516 & 0 \\ 0 & 0.1830 \end{bmatrix} \tag{3.188}$$

Note that the diagonal nature of $\Sigma_{T|T}$ comes from the linear observation model and independent observation noise. The smaller variance on $\Sigma_{\alpha_{T-1}}$ is to be expected: this is exactly what filtering is supposed to do — combine many observations to obtain a more certain distribution on the unobserved state than is possible from using a single observation. The tighter distribution means that the transition is more accurately captured by a linearisation, which would map an input Gaussian onto an exact output Gaussian, than would be the case for a larger distribution.

The second reason is that a GP is a distribution over single-valued functions: the standard mathematical functions, which have a single output for every input. This means that going forward through a function is much simpler than going backwards, where a single output value can correspond to multiple input values. The conclusion from this analysis is that using sampling to invert the GP transition

function can bring little gain compared to the moment-matching/linearisation approaches, due to the inevitable moment-matching, which is part of the EP algorithm.

3.6.4 M Step

In the previous section we looked at five different ways of finding an approximation, q_{smooth}, to the smoothing posterior on the latent states, $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$. Doing this is only a stepping stone to our real goal, however, which is to train the model parameters. This learning is done in the maximisation (or minimisation) step of the EM algorithm, which we look at in this section. We use the approximate latent state posterior to find an (approximate) upper bound on the negative log marginal likelihood. We can then differentiate this bound with respect to the parameters and hence use gradient descent optimisation. In fact, we show that some of the parameters, including the key pseudo-targets, can be solved for directly, without the need of an optimisation algorithm.

The approximate upper bound on the negative log likelihood can be written,

$$\begin{aligned} Q &= -\mathbb{E}_{q_{\text{smooth}}(\boldsymbol{x}_{1:T}|y_{1:T},\theta)}\left[\log p(\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:T} \mid \boldsymbol{\theta})\right] \\ &= \underbrace{-\sum_{t=2}^{T} \mathbb{E}_{q(x_{t-1:t}|y_{1:T})}\left[\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\right]}_{\text{Transition term}} \underbrace{-\sum_{t=1}^{T} \mathbb{E}_{q(x_t|y_{1:T})}\left[\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\right]}_{\text{Observation term}} \underbrace{-\mathbb{E}_{q(x_1|y_{1:T})}\left[\log p(\boldsymbol{x}_1 \mid \boldsymbol{\theta})\right]}_{\text{First state}} \end{aligned} \tag{3.189}$$

where we are taking the expectations w.r.t the approximate latent state smoothing posteriors found in the E step. However, the GP in the transition function means that we cannot compute the expectation in the transition term exactly:

$$p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{x}_t; \mathfrak{m}(\boldsymbol{x}_{t-1}), \mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon\right) = \prod_{e=1}^{D} \mathcal{N}\left(x_{t,e}; \mathfrak{m}_e(\boldsymbol{x}_{t-1}), \mathfrak{s}_e(\boldsymbol{x}_{t-1}) + \sigma^2_{\epsilon,e}\right) \tag{3.190}$$

where we have used the fact that the GP transition functions for each output dimension are independent (recall that we are using D separate GPs to model the transition to each of the D output dimensions). Thus,

$$-\mathbb{E}\left[\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\right] = \frac{1}{2}\sum_{e=1}^{D} \mathbb{E}\left[\frac{(x_{t,e} - \mathfrak{m}_e(\boldsymbol{x}_{t-1}))^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}) + \sigma^2_{\epsilon,e}}\right] + \frac{1}{2}\sum_{e=1}^{D} \mathbb{E}\left[\log(\mathfrak{s}_e(\boldsymbol{x}_{t-1}) + \sigma^2_{\epsilon,e})\right] + \frac{D}{2}\log 2\pi \tag{3.191}$$

with the expectations taken over $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$, according to their approximate smoothing distribution. The difficulty in equation 3.191 is that the GP predictive variance (highlighted in red), which appears as an inverse, depends on $\boldsymbol{x}_{t-1}$. This makes the first expectation in equation 3.191 intractable to compute. To solve this expectation Turner et al. (2010) make a simple approximation: they replace the expectation of the quotient with the ratio of the expectations. The second expectation is also intractable, however, as we are already working with a bound on the log likelihood, we can use Jensen's inequality to swap the expectation and the logarithm without changing the setup dramatically. Using

these steps the approximate transition term in the negative log likelihood is,

$$Q^{\text{trans}} \triangleq \frac{1}{2}\frac{(\mu_{t|T,e} - M_{t,e})^2 + \Sigma_{t|T,e} + \mathbb{V}_{q(x_{t-1}|y_{1:T})}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1})\right] - 2\mathbb{C}_{q(x_{t,e},x_{t-1}|y_{1:T})}[x_{t,e},\, \boldsymbol{x}_{t-1}]\, C_{t,e}}{\mathbb{E}_{q(x_{t-1}|y_{1:T})}\left[\mathfrak{s}_e(\boldsymbol{x}_{t-1})\right] + \sigma^2_{\epsilon,e}} + \frac{1}{2}\sum_{e=1}^{D}\log\left(\mathbb{E}_{q(x_{t-1}|y_{1:T})}\left[\mathfrak{s}_e(\boldsymbol{x}_{t-1})\right] + \sigma^2_{\epsilon,e}\right) + \frac{D}{2}\log 2\pi \tag{3.192}$$

where,

$$\begin{aligned} M_{t,e} &= \mathbb{E}_{q(x_{t,e},x_{t-1}|y_{1:T})}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1})\right] \\ C_{t,e} &= \mathbb{V}_{q(x_{t-1}|y_{1:T})}[\boldsymbol{x}_{t-1}]^{-1}\,\mathbb{C}_{q(x_{t-1}|y_{1:T})}\left[\boldsymbol{x}_{t-1},\ \mathfrak{m}_e(\boldsymbol{x}_{t-1})\right] \end{aligned} \tag{3.193}$$

are the mean and input-output covariance (divided by the input variance) of a GP with a Gaussian test input, as defined in equations 1.28 and 1.30. Likewise, the 'variance of the mean', $\mathbb{V}_{x_{t-1}}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1})\right]$, and the 'mean of the variance', $\mathbb{E}_{x_{t-1}}\left[\mathfrak{s}_e(\boldsymbol{x}_{t-1})\right]$ are also computable and form the two parts of the variance in equation 1.29. Thus under these approximations the bound on the log likelihood can be computed and differentiated for use in gradient-based optimisation

The parameter $\boldsymbol{\beta}$ (the inverse covariance matrix times the pseudo-targets) appears linearly in the transition model term of the likelihood. It can therefore be solved for directly as a function of the other parameters,

$$\begin{aligned} &\frac{\partial Q}{\partial \boldsymbol{\beta}_e} = 0 \\ \Rightarrow\ &\frac{\partial}{\partial \boldsymbol{\beta}_e}\sum_{t=2}^{T}\frac{1}{2}\sum_{e=1}^{D}\frac{(\mu_{t|T,e} - M_{t,e})^2 + \Sigma_{t|T,e} + \mathbb{V}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1})\right] - 2\mathbb{C}[x_{t,e},\, \boldsymbol{x}_{t-1}]\,\boldsymbol{C}_{t,e}}{\mathbb{E}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{s}_e(\boldsymbol{x}_{t-1})\right] + \sigma^2_{\epsilon,e}} = 0 \\ \Rightarrow\ &\sum_{t=2}^{T}\frac{\partial}{\partial \boldsymbol{\beta}_e}\frac{(\mu_{t|T,e} - M_{t,e})^2 + \Sigma_{t|T,e} + \mathbb{V}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1})\right] - 2\mathbb{C}[x_{t,e},\, \boldsymbol{x}_{t-1}]\,\boldsymbol{C}_{t,e}}{\mathbb{E}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{s}_e(\boldsymbol{x}_{t-1})\right] + \sigma^2_{\epsilon,e}} = 0 \\ \Rightarrow\ &\sum_{t=2}^{T}\frac{1}{\bar{s}(\boldsymbol{x}_{t-1})}\left(2\left(M_{t,e} - \mu_{t|T,e}\right)\frac{\partial M_{t,e}}{\partial \boldsymbol{\beta}_e} + \frac{\partial \mathbb{V}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1},\boldsymbol{\theta})\right]}{\partial \boldsymbol{\beta}_e} - 2\,\mathbb{C}[x_{t,e},\, \boldsymbol{x}_{t-1}]\frac{\partial \boldsymbol{C}_{t,e}}{\partial \boldsymbol{\beta}_e}\right) = 0 \end{aligned} \tag{3.194}$$

$M_{t,e}$ and $\boldsymbol{C}_{t,e}$ are linear functions of $\boldsymbol{\beta}_e$ and so their derivatives w.r.t. $\boldsymbol{\beta}_e$, $\frac{\partial M_{t,e}}{\partial \boldsymbol{\beta}_e}$ and $\frac{\partial \boldsymbol{C}_{t,e}}{\partial \boldsymbol{\beta}_e}$, are not functions of $\boldsymbol{\beta}_e$. $\mathbb{V}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1},\boldsymbol{\theta})\right]$ is a quadratic function of $\boldsymbol{\beta}_e$, hence its derivative, $\frac{\partial \mathbb{V}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1})\right]}{\partial \boldsymbol{\beta}_e}$ is a linear function of $\boldsymbol{\beta}_e$. $\bar{s}(\boldsymbol{x}_{t-1}) = \mathbb{E}_{\boldsymbol{x}_{t-1}}[\mathfrak{s}_e(\boldsymbol{x}_{t-1})] + \sigma^2_{\epsilon,e}$ and is not a function of $\boldsymbol{\beta}_e$. If the eth transition GP has a covariance function k_e and a linear mean function $\boldsymbol{a}_e^T\,\boldsymbol{x}_{t-1} + b_e$,

$$\frac{\partial \mathbb{V}_{\boldsymbol{x}_{t-1}}\left[\mathfrak{m}_e(\boldsymbol{x}_{t-1})\right]}{\partial \boldsymbol{\beta}_e} = 2\mathbb{V}\left[\boldsymbol{k}_e(\tilde{X}, \boldsymbol{x}_{t-1})\right]^T\boldsymbol{\beta}_e + 2\mathbb{C}\left[\boldsymbol{k}_e(\tilde{X}, \boldsymbol{x}_{t-1}),\, \boldsymbol{x}_{t-1}\right]\boldsymbol{a}_e \tag{3.195}$$

and

$$M_{t,e} = \mathbb{E}_{q(x_{t-1}|y_{1:T})}[\boldsymbol{k}_e(\tilde{X}, \boldsymbol{x}_{t-1})]^T\boldsymbol{\beta}_e \tag{3.196}$$

Substituting equation 3.195 into equation 3.194 and writing $\boldsymbol{k}_e(\tilde{X}, \boldsymbol{x}_{t-1})$ as $\boldsymbol{k}_e^*$,

$$\begin{aligned} 0 &= \sum_{t=2}^{T} \bar{s}(\boldsymbol{x}_{t-1})^{-1} \left(2 \left(\mathbb{E}[\boldsymbol{k}_e^*]^T \boldsymbol{\beta}_e - \mu_{t|T,e} \right) \frac{\partial M_{t,e}}{\partial \boldsymbol{\beta}_e} + 2\mathbb{V}\left[\boldsymbol{k}_e^*\right]^T \boldsymbol{\beta}_e + 2\mathbb{C}\left[\boldsymbol{k}_e^*, \boldsymbol{x}_{t-1}\right] \boldsymbol{a}_e - 2\,\mathbb{C}[x_{t,e}, \boldsymbol{x}_{t-1}] \frac{\partial \boldsymbol{C}_{t,e}}{\partial \boldsymbol{\beta}_i} \right) \\ &= 2 \left[\sum_{t=2}^{T} \bar{s}(\boldsymbol{x}_{t-1})^{-1} \left(\mathbb{E}[\boldsymbol{k}_e^*]\, \mathbb{E}[\boldsymbol{k}_e^*]^T + \mathbb{V}\left[\boldsymbol{k}_e^*\right] \right) \right] \boldsymbol{\beta}_e \\ &\qquad + 2 \sum_{t=2}^{T} \bar{s}(\boldsymbol{x}_{t-1})^{-1} \left(-\mu_{t|T,e} \frac{\partial M_{t,e}}{\partial \boldsymbol{\beta}_e} + \mathbb{C}\left[\boldsymbol{k}_e^*, \boldsymbol{x}_{t-1}\right] \boldsymbol{a}_e - \mathbb{C}[x_{t,e}, \boldsymbol{x}_{t-1}] \frac{\partial \boldsymbol{C}_{t,e}}{\partial \boldsymbol{\beta}_e} \right) \end{aligned} \tag{3.197}$$

$$\boldsymbol{\beta}_e = \left[\sum_{t=2}^{T} \bar{s}(\boldsymbol{x}_{t-1})^{-1} \mathbb{E}\left[\boldsymbol{k}_e^* \left(\boldsymbol{k}_e^*\right)^T \right] \right]^{-1} \sum_{t=2}^{T} \bar{s}(\boldsymbol{x}_{t-1})^{-1} \left(-\mu_{t|T,e} \frac{\partial M_{t,e}}{\partial \boldsymbol{\beta}_e} + \mathbb{C}\left[\boldsymbol{k}_e^*, \boldsymbol{x}_{t-1}\right] \boldsymbol{a}_e - \mathbb{C}[x_{t,e}, \boldsymbol{x}_{t-1}] \frac{\partial \boldsymbol{C}_{t,e}}{\partial \boldsymbol{\beta}_e} \right) \tag{3.198}$$

The observation term in equation 3.189 can be computed exactly,

$$\begin{aligned} \mathbb{E}_{x_t}\left[\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\right] &= -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_\nu| - \frac{1}{2}\mathbb{E}_{x_t}\left[(\boldsymbol{y}_t - \boldsymbol{x}_t)^T \Sigma_\nu^{-1} (\boldsymbol{y}_t - \boldsymbol{x}_t)\right] \\ &= -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_\nu| - \frac{1}{2}(\boldsymbol{y}_t - \boldsymbol{\mu}_{t|T})^T \Sigma_\nu^{-1} (\boldsymbol{y}_t - \boldsymbol{\mu}_{t|T}) - \frac{1}{2}\mathrm{trace}\left(\Sigma_\nu^{-1}\, \Sigma_{t|T}\right) \end{aligned} \tag{3.199}$$

We can solve for the observation noise variance parameter (chosen to be the log of the standard deviation),

$$\begin{aligned} & \frac{\partial Q}{\partial \log \sigma_{\nu,e}} = 0 \\ \Rightarrow\ & \frac{\partial}{\partial \log \sigma_{\nu,e}} \left\{ \frac{1}{2} \sum_{e=1}^{D} \frac{(y_{t,e} - \mu_{t|T,e})^2 + \sigma_{t|T,e}^2}{\sigma_{\nu,e}^2} + \sum_{e=1}^{D} \log \sigma_{\nu,e} \right\} = 0 \\ \Rightarrow\ & \frac{1}{2}\left((y_{t,e} - \mu_{t|T,e})^2 + \sigma_{t|T,e}^2 \right) \frac{\partial \sigma_{\nu,e}^{-2}}{\partial \log \sigma_{\nu,e}} + 1 = 0 \\ \Rightarrow\ & -\frac{(y_{t,e} - \mu_{t|T,e})^2 + \sigma_{t|T,e}^2}{\sigma_{\nu,e}^2} + 1 = 0 \\ \Rightarrow\ & \sigma_{\nu,e}^2 = (y_{t,e} - \mu_{t|T,e})^2 + \sigma_{t|T,e}^2 \\ \Rightarrow\ & \log \sigma_{\nu,e} = \frac{1}{2}\log\left((y_{t,e} - \mu_{t|T,e})^2 + \sigma_{t|T,e}^2 \right) \end{aligned} \tag{3.200}$$

Solving for the pseudo-targets and the observation noise variance greatly reduces the number of parameters for which we must optimise. We can compute derivatives w.r.t. the remaining parameters, which requires no more than the application of simple calculus, and thus optimise them via gradient descent.

3.6.5 Analysis

We analyse the timing of the various E step algorithms as well as the M step on systems varying from one dimension to ten dimensions. A single observed trajectory of twenty time steps was used as training data, and EP was set to only use one iteration. The sample based EP algorithms were set to use ten thousand samples. Figure 3.25 shows the results of the timing experiment. As expected the analytic E

step algorithms are considerably faster than the sampling based approaches. The MCMC-EP variant is excruciatingly slow compared to all other parts of the algorithm, due its sequential nature. The M step takes longer than the E step (excluding MCMC-EP) although the EP algorithm timings are only shown for a single EP iteration. The M step shows a quadratic dependence on dimension as we would expect from the D lots of D-dimensional GP predictions at its heart.

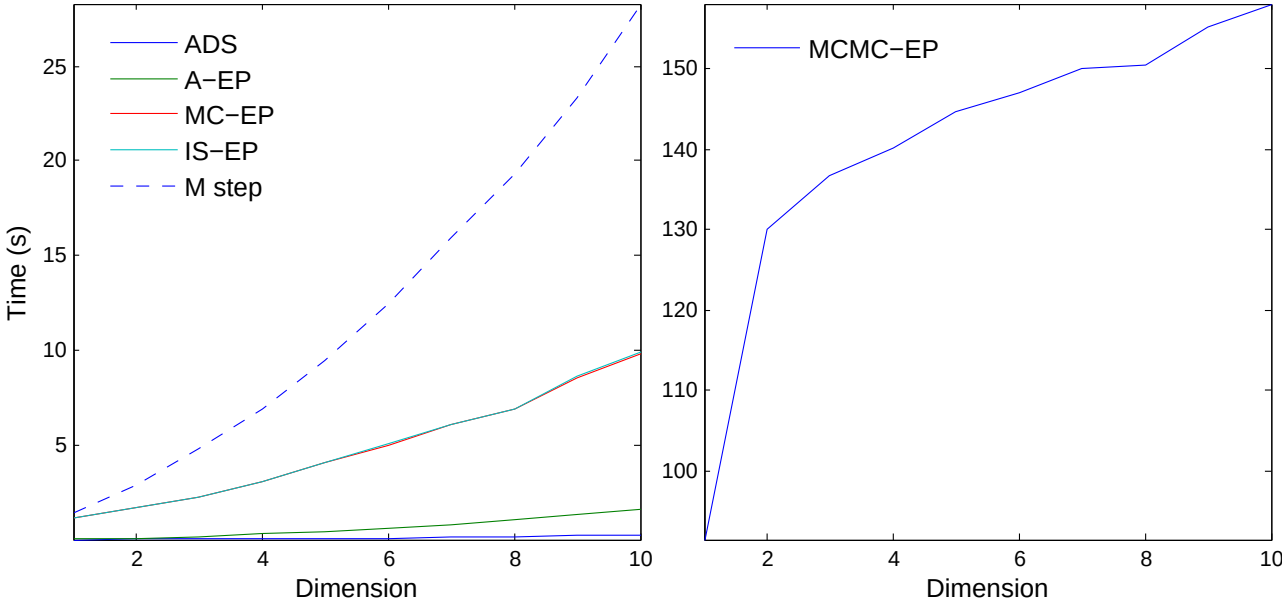


Figure 3.25: Wall clock timings in seconds for the various E step methods and the M step for the approximate-analytic EM algorithm.

We next look at the number of EP iterations required as a function of the number of samples drawn at each step. To test this we ran the EM algorithm with the importance sampling version of EP on three test data sets: the one dimensional kink function (as described in previous sections), the four dimensional cart & pendulum system (section 1.7.1), and the ten dimensional unicycle system (section 1.7.2). Figure 3.26 shows how the EP messages converge over ten EP iterations for multiple runs. After each iteration of EP, the current messages were used to compute the first two moments of the latent state posteriors, i.e. after the ith EP iteration we compute $\boldsymbol{\mu}^i_{t|T}$ and $\Sigma^i_{t|T}$ where,

$$q^{\mathrm{EP},i}_{\mathrm{smooth}}(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{\theta}) \;=\; \alpha^i_t(\boldsymbol{x}_t)\,\beta^i_t(\boldsymbol{x}_t) \;\propto\; \mathcal{N}\left(\boldsymbol{x}_t;\; \boldsymbol{\mu}^i_{t|T},\; \Sigma^i_{t|T}\right) \tag{3.201}$$

for $t = 1$ to T. We then compute the difference between these moments and the values they obtain after ten EP iterations, for example,

$$\boldsymbol{d}^i_{\mu_t} \;=\; \boldsymbol{\mu}^i_{t|T} \;-\; \boldsymbol{\mu}^{10}_{t|T} \tag{3.202}$$

Finally we plot how the standard deviation of $\boldsymbol{d}^i$ varies with EP iteration, where $\boldsymbol{d}^i$ is the concatenation of all the differences, for both the mean and variance and for $t = 1$ to T. Figure 3.26 shows that EP converges rapidly for 100+ samples in the one dimensional system, only requiring one or two iterations. Similar results are seen for the cart and pendulum system although we needed 1000+ particles for quick convergence. In the unicycle system, EP is much slower to converge — it is not clear that we have observed convergence within the ten EP iterations. That said the average distance that the means move each iteration is less than one percent of their value (slightly more for the variance estimates) and so it seems unlikely this has much of an effect on final performance of the model.

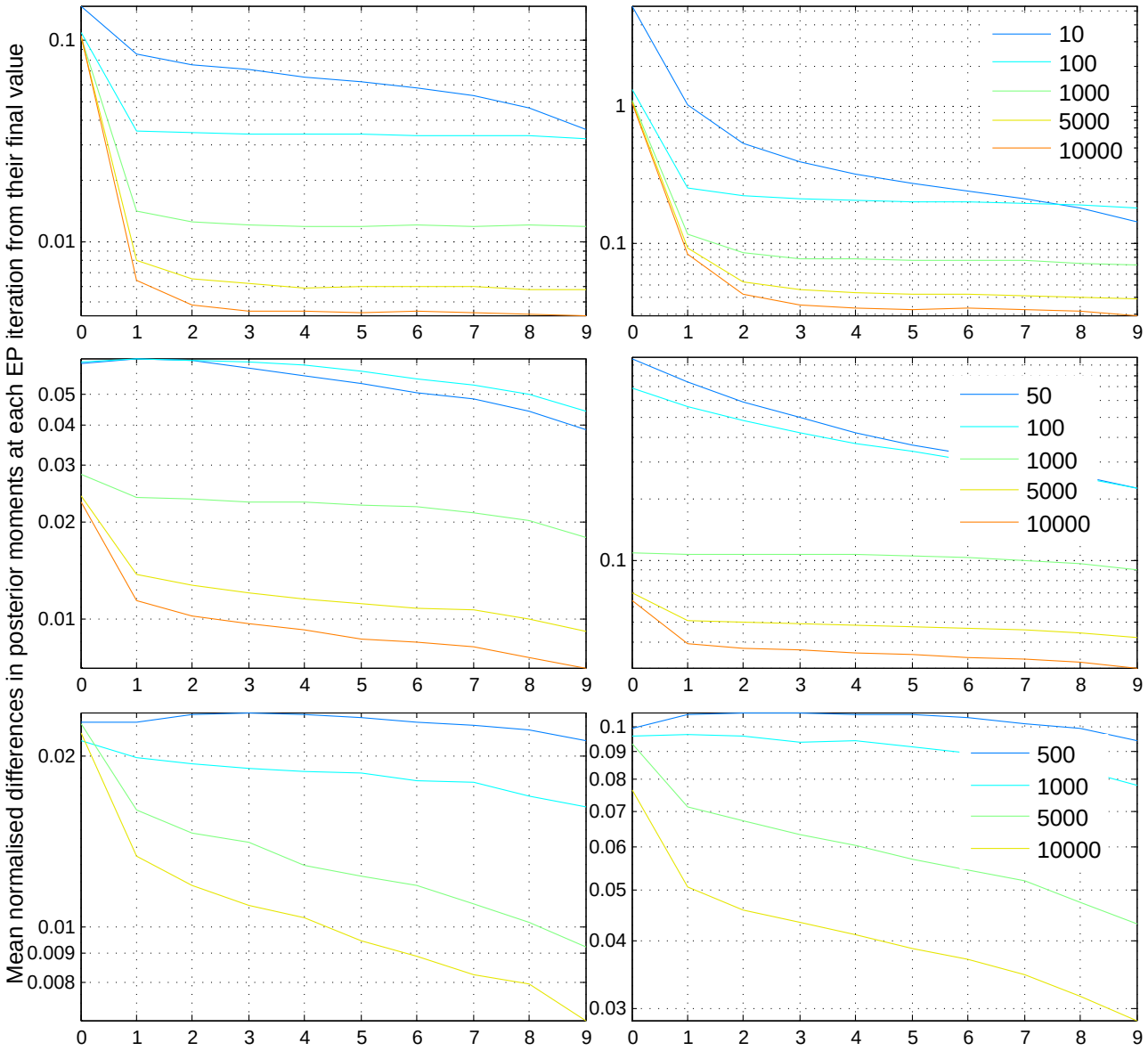


Figure 3.26: How EP's estimates of the means (left column) and variances (right column) of the states vary with EP iteration. The x-axis shows the EP iteration number and the y-axis shows the mean normalised difference between the moment settings at that iteration from their final values at iteration ten. A flat line close to zero indicates that EP has converged. The different lines show how convergence is affected by the number of samples drawn.

We want to see how the number of samples drawn inside each EP update effects performance and compare the sampling EP methods to the moment-matching based approach. To do this we trained the model on the three data sets for different numbers of samples and report the training and test performance in figure 3.27. Training performance is measured by the M step approximate upper bound on the marginal likelihood (equation 3.189) and test performance on the predictive one-step-ahead metric previously described (equation 3.75). We first look at the test performance where we can see clear benefits from using sampling in the one dimensional function and the cart and pendulum, as well as clear evidence of overfitting. For the higher dimensional systems overfitting kicks in early, after only three or four EM steps. We can note that both the sampling and analytic versions of EP lead to overfitting and so it is a more general property of the method, rather than the approximations we made within EP. As one might expect, the largest difference in performance between the analytic

EP method and the sampling EP methods occurs for the smaller dimensional system. As the system dimension increases the sampling approaches perform more poorly, despite the fact that we increased the number of samples being taken.

The figure shows some results which are not so easily explainable and which we do not yet fully understand. For example, for the kink function and the cart & pendulum, the best performance was found with the fewest number of samples. Furthermore, for the cart & pendulum in particular, the training NLP values start to get worse as we make more EM iterations for either 1000+ samples or the analytic approach. This seems to be linked to numerical instabilities in solving for the pseudo-targets, which can lead to poor values of pseudo-target being chosen. There are several places where care needs to be taken with the sampling methods, for example where all the samples can be evaluated to be in an area with zero probability under the model, or where an estimated covariance matrix is non-positive-definite. In these cases we need to define how the algorithm proceeds. These choices can have a large effect on the eventual performance of the method, and it is not always clear what the best choice should be. However, it is clear from figure 3.27 that these issues need to be solved if this method is to be fully competitive with the others.

3.7 Particle Filter

The previous two methods we have looked at make approximations such that we can train the GP state space model using analytic methods (although we introduced some simple sampling within EP). An alternative approach is to use more sophisticated sampling methods, which require fewer approximations. In many cases they can also be a lot simpler to implement than analytic approximations. A lot of research work has been invested into using sampling in dynamical models, with probably the most popular methods being based on sequential Monte Carlo. Sequential Monte Carlo (SMC) methods have been used for many years for nonlinear filtering and smoothing (Cappé et al., 2005). They can also be used for parameter estimation, in either a Bayesian or Maximum Likelihood manner (Kantas et al., 2009). In keeping with the previous two models we present here a maximum likelihood method, which computes an estimate of the marginal likelihood and its derivative w.r.t. the parameters. Our derivation closely follows Algorithm 1 of Poyiadjis et al. (2011).

In SMC we approximate the series of latent state filter posteriors $\{p(\boldsymbol{x}_{1:t} \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})\}_{t=1}^{T}$ by a cloud of weighted samples, or *particles*. These particles are sequentially sampled from the posteriors by propagating the particles from the previous time step forward, using importance sampling and resampling techniques. Gaussian Process state space models readily lend themselves to SMC methods as we can sample directly from the 'optimal' proposal densities (Doucet and Johansen, 2009), given our Gaussian observation model.

Suppose that a set of particles $\{\boldsymbol{x}_{1:t-1}^{i}\}_{i=1}^{N}$ and their corresponding weights $\{w_{t-1}^{i}\}_{i=1}^{N}$, are our approximation to the joint filter posterior of states up until time $t-1$, $p(\boldsymbol{x}_{1:t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$. This implies that we are approximating the posterior with a set of weighted delta functions located at each of the sample points,

$$p(\boldsymbol{x}_{1:t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \approx \sum_{i=1}^{N} w_{t-1}^{i}\, \delta(\boldsymbol{x}_{1:t-1}^{i}) \tag{3.203}$$

Given this distribution for times 1 to $t-1$, we want to extend it to include the next time step t. We

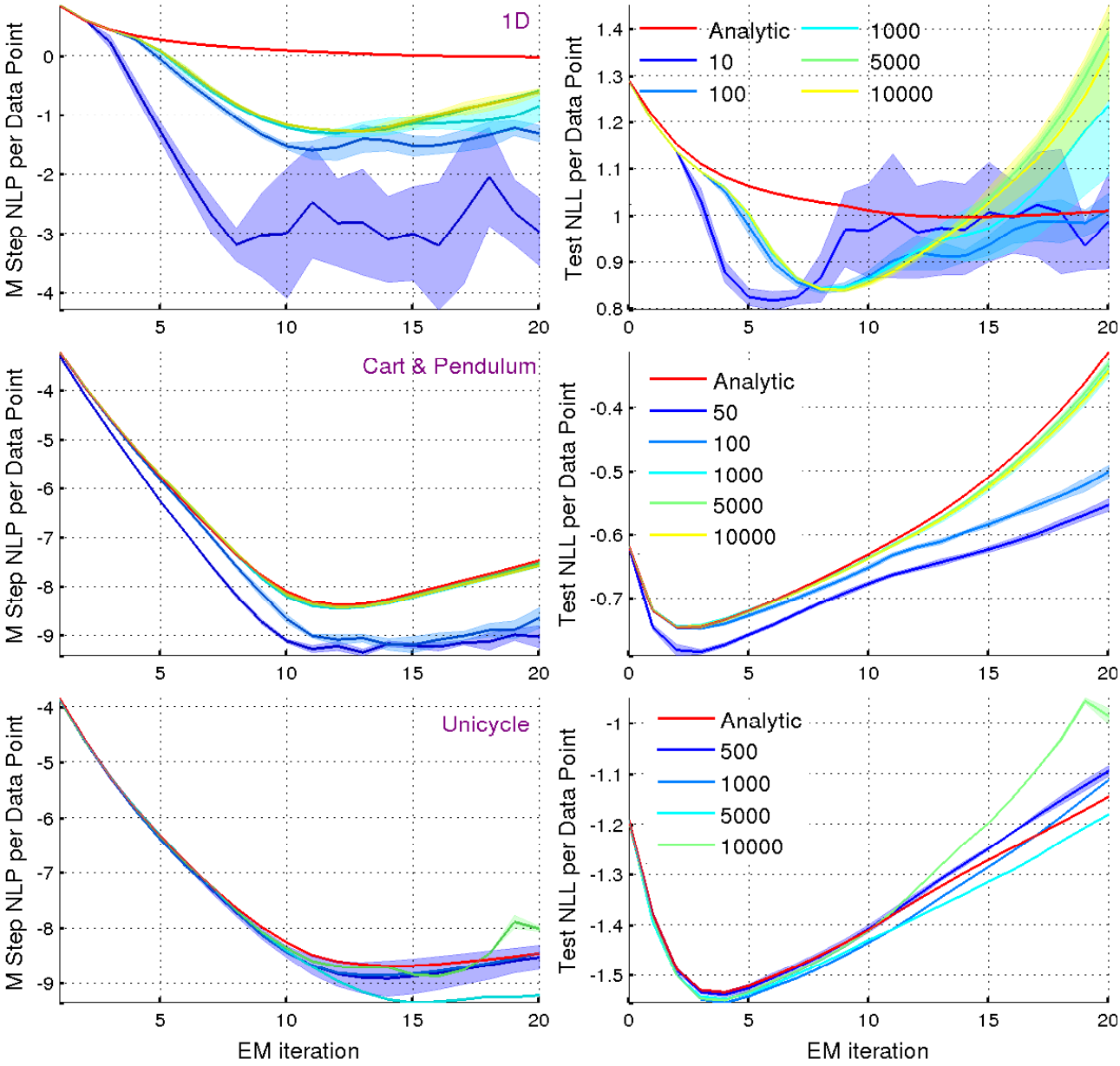


Figure 3.27: Training (left) and test (right) performance for the EP-EM algorithms as a function of EM iteration number. The training performance is measured by the value of the M step objective function, which is an approximate upper bound on the negative log marginal likelihood. The test performance is measured by computing the test negative log probability according to equation 3.75. Note that the test and training metrics are not directly comparable. The top row of plots show results for the 1D test function, the middle for the 4D cart & pendulum test system, and the bottom the 10D unicycle.

do this by propagating the particles from time step $t-1$ forward in time. The simplest method for propagating the particles to time t is the 'bootstrap filter' which uses the relation,

$$\underbrace{p(\boldsymbol{x}_{1:t} \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})}_{\text{Posterior for } t=1:t} \propto \underbrace{p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})}_{\text{Observation}} \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\text{Transition}} \underbrace{p(\boldsymbol{x}_{1:t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Posterior for } t=1:t-1} \tag{3.204}$$

and for each sample $\boldsymbol{x}^i_{1:t-1}$ follows the procedure:

1. Sample from the GP transition density, $\boldsymbol{x}^i_t \sim p(\boldsymbol{x}_t \mid \boldsymbol{x}^i_{t-1}, \boldsymbol{\theta})$

2. Update weight with observation density, $\tilde{w}_t^i = p(\boldsymbol{y}_t \mid \boldsymbol{x}_t^i, \boldsymbol{\theta})\, w_{t-1}^i$

The weights are then normalised,

$$w_t^i = \frac{\tilde{w}_t^i}{\sum_{i=1}^N \tilde{w}_t^i} \tag{3.205}$$

This completes the propagation step and thus we now have a sample estimate of the joint posterior up to and including time t,

$$p(\boldsymbol{x}_{1:t} \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \approx \sum_{i=1}^N w_t^i\, \delta(\boldsymbol{x}_{1:t}^i) \tag{3.206}$$

The problem with this method is that the transition density and the observation may lead to very different distributions over $\boldsymbol{x}_t$ and so sampling from one and weighting by the other can result in very few samples with meaningful weights. In our GP state space model setup we can do a lot better: in fact we can sample from the 'optimal' proposal density (minimal variance of the importance weights) (Doucet and Johansen, 2009), $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{y}_t, \boldsymbol{\theta})$. The key difference with this sampling density is that it conditions on the observation $\boldsymbol{y}_t$ as well as the value of $\boldsymbol{x}_{t-1}$. That is, we sample from the distribution on $\boldsymbol{x}_t$ given both the transition information (from $\boldsymbol{x}_{t-1}$) and the observation information, rather than sampling using just the transition information and then weighting using the observation. The importance weights corresponding to this 'optimal' sampling density are found by evaluating $p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})$. This comes from rewriting the relation in equation 3.204 as,

$$\underbrace{p(\boldsymbol{x}_{1:t} \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})}_{\text{Posterior for } t=1:t} \propto \underbrace{p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\text{Weight function}} \underbrace{p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\text{Sampling density}} \underbrace{p(\boldsymbol{x}_{1:t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Posterior for } t=1:t-1} \tag{3.207}$$

Both the sampling density and the weighting function for the ith particle follow from the joint Gaussian distribution,

$$p(\boldsymbol{x}_t, \boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta}) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{y}_t \end{bmatrix}; \begin{bmatrix} \mathfrak{m}(\boldsymbol{x}_{t-1}^i) \\ C\,\mathfrak{m}(\boldsymbol{x}_{t-1}^i) \end{bmatrix}, \begin{bmatrix} \mathfrak{s}(\boldsymbol{x}_{t-1}^i) & \mathfrak{s}(\boldsymbol{x}_{t-1}^i)\, C^T \\ C\,\mathfrak{s}(\boldsymbol{x}_{t-1}^i) & C\,\mathfrak{s}(\boldsymbol{x}_{t-1}^i)\, C^T + \Sigma_\nu \end{bmatrix} \right) \tag{3.208}$$

Furthermore, we can use the same terms to compute the approximate contribution at time t to the marginal likelihood (equation 3.76), which is actually what our goal is,

$$\begin{aligned} p(\boldsymbol{y}_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) &= \int \underbrace{p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})}_{\text{Weight function}} \underbrace{p(\boldsymbol{x}_{t-1} \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})}_{\text{Posterior for } t-1} d\boldsymbol{x}_{t-1} \\ &\approx \frac{1}{N} \sum_{i=1}^N p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})\; w_{t-1}^i \end{aligned} \tag{3.209}$$

where $p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})$ is the second marginal distribution in equation 3.208 and the $\boldsymbol{x}_{t-1}^i$ are our sample approximation to the posterior at time t, along with weights w_{t-1}^i. Given that we observe $\boldsymbol{y}_t$, the density $p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})$ evaluates to a number for each value of $\boldsymbol{x}_{t-1}^i$.

The sequential importance sampling algorithm outlined above suffers from a systemic weakness however: weight degeneracy. This is the phenomenon whereby after multiple propagation steps one weight dominates all the others. This results in the posteriors being represented by effectively fewer and fewer particles as time increases, which leads to poor approximations. To avoid this problem the sequential importance resampling (SIR) algorithm was introduced (Rubin, 1987; Gordon et al., 1993). In SIR particles are resampled such that those with high weight are replicated and those with low weight

are pruned. As this is a key step in the particle filter it is unsurprising that a number of different resampling algorithms have been published (Douc and Cappé, 2005). The most basic form of resampling is multinomial: the new set of particles are selected from the old set according to a multinomial distribution with probabilities given by the normalised set of weights, i.e. $p(\tilde{\boldsymbol{x}}_t^i = \boldsymbol{x}_t^j) = w_t^j$. However, once again the GP-SSM lends itself to a more advanced method, which gains performance not from changing the sampling methodology but rather the distribution from which we resample. Following the auxiliary particle filter (Pitt and Shephard, 1999), we instead resample according to $p(\boldsymbol{y}_{t+1} \mid \boldsymbol{x}_t^i, \boldsymbol{\theta})$, which results in the importance weights all being identically equal to one. The use of the auxiliary particle filter methodology further improves the particle filter approximation; methods using this step are referred to as *fully adapted*, and the methodology itself has even been referred to as 'optimal' (Douc et al., 2009).

A downside of the resampling step is that we cannot take derivatives of the marginal likelihood by differentiating the particle approximation procedure — resampling is non-differentiable because we are selecting from a set of discrete entities (the previous set of particles). This differs from the previous sampling steps, where we were sampling from continuous probability densities, which is a process we can differentiate by fixing random seeds. Therefore, we make use of the Fisher Identity,

$$\nabla_\theta \log p(\boldsymbol{y}_{1:t} \mid \boldsymbol{\theta}) = \int \underbrace{p(\boldsymbol{x}_{1:t} \mid \boldsymbol{y}_{1:t}, \boldsymbol{\theta})}_{\text{Posterior for } t=1:t} \nabla_\theta \log p(\boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t} \mid \boldsymbol{\theta}) \, d\boldsymbol{x}_{1:t} \tag{3.210}$$

This allows us to swap the calculation of the derivative of the log marginal likelihood (marginalising over $\boldsymbol{x}_{1:T}$), $\log p(\boldsymbol{y}_{1:t} \mid \boldsymbol{\theta})$, with the derivative of the complete log likelihood, $\log p(\boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t} \mid \boldsymbol{\theta})$, which is much easier to compute. We can see from equation 3.210 that we need to find the expectation of the derivatives w.r.t. the filter posterior in order to solve the integral. However, we can immediately recognise that we we already have a particle approximation for the filter posterior: as derived int he previous paragraphs. We therefore just need to compute the derivative of the complete log likelihood for each sample of $\boldsymbol{x}_{1:T}$, which we have already drawn from the filter posterior. The equation for the complete log likelihood was given previously in the M step section of the approximate-analytic EM algorithm, equation 3.189; we repeat it here for the particle approximation,

$$\log p(\boldsymbol{x}_{1:t}^i, \boldsymbol{y}_{1:t} \mid \boldsymbol{\theta}) = \underbrace{\sum_{t=2}^{T} \log p(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})}_{\text{Transition term}} + \underbrace{\sum_{t=1}^{T} \log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t^i, \boldsymbol{\theta})}_{\text{Observation term}} + \underbrace{\log p(\boldsymbol{x}_1^i \mid \boldsymbol{\theta})}_{\text{First term}} \tag{3.211}$$

Each of the terms in equation 3.211 is the logarithm of a Gaussian probability and thus is computable in closed form: the transition term is a GP prediction with process noise, we have a linear Gaussian observation model, and we can place a Gaussian prior on the first state. Furthermore, the sampling approximation means that we can take derivatives w.r.t. the parameters without needing extra approximations such as was required in the approximate-analytic M step. It is interesting to note the difference between using the Fisher identity to find the derivatives and differentiating the particle filter's approximation to the marginal likelihood directly (if we could actually do this): in the first case we are forming an approximation to the true derivatives, in the second we would be differentiating (exactly) an approximation to the marginal likelihood. We suggest that it is more desirable to use an approximation to the true derivatives rather than the true derivatives of an approximation (which is fortunate as we cannot compute the true derivatives of the PF). The derivatives, the marginal

likelihood, and the required particle approximations to the filter posteriors can all be computed sequentially in one forward pass, as in Algorithm 1 of Poyiadjis et al. (2011). Algorithm 4 summarises the steps for computing the approximate negative log marginal likelihood and its derivatives, using the ingredients outlined above.

Algorithm 4 Particle Filter estimate of log likelihood and derivatives

For time step $t = 1$

- Compute first log marginal likelihood approximation term, $\log q^{\mathrm{PF}}(\boldsymbol{y}_1 \mid \boldsymbol{\theta})$, based on the Gaussian prior, $p(\boldsymbol{x}_1 \mid \boldsymbol{\theta})$
- Draw initial particles from prior, $\boldsymbol{x}_1^i \sim p(\boldsymbol{x}_1 \mid \boldsymbol{\theta})$
- Set the derivative, $\boldsymbol{\partial}_t^i = \nabla_\theta \log q^{\mathrm{PF}}(\boldsymbol{y}_1 \mid \boldsymbol{\theta})$

For time step $t > 1$

- Compute the weights using Auxiliary PF methodology,
  - $\rightarrow \tilde{w}_t^i = p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})$
  - $\rightarrow w_t^i = \dfrac{\tilde{w}_t^i}{\sum_{i=1}^N \tilde{w}_t^i}$
- Update log marginal likelihood approximation, $\log q^{\mathrm{PF}}(\boldsymbol{y}_{1:t} \mid \boldsymbol{\theta}) = \log q^{\mathrm{PF}}(\boldsymbol{y}_{1:t-1} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})$
- Resample particle approximation to the latent state posterior and the derivatives at $t-1$, $\{\boldsymbol{x}_{t-1}^i, \boldsymbol{\partial}_{t-1}^i\}_{i=1}^N$, according to w_t^i to obtain a new set, $\{\hat{\boldsymbol{x}}_{t-1}^i, \hat{\boldsymbol{\partial}}_{t-1}^i\}_{i=1}^N$
- For $i = 1 : N$,
  - $\rightarrow$ Sample particles at time t, $\boldsymbol{x}_t^i \sim p(\boldsymbol{x}_t \mid \boldsymbol{y}_t, \hat{\boldsymbol{x}}_{t-1}^i, \boldsymbol{\theta})$
  - $\rightarrow$ Update derivative estimate, $\boldsymbol{\partial}_t^i = \boldsymbol{\partial}_{t-1}^i + \nabla_\theta \log p(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta}) + \nabla_\theta \log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t^i, \boldsymbol{\theta})$

After the final time step T return the estimate of the marginal likelihood and its derivatives.

Once we have an estimate of the marginal likelihood and its derivatives we must now decide how to optimise the parameters. Poyiadjis et al. (2011) suggest using steepest gradient ascent with a fixed step size, however, we found this to give very slow optimisation performance. The parameter search space often has high curvature and so steepest descent struggles to make rapid progress. Furthermore, the evidence from our experiments showed that the marginal likelihood estimate given by the particle filter was a lot noisier than the corresponding estimate of the derivative, as can be seen in figure 3.28. This meant that standard gradient based optimisers which rely on the function value as well as the gradient give very poor performance. We therefore use a derivative-only algorithm to fit the parameters. This algorithm, outlined in algorithm 5, uses the size of the gradient to determine when the search direction minimum has been found. In addition, we found that the derivative estimates were sufficiently stable that we could use an approximate Newton algorithm to provide search directions rather than steepest descent.

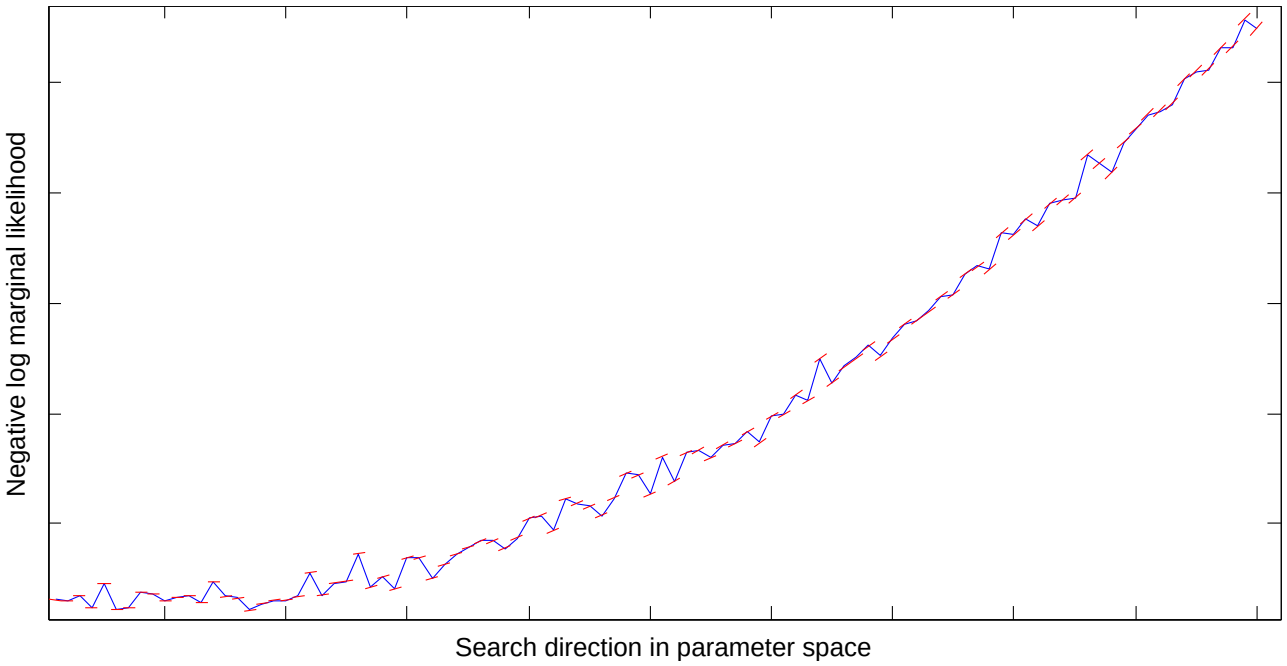


Figure 3.28: Example comparison of the negative log marginal likelihood (NLML) and its derivatives as estimated by a particle filter with 5000 particles on the 4D cart & pendulum system. The continuous blue line represents the estimate of the negative log marginal likelihood for various parameter settings, searching along the direction of steepest gradient descent. The red lines show the derivative of the NLML evaluated at each point and projected along the gradient direction. It is clear to see that the derivatives are in good agreement with each other and with the general shape of the likelihood surface. The individual estimates of the NLML are very noisy however.

Algorithm 5 Direct Optimisation with a Particle Filter

- Use particle filter to estimate derivative of negative log marginal likelihood
- Set initial search direction to steepest descent and step size to a small value
- Loop until optimisation is complete/terminated:
  - Perform a linesearch in search direction:
    - Linesearch:
      - ∗ Take proposed step and evaluate new derivative with PF
      - ∗ Project new derivative into search direction and test its magnitude: if below threshold then accept new point and terminate linesearch
      - ∗ Else, fit a quadratic to all evaluated derivatives in current search direction
      - ∗ Set new parameters to be the location of minimum of the quadratic function
  - If termination criteria are not met: perform BFGS update to obtain new search direction
- Return optimised parameters

3.7.1 Analysis

When implementing this algorithm we must choose the number of particles to use, which will be a speed accuracy trade-off. Figure 3.29 shows how the performance of the PF direct optimisation algorithm varies with the number of particles for the three test systems: the 1D kink function, the 4D cart & pendulum system, and the 10D unicycle system. The figure shows a clear improvement in

performance as we increase the number of particles; for the one dimensional system the improvement stops as five thousand, from there increasing to ten thousand particles has very little effect on either the test or the training probabilities. We can also see that after two hundred function evaluations the algorithm run with five and ten thousand is close to convergence—there is almost no further gain. For the cart & pendulum system the performance gap between five and ten thousand particles is more noticeable, especially in the training NLML, although they converge to a similar value in the test NLP. We see, though, that we need around four hundred function evaluations for the five and ten thousand particle runs to converge, and more for the runs with fewer particles.

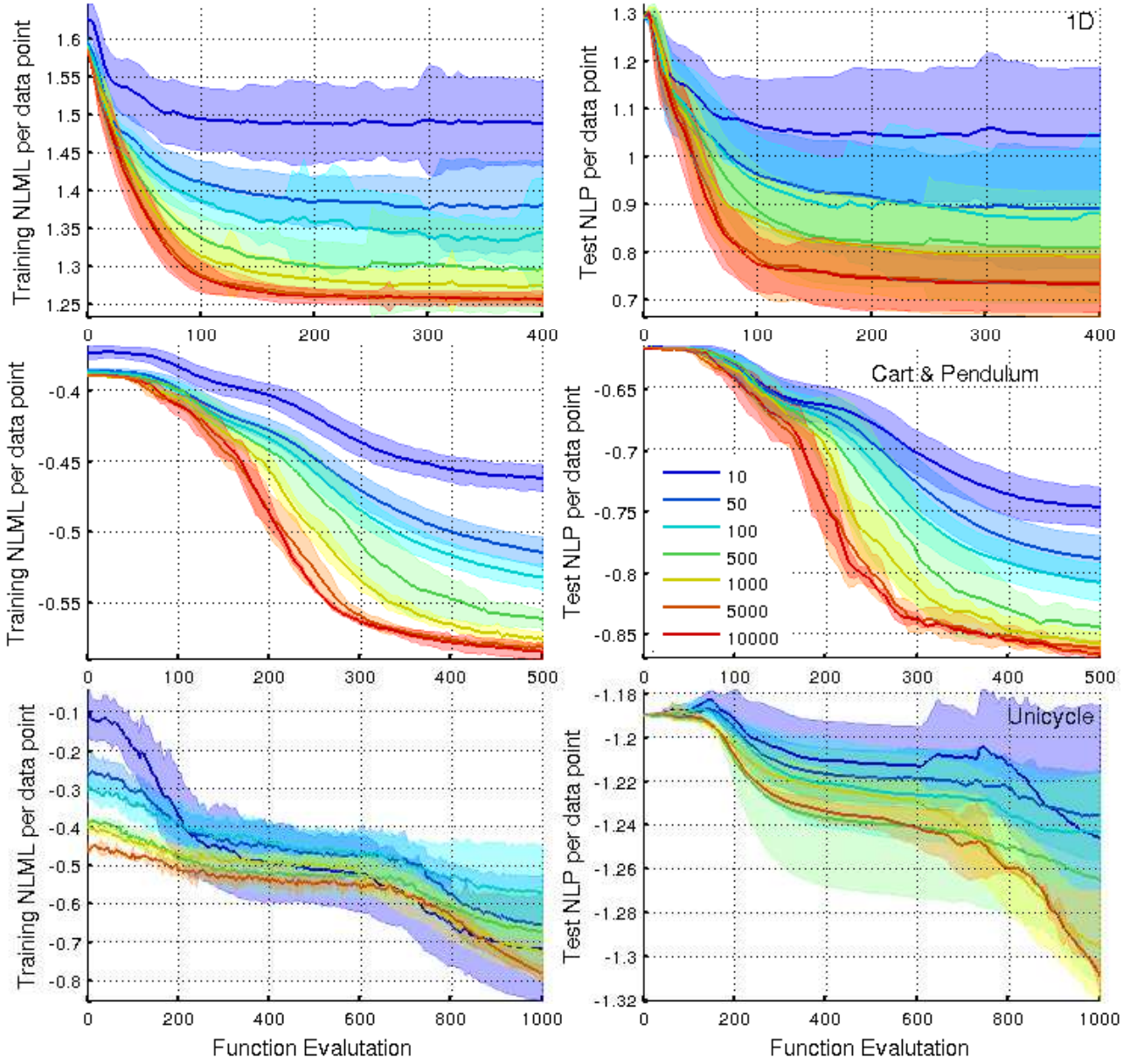


Figure 3.29: Training (left) and test (right) performance for the particle filter algorithm as a function of gradient optimisation iteration number. The training performance is measured by the particle filter's estimate of the negative log marginal likelihood of the training points. The test performance is measured by computing the test negative log probability according to equation 3.75. Note that the test and training metrics are not directly comparable. The top row of plots show results for the 1D test function, the middle for the 4D cart & pendulum test system, and the bottom the 10D unicycle.

Particle methods can often be slow to run and so we investigated running time on a fairly standard desktop computer, with the results shown in figure 3.30. We ran the particle filter algorithm once and

measured how long it took to compute a single set of derivatives for a single observed trajectory of twenty time steps. Clearly, when the algorithm is used to train a model we will need more than one gradient optimisation step and will likely have more than one observed trajectory. However, the time taken should scale linearly with these quantities and so the results in the figure can be used to estimate computation time for a wide range of problems. We looked at time taken for systems with a state dimension ranging from one to ten. We expect to see a quadratic dependence on system dimension, as at the heart of the algorithm we make D lots of GP predictions with a D dimensional input each time. The quadratic shape is clear to see from the plots. We can also see a linear rise in time taken w.r.t. the number of particles used, apart from the rise between one hundred and five hundred particles. This is explained by considering the gain we achieve by using parallelisation: much of the computation is spent on operations such as matrix multiplication, which is automatically parallelised by languages such as Matlab. However, as the number of particles rises the memory requirements reduce the effect of parallelisation and we see the linear increase in time that we expect from the theory.

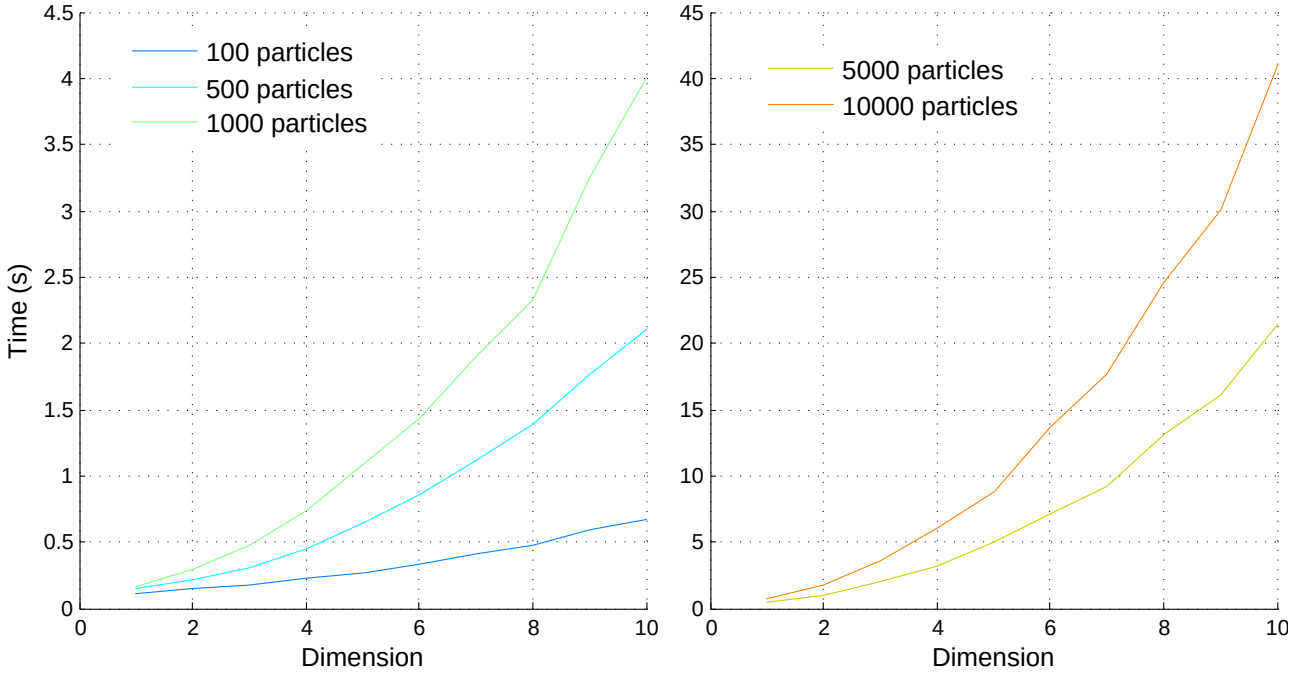


Figure 3.30: Wall clock times for the Particle Filter algorithm to compute the marginal likelihood estimate and its gradient using varying numbers of particles on systems with dimension ranging from 1 to 10. The data set consisted of a single observed trajectory of length 20 time steps.

3.8 Particle EM

The largest drawback with the particle filter method discussed in the previous section is its reliance on estimated derivatives for optimisation—this can make parameter fitting very slow. Study of the marginal likelihood bound used by the M step, equation 3.189, reveals that it is both computable and differentiable for samples of $\boldsymbol{x}_{1:T}$. This suggests a return to the EM algorithm where we use a sampling method to draw trajectories from the latent state posterior, $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, and then optimise the parameters $\boldsymbol{\theta}$ based on these samples. This is different to the sampling methods discussed in the previous EM section, where they played a role inside the EP algorithm. There samples were only used

to find distributions over either the previous time step or the subsequent. Here we wish to sample entire trajectories from the joint smoothing posterior, $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$. This is a much more demanding requirement. The use of Monte Carlo methods in the E step of the EM algorithm was introduced by Wei and Tanner (1990) and has been considered by many authors before, for example Levine and Casella (2001); Booth and Hobert (1999); Chan and Ledolter (1995). For the specific case of the time series latent variable model, sequential Monte Carlo approaches have been used to great effect (Cappé et al., 2005; Olsson et al., 2008; Schön et al., 2011)

3.8.1 Particle E step

The particle filter of the previous section provides an estimate of the required posterior, $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, using a single forward sweep through time (filtering, hence the name). However, due to path degeneracy problems this can be a very poor estimate. Figure 3.31 demonstrates the path degeneracy problem by running a particle filter with ten particles on the 1D 'kink' function shown earlier (see figure 3.18). Whilst the particle filter has good sample diversity marginally at each time step (left plot), which was all that was required in the previous section, the right plot shows that the joint distribution, which we need in this section, is very poorly approximated. A better approximation can be achieved by using a backward simulation method (Lindsten and Schön, 2013) in addition to the forward sweep, although these methods tend to have a complexity which is quadratic in the number of particles, which quickly becomes infeasible. A different set of algorithms come from combining SMC with MCMC; these are typically known as particle MCMC (PMCMC) methods (Andrieu et al., 2010). In these algorithms a particle filter is used to provide a proposal distribution for a MCMC step. PMCMC algorithms have some advantages over traditional particle filters. For example they do not rely on asymptotics, in terms of the number of particles, to generate samples from the true posterior density as particle filters do. We therefore choose to use a PMCMC method to approximate the latent state posterior density in the E step.

We use a recently developed PMCMC method termed particle Gibbs with ancestor sampling (PGAS) (Lindsten et al., 2012). Given an initial sampled trajectory, PGAS uses a conditional particle filter to generate a new trajectory as per a particle Gibbs approach; that is, we draw a new trajectory conditioned on both the observations and on a previous trajectory, $p(\boldsymbol{x}^i_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{x}^{i-1}_{1:T}, \boldsymbol{\theta})$. The conditional particle filter does this by setting one of the particles at each time step to the value of the conditioned data point: if we wish to use N particles then $N-1$ are sampled normally and one is set deterministically to the value from the conditioned trajectory. Given that we fix one particle path, which cannot be pruned away, the path degeneracy problem in a conditional particle filter means that the generated trajectories collapse towards the conditioned trajectory. This means that the newly generated particle paths are very similar to the conditioned trajectory, thus exploration of the space is very slow. The ancestor sampling step in PGAS mitigates this problem by resampling the ancestor of the conditioned particle at each step.

3.8.2 PGAS E Step Comparison

We now test the PGAS method for approximating the latent state posterior and compare it to the E step methods discussion previously. First we look at the same 1D 'kink' function as before (figure

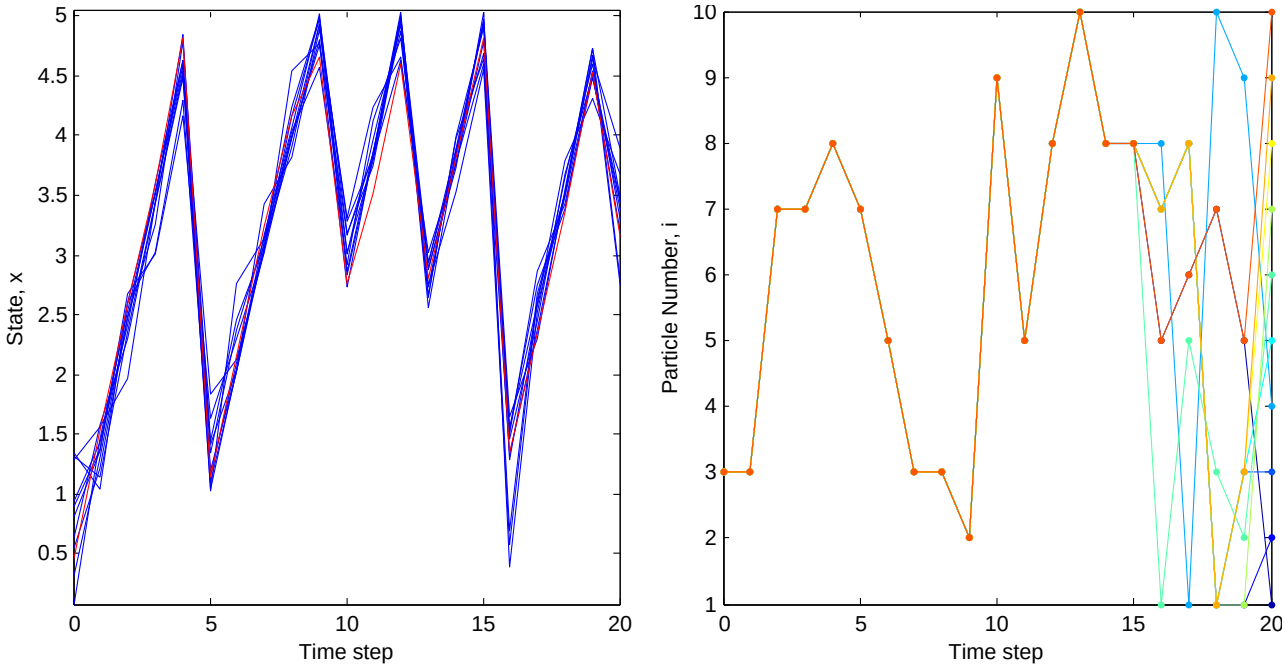


Figure 3.31: Example of the path degeneracy problem. The left plot shows the state values of 10 particles conditioned on an observed trajectory from the 1D 'kink' function described earlier. The red line shows the true latent trajectory. The sampled trajectories cover the true trajectory and have good diversity. The right plot shows the ancestry of each of the particles: tracing each particle back through the resampling steps. Due to the replication and pruning which occurs in resampling the number of separate paths reduces as we move further back in time. By time step 15 all the paths have converged and from then until zero the joint latent state posterior is approximated by a single particle.

3.18). Figure 3.32 shows the inferred latent state posterior using both the importance sampling EP algorithm and the PGAS algorithm over a trajectory of 20 time steps. For illustration purposes only, we fitted a mixture of three Gaussians to the samples generated by the PGAS algorithm. The figure clearly shows the advantage of being able to admit a multimodal distribution for the posterior. This is particularly the case for this 1D function as probability mass either side of the 'kink' is pushed further apart on the next iteration (as the kink is a sharp change in transition dynamics). The figure shows the effects of the EP approximation: the EP posteriors closely resemble Gaussians moment-matched to the PGAS samples. This is particularly noticeable for the first four time steps in figure 3.32, where the EP posteriors have a mean in the middle of the two PGAS posterior peaks, and a larger variance. The figure suggests that the PGAS method outperforms the EP method by a considerable amount, at least for this particular system. We can quantify this by computing the log likelihood for the true latent state under the inferred posteriors, again using a mixture of 3 Gaussians for PGAS. These results are shown in table 3.2, where we can see that PGAS does indeed outperform all the other methods.

Table 3.2: Log likelihoods per data point for the true latent states under the posteriors inferred by each of the algorithms.

ADS/A-EP	MC-EP	IS-EP	MCMC-EP	PGAS
0.31	0.55	0.65	0.75	0.96

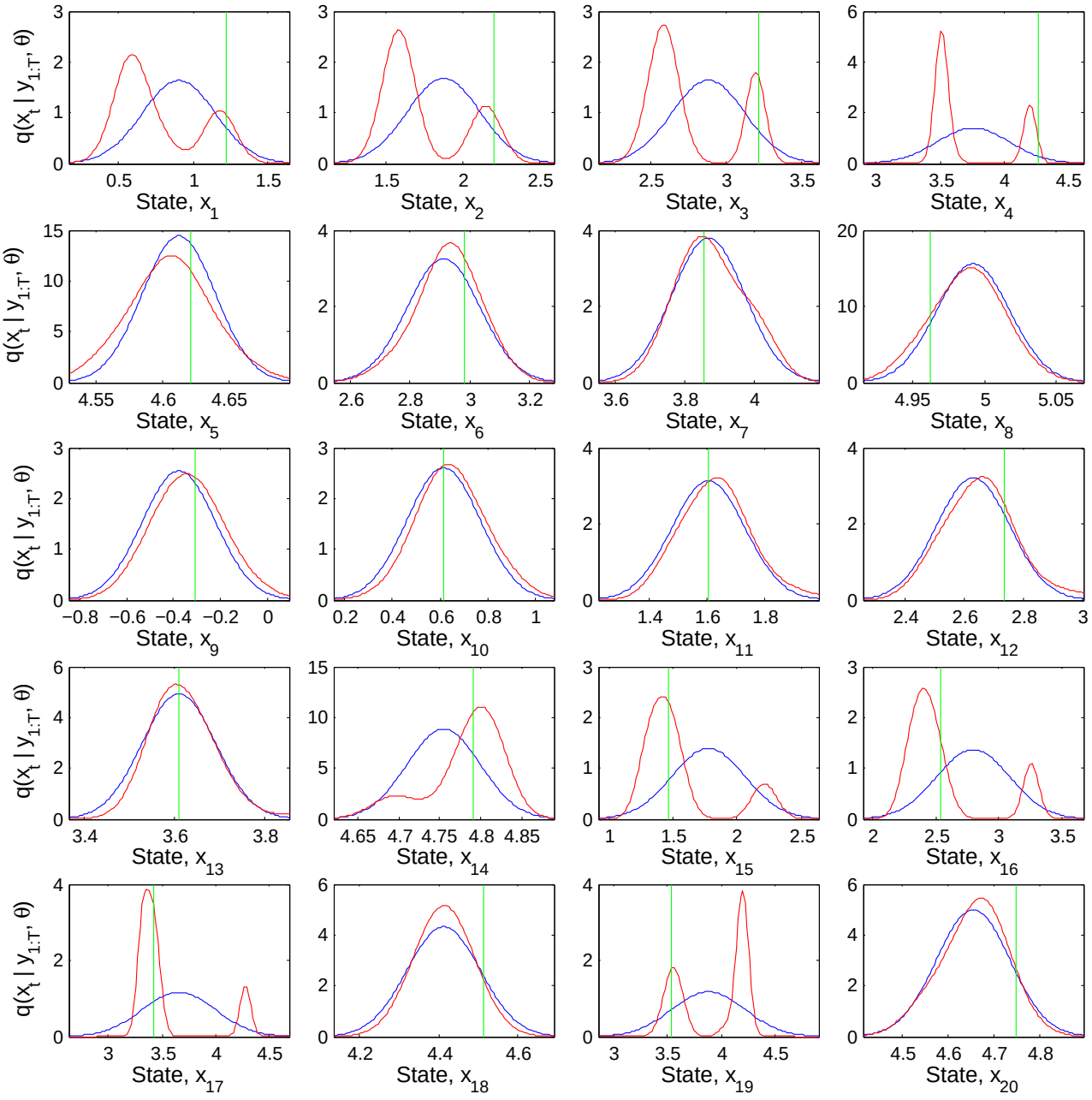


Figure 3.32: Example inferred latent state posteriors using the importance sampling EP algorithm (section 3.6.2) in blue, and the PGAS algorithm in red over a trajectory of 20 time steps for the 1D kink function data set. The vertical green line shows the true location of the latent state. The better performing algorithm is the one with the highest probability at the location of the real latent state. PGAS used 10000 particles and 1000 samples, many more than is actually required.

Algorithm 6 E step using PGAS

- Initialise conditioned trajectory, $\boldsymbol{x}_{1:T}[0]$
- For $k = 1 : K$, run conditional particle filter to sample K trajectories from the joint posterior. Each time, condition on $\boldsymbol{x}_{1:T}[k-1]$

  Conditional Particle Filter

  - For time step $t = 1$
    - For $i = 1 : N-1$, draw initial particles from prior, $\boldsymbol{x}_1^i \sim p(\boldsymbol{x}_1 \mid \boldsymbol{\theta})$
    - Set Nth particle to conditioned value, $\boldsymbol{x}_1^N = \boldsymbol{x}_1^{\text{cond}}$
  - For time step $t = 2 : T$
    - For $i = 1 : N$, compute transition probabilities, $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})$
    - Compute and normalise weights
      - $\tilde{w}_t^i = p(\boldsymbol{y}_t \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{\theta})$
      - $w_t^i = \dfrac{\tilde{w}_t^i}{\sum_{i=1}^N \tilde{w}_t^i}$
    - For $i = 1 : N-1$, resample the particles according to the weights $P(\hat{\boldsymbol{x}}_{t-1}^i = \boldsymbol{x}_{t-1}^j) = w_t^j$
    - Sample the ancestor for the Nth particle (the conditioned particle) according to, $P(a_t^N = j) = w_t^j \, p(\boldsymbol{x}_t^{\text{cond}} \mid \boldsymbol{x}_{t-1}^j, \boldsymbol{\theta})$. This is the 'ancestor sampling' step.
    - For $i = 1 : N-1$, sample new particles $\boldsymbol{x}_t^i \sim p(\boldsymbol{x}_t \mid \hat{\boldsymbol{x}}_{t-1}^i, \boldsymbol{y}_t, \boldsymbol{\theta})$
    - Set Nth particle to conditioned value, $\boldsymbol{x}_t^N = \boldsymbol{x}_t^{\text{cond}}$
  - From the N particles sample one according to the final weights $\{w_T^i\}_{i=1}^N$ and trace back its ancestry to provide a complete trajectory
  - Set $\boldsymbol{x}_{1:T}[k]$ equal to the new sampled complete trajectory
- Return sampled trajectories, $\boldsymbol{x}_{1:T}[1 : K]$

We extend this test by running the PGAS algorithm on the same one hundred trajectories from the 1D 'kink' function as we ran the other E step algorithms on in section 3.6.3. The PGAS E step algorithm, as described in algorithm 6, has two key parameters to set: the number of particles to use in the particle filter, and the number of complete trajectories to sample from the joint latent state posterior. To test the effects of these parameters, we ran the algorithm with five settings of each. The results are shown in figure 3.33 where we show the relative performance of the PGAS E step algorithm to the importance sampling EP E step algorithm. It is immediately obvious from the figure that the number of samples has a much bigger overall effect on performance than the number of particles. This fits with our expectation: the number of particles affects the autocorrelation of the samples drawn but Lindsten (Lindsten et al., 2012) showed the gains made by increasing the number of particles quickly saturated. With on the order of five hundred samples we consistently approximate the latent state posterior more accurately that the EP method using only 100 particles. The conclusion is that it is vitally important to use enough samples else the approximation is likely to be poor.

We would expect the number of particles and samples needed for an accurate approximation to change as the number of dimensions increases from the 1D problem considered so far. We have previously seen how the sample-based EP algorithms performed well in 1D only to have minimal gain in higher

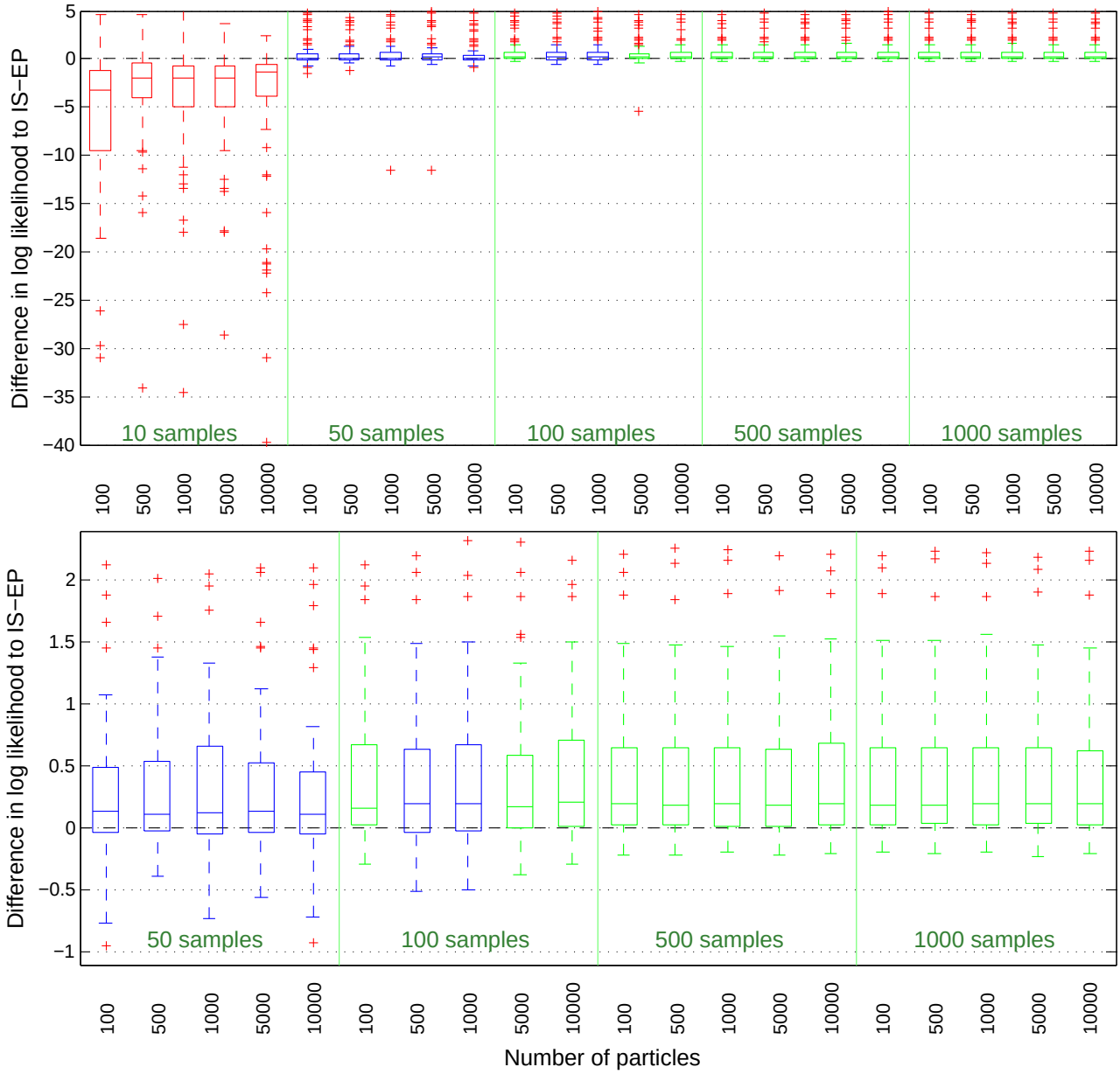


Figure 3.33: Comparison of PGAS to IS-EP algorithm on 100 trials of the 1D 'kink' function. The y-axis shows the difference in log likelihood per data point between PGAS and the importance sampling implementation of EP. The plots show how the performance of PGAS varies as we change the number of particles and the number of sampled trajectories. The bottom plot shows a zoomed section of the top plot. The box plots are coloured red/green if at least 75% of the data lies below/above zero.

dimensions over approximate-analytic methods. We therefore apply the PGAS E step algorithm to the same set of one to ten dimensional systems as before. The results of this experiment are shown in figure 3.34. The experiment shows how more samples are required for higher dimensional problems as expected. It should be noted that the importance sampling EP method is using 10,000 samples inside the EP updates, although these are only one-step samples rather than complete sampled trajectories. The summary plot in the bottom right corner of figure 3.34 shows that for four dimensional and higher problems PGAS has no significant gains over any of the previous methods, at least not for up to a thousand samples. It is interesting to note that once again the number of particles plays a very insignificant role in the performance of the PGAS algorithm. This means we can make computational savings by keeping the number of particles low.

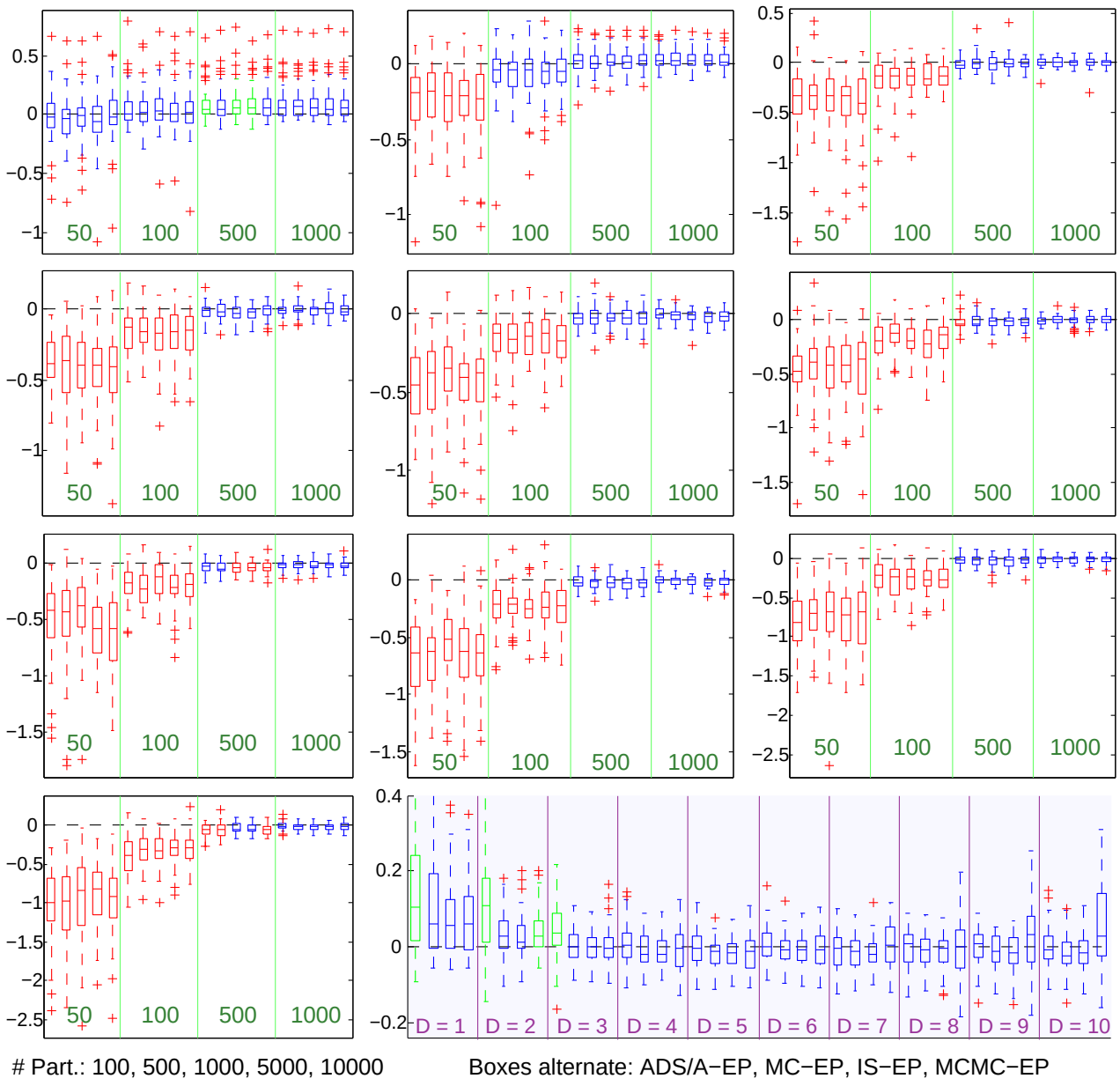


Figure 3.34: Comparison of PGAS to IS-EP algorithm on 50 trials of 1 to 10 dimensional random GP systems. The y-axis shows the difference in log likelihood per data point between PGAS and the importance sampling implementation of EP. The plots show how the performance of PGAS varies as we change the number of particles and the number of sampled trajectories. The vertical green bars separate trials with the same number of samples but differing numbers of particles. The bottom-right plot shows the difference in log likelihood per data point between the PGAS E step with 10000 particles and 1000 samples and the other four E step algorithms over the different dimensions. The box plots are coloured red/green if at least 75% of the data lies below/above zero.

Table 3.3 shows the wall-clock for a typical run of the PGAS E step. Analysis of relative timings of the 100 trials confirms that the complexity is linear in both the number of particles and the number of samples.

Table 3.3: Example wall-clock timings in seconds for a run of the PGAS E step algorithm. These timings could vary significantly depending on implementation and how much parallelisation is used.

Number of: particles ↓, samples →	10	50	100	500	1000
100	0.6	1.7	3.9	12.7	20.8
500	3.3	6.3	10.7	79.0	141.8
1000	7.4	17.8	41.9	156.0	216.0
5000	28.2	80.9	136.8	618.0	1086.4
10000	54.1	140.5	236.1	1393.5	3036.4

3.8.3 Particle M Step

Once we have a sample approximation to the posterior $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{\theta})$ we can look at optimising the parameters in the M step. The upper bound on the negative log likelihood is, once again, given by,

$$\begin{aligned} Q &= -\mathbb{E}_{x_{1:T}\sim p(x_{1:T}|y_{1:T},\boldsymbol{\theta})}\left[\log p(\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:T} \mid \boldsymbol{\theta})\right] \\ &= \underbrace{-\sum_{t=2}^{T} \mathbb{E}_{q(x_{t-1:t}|y_{1:T})}\left[\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\right]}_{\text{Transition term}} \underbrace{-\sum_{t=1}^{T} \mathbb{E}_{q(x_t|y_{1:T})}\left[\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta})\right]}_{\text{Observation term}} \underbrace{-\mathbb{E}_{q(x_1|y_{1:T})}\left[\log p(\boldsymbol{x}_1 \mid \boldsymbol{\theta})\right]}_{\text{First term}} \end{aligned} \tag{3.212}$$

The Transition Term

The transition model term from the negative log likelihood bound is,

$$Q_{\text{trans}} = \sum_{t=2}^{T} -\mathbb{E}_{q(x_{t-1:t}|y_{1:T})}\left[\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\right] \tag{3.213}$$

with,

$$\begin{aligned} -\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) &= -\log \mathcal{N}(\boldsymbol{x}_t;\ \mathfrak{m}(\boldsymbol{x}_{t-1}),\ \mathfrak{s}(\boldsymbol{x}_{t-1}) + \Sigma_\epsilon) \\ &= -\sum_{e=1}^{D} \log \mathcal{N}(\mathfrak{m}_e(\boldsymbol{x}_{t-1}),\ \mathfrak{s}_e(\boldsymbol{x}_{t-1}) + \sigma_{\epsilon,e}^2) \\ &= \frac{1}{2}\sum_{e=1}^{D} \frac{(x_{t,e} - \mathfrak{m}_e(\boldsymbol{x}_{t-1}))^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}) + \sigma_{\epsilon,e}^2} + \frac{1}{2}\sum_{e=1}^{D} \log(\mathfrak{s}_e(\boldsymbol{x}_{t-1}) + \sigma_{\epsilon,e}^2) + \frac{D}{2}\log 2\pi \end{aligned} \tag{3.214}$$

We need to take the expectation over the latent states $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$, which we do by using the K

sampled trajectories drawn in the E step,

$$-\mathbb{E}\left[\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\right] \approx \frac{1}{K}\sum_{i=1}^{K}\left[\frac{1}{2}\sum_{e=1}^{D}\frac{(x_{t,e}^i - \mathfrak{m}_e(\boldsymbol{x}_{t-1}^i))^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2} + \frac{1}{2}\sum_{e=1}^{D}\log(\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2)\right] + \frac{D}{2}\log 2\pi \tag{3.215}$$

This gives the complete expression for the transition model term as,

$$\begin{aligned}\sum_{t=2}^{T} &-\mathbb{E}_{\boldsymbol{x}_{t-1},\boldsymbol{x}_t}\left[\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\right] \\ &\approx \frac{1}{2K}\sum_{i=1}^{K}\sum_{t=2}^{T}\sum_{e=1}^{D}\left[\frac{(x_{t,e}^i - \mathfrak{m}_e(\boldsymbol{x}_{t-1}^i))^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2} + \log(\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2)\right] + \frac{D(T-1)}{2}\log 2\pi \triangleq Q_{\text{trans}}^{\text{PEM}}\end{aligned} \tag{3.216}$$

Only the transition term depends on $\boldsymbol{\beta}$ and within this term only the GP mean $\mathfrak{m}$ is a function of $\boldsymbol{\beta}$. Thus,

$$\begin{aligned}\arg\min_{\boldsymbol{\beta}}\left\{\sum_{t=2}^{T} -\mathbb{E}\left[\log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta})\right]\right\} &= \arg\min_{\boldsymbol{\beta}}\left\{\frac{1}{2K}\sum_{i=1}^{K}\sum_{t=2}^{T}\sum_{e=1}^{D}\frac{(x_{t,e}^i - \mathfrak{m}_e(\boldsymbol{x}_{t-1}^i))^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2}\right\} \\ &= \arg\min_{\boldsymbol{\beta}}\left\{\sum_{i=1}^{K}\sum_{t=2}^{T}\sum_{e=1}^{D}\frac{(x_{t,e}^i - k_e(\boldsymbol{x}_{t-1}^i, \tilde{X})\,\boldsymbol{\beta}_e)^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2}\right\}\end{aligned} \tag{3.217}$$

Each output is independent of the others thus we want to solve,

$$\begin{aligned}\boldsymbol{\beta}_e^* &= \arg\min_{\boldsymbol{\beta}_e}\left\{\sum_{i=1}^{K}\sum_{t=2}^{T}\frac{(x_{t,e}^i - k_e(\boldsymbol{x}_{t-1}^i, \tilde{X})\,\boldsymbol{\beta}_e)^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2}\right\} \\ &= \arg\min_{\boldsymbol{\beta}_e}\left\{||\boldsymbol{\chi}_e - \boldsymbol{\kappa}_e\,\boldsymbol{\beta}_e||^2\right\}\end{aligned} \tag{3.218}$$

where we have stacked the states over time and then over samples,

$$\chi_{t,e}^i = \frac{x_{t,e}^i}{\sqrt{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2}}, \qquad \kappa_{t,e}^i = \frac{k_e(\tilde{X}, \boldsymbol{x}_{t-1}^i)}{\sqrt{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) + \sigma_{\epsilon,e}^2}} \tag{3.219}$$

$$\boldsymbol{\chi}_e = [\chi_{2,e}^1, \ldots, \chi_{T,e}^1, \chi_{2,e}^2, \ldots, \chi_{T,e}^2, \ldots, \chi_{T,e}^K]^T \qquad \text{a } (T-1)K \times 1 \text{ vector} \tag{3.220}$$

$$\boldsymbol{\kappa}_e = \left[\kappa_{1,e}^1, \ldots, \kappa_{T-1,e}^1, \kappa_{1,e}^2, \ldots, \kappa_{T-1,e}^2, \ldots, \kappa_{T-1,e}^K\right]^T \qquad \text{a } (T-1)K \times \tilde{N} \text{ matrix} \tag{3.221}$$

Equation 3.218 can be recognised as the linear least squares problem. The solution is therefore,

$$\boldsymbol{\beta}_e^* = \left(\boldsymbol{\kappa}_e^T\,\boldsymbol{\kappa}_e\right)^{-1}\boldsymbol{\kappa}_e^T\,\boldsymbol{\chi}_e \tag{3.222}$$

The Observation Term

The observation model term from the negative log likelihood bound is,

$$Q_{\text{observ}} \;=\; -\sum_{t=1}^{T} \mathbb{E}_{q(x_t|y_{1:T})} \left[\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t,\, \boldsymbol{\theta})\right] \tag{3.223}$$

From the state-space model,

$$\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta} \;=\; C\,\boldsymbol{x}_t \;+\; \nu_t \tag{3.224}$$

Therefore,

$$\begin{aligned} -\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t,\, \boldsymbol{\theta}) \;&=\; -\log \mathcal{N}(\boldsymbol{y}_t;\; C\,\boldsymbol{x}_t,\; \Sigma_\nu) \\ &=\; -\sum_{e=1}^{D} \log \mathcal{N}(y_{t,e};\; C_e\,\boldsymbol{x}_t,\; \sigma_{\nu_e}^2) \\ &=\; \frac{1}{2}\sum_{e=1}^{D} \frac{(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t)^2}{\sigma_{\nu_e}^2} \;+\; \frac{1}{2}\sum_{e=1}^{D} \log \sigma_{\nu_e}^2 \;+\; \frac{D}{2}\log 2\pi \end{aligned} \tag{3.225}$$

We take the expectation over $\boldsymbol{x}$ by using the Monte Carlo estimate,

$$\begin{aligned} -\mathbb{E}_{\boldsymbol{x}_t}\left[\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t,\, \boldsymbol{\theta})\right] \;&\approx\; \frac{1}{K}\sum_{i}^{K}\left[\frac{1}{2}\sum_{e=1}^{D} \frac{(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2}{\sigma_{\nu_e}^2} \;+\; \frac{1}{2}\sum_{e=1}^{D} \log \sigma_{\nu_e}^2 \;+\; \frac{D}{2}\log 2\pi\right] \\ &=\; \frac{1}{2K}\sum_{i=1}^{K}\sum_{e=1}^{D} \frac{(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2}{\sigma_{\nu_e}^2} \;+\; \sum_{e=1}^{D} \log \sigma_{\nu_e} \;+\; \frac{D}{2}\log 2\pi \end{aligned} \tag{3.226}$$

This gives the complete expression for the observation term as,

$$-\sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}_t}\left[\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t,\, \boldsymbol{\theta})\right] \;\approx\; \frac{1}{2K}\sum_{t=1}^{T}\sum_{i=1}^{K}\sum_{e=1}^{D} \frac{(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2}{\sigma_{\nu_e}^2} \;+\; T\sum_{e=1}^{D} \log \sigma_{\nu_e} \;+\; \frac{TE}{2}\log 2\pi \;\triangleq\; Q_{\text{observ}}^{\text{PEM}} \tag{3.227}$$

The observation noise only appears in the observation term and we can solve for it,

$$\begin{aligned} \frac{\partial Q_{\text{observ}}^{\text{PEM}}}{\partial \log \sigma_{\nu,e}} \;&=\; \frac{\partial}{\partial \log \sigma_{\nu,e}}\left\{\frac{1}{2K}\sum_{t=1}^{T}\sum_{i=1}^{K}\sum_{e=1}^{D} \frac{(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2}{\sigma_{\nu_e}^2} \;+\; T\sum_{e=1}^{D} \log \sigma_{\nu,e}\right\} \\ &=\; \frac{1}{2K}\sum_{t=1}^{T}\sum_{i=1}^{K}(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2 \frac{\partial \sigma_{\nu_e}^{-2}}{\partial \log \sigma_{\nu,e}} \;+\; T \\ &=\; -\frac{1}{K\,\sigma_{\nu_e}^2}\sum_{t=1}^{T}\sum_{i=1}^{K}(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2 \;+\; T \;=\; 0 \end{aligned} \tag{3.228}$$

$$\begin{aligned} \Rightarrow\; T \;&=\; \frac{1}{K\,\sigma_{\nu_e}^2}\sum_{t=1}^{T}\sum_{i=1}^{K}(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2 \\ \Rightarrow\; \sigma_{\nu_e}^2 \;&=\; \frac{1}{K\,T}\sum_{t=1}^{T}\sum_{i=1}^{K}(y_{t,e} \;-\; C_e\,\boldsymbol{x}_t^i)^2 \end{aligned} \tag{3.229}$$

We can also take the derivatives w.r.t. the other parameters and optimise these via gradient descent.

3.8.4 Implementation

As has been previously stated the goal we are trying to achieve is to learn a model for the observed data which is as accurate as possible in as short an amount of time as possible. In this section we will analyse some of the speed-accuracy trade-offs present in the particle EM approach.

There are four key algorithm-parameters (as opposed to model parameters) which affect the runtime and modelling accuracy of the the PGAS-EM algorithm: the number of particles used in the particle filter, the number of sampled trajectories drawn in each E step, the number of optimisation iterations in each M step, and the total number of EM iterations used. Figures 3.33 and 3.34 demonstrated that performance was insensitive to the number of particles used and so we will fix this parameter to a hundred particles—the smallest number which we tested earlier. We now consider how the number of samples drawn affects the runtime of the E and M steps.

Figure 3.35 shows how long the E and M steps take when drawing sampled trajectories of length twenty time steps and with dimension varying from one to ten. The figure shows that the E step time increases close to linearly with dimension whereas the M step increases closer to quadratically (this despite the fact that both steps have a D squared operation in them—making D GP predictions with a D-dimensional state). Both times theoretically will grow linearly with the number of sampled trajectories drawn (until memory limitations come into play). However, the E step cannot be parallelised over samples due to the conditional particle filter—each sample is conditioned on the previous and hence they must be drawn sequentially. In the M step the samples can be considered simultaneously. This means that when actually implemented the M step computation cost grows sub-linearly. This can be seen in figure 3.35 by comparing the plots for 1, 10 and 100 samples. The relative time increase is much greater for the E step than for the M step between these plots. For example, for a 4D system the E step is considerably faster than the M step when we only draw a single sample, they roughly take the same time if we draw 10 trajectories, and the E step is slower than the M step for 100 sampled trajectories. Note that this effect disappears when we compare the plots for 100 and 1000 samples: here the runtime has increased linearly for both the E and the M step. This is because we have already exhausted the gains from parallelisation by 100 samples. In general we will be drawing more samples for larger dimensional systems (due to the results in figure 3.34), therefore it is likely we will always be in the regime where the M step takes longer than the E step.

We now look at how performance is affected by the number of sampled trajectories drawn. This is make slightly more complex by the requirement for multiple observed trajectories in order to obtain enough data to fit the model parameters satisfactorily. If we have multiple observed trajectories and a limited computational budget to spend on samples then for each E step we could select a small number of observed trajectories from which to draw many samples, or do the opposite and use all the trajectories but only draw a very few samples from each. Or we can use a strategy in between, trading-off the number of observed trajectories against the number of trajectories sampled from each. However, we hypothesise that it will be better to take this trade-off to the limit and use all the observed trajectories, even if this means limiting ourselves to very few sampled trajectories per observed trajectory. To test this we collect multiple observed trajectories from three different systems, the 1D kink function, the 4D cart & pendulum system and the 10 dimensional unicycle system. We first draw a fixed number of sampled trajectories from every observed trajectories and use these to train the model. Figure 3.36 shows the results.

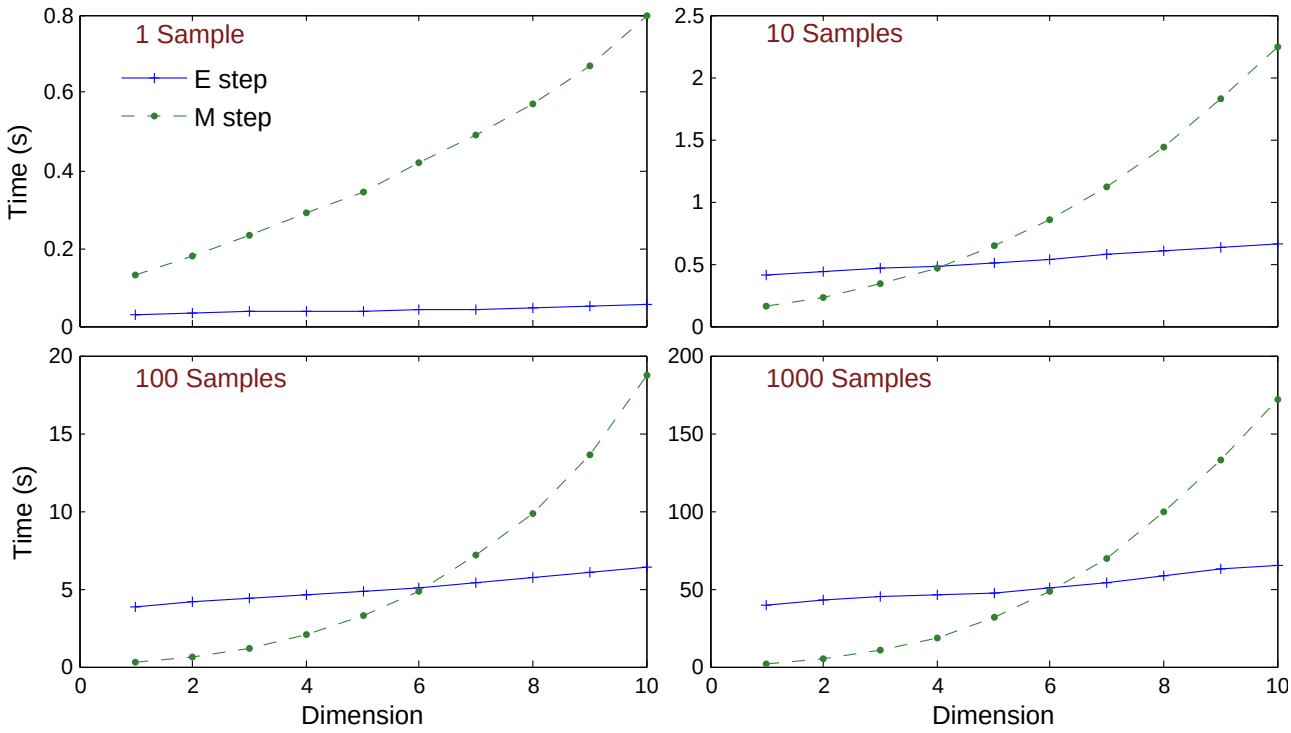


Figure 3.35: Wall clock times in seconds for the PGAS EM algorithm for a single observed trajectory of 20 timesteps, drawing four different numbers of sampled trajectory and using 100 particles. Times are averaged over 50 runs on a quad-core i7 running at 2.79GHz

Figure 3.36 shows that even when we only draw a few, say ten, sampled trajectories from each observed trajectory we achieve comparable performance to the times when we draw many more trajectories (e.g. one hundred). This supports the notion that few samples from more observed trajectories is better than more samples from fewer trajectories. It is hard to draw conclusions about how fast the EM algorithm converges from the figure as it varies strongly. For the 1D system we need around twenty EM steps, for the cart and pendulum we seem to need closer to sixty, but for the unicycle we only need ten to fifteen. We do not fully understand this discrepancy, but feel that it might be important to understand the convergence of the algorithm if we wish to develop it further. In particular the increase in test likelihood between five and ten EM iterations is particularly perplexing. It is pleasing to see that there is no real evidence of overfitting ocurring.

3.8.5 Particle Stochastic Approximation EM

The particle EM algorithm described in the previous sections is very wasteful of samples — after each M step update we must discard all previous samples and collect new ones. The stochastic approximation EM (SAEM) algorithm, introduced in Delyon et al. (1999), was designed to improve on this shortcoming. Recently, Lindsten combined the SAEM with his previously published PGAS algorithm, exploiting the strengths of both methods to create a more sample-efficient particle EM algorithm (Lindsten, 2013a). The basic idea of SAEM is to update an approximation to the Q function in the M step after only completing a partial E step, that is only drawing a small number of sampled trajectories (often just a single sample). In other words we do not wait until we have K samples from the joint latent state distribution before we start optimising the parameters but rather start optimisation using only a small set of samples. If we employ a damping (step size) schedule with the

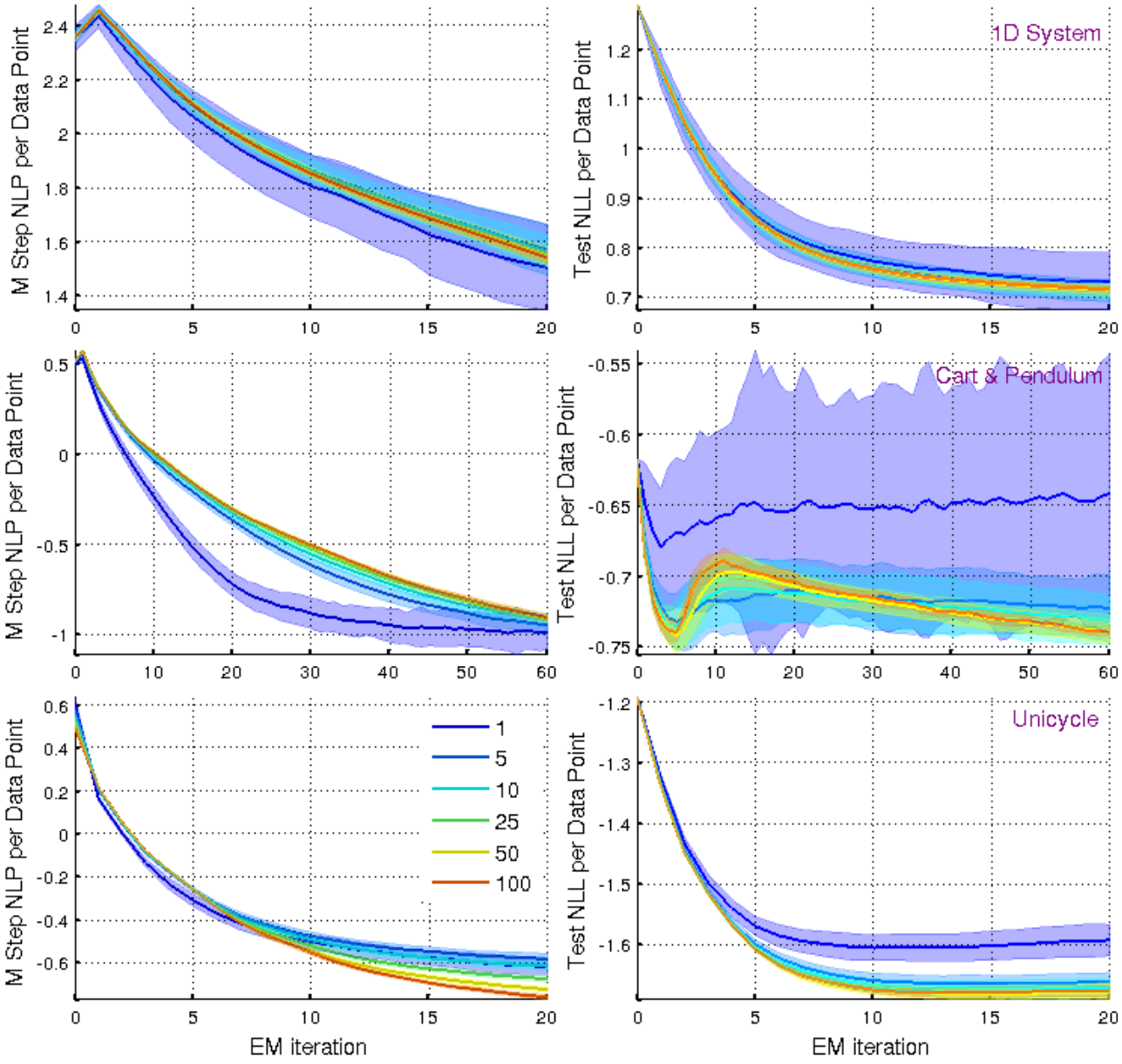


Figure 3.36: Training (left) and test (right) performance for the PGAS-EM algorithm as a function of EM iteration number. The training performance is measured by the value of the M step objective function, which is an approximate upper bound on the negative log marginal likelihood. The test performance is measured by computing the test negative log probability according to equation 3.75. Note that the test and training metrics are not directly comparable. The top row of plots show results for the 1D test function, the middle for the 4D cart & pendulum test system, and the bottom the 10D unicycle.

same conditions as usually applied in methods such as stochastic gradient descent then SAEM can be shown to converge (under a few appropriate assumptions) (Delyon et al., 1999). In SAEM the M step objective function, at the jth evaluation, is given by,

$$\hat{Q}_j(\boldsymbol{\theta}) \;=\; (1-\gamma_j)\,\hat{Q}_{j-1}(\boldsymbol{\theta}) \;+\; \gamma_j \sum_{i=1}^{K_j} \log p(\boldsymbol{x}^i_{1:T}, \boldsymbol{y}_{1:T} \mid \boldsymbol{\theta}) \tag{3.230}$$

where γ_j is a variable step size parameter satisfying,

$$\sum_j \gamma_j \;=\; \infty, \qquad \sum_j \gamma_j^2 \;<\; \infty \tag{3.231}$$

and K_j is the number of samples we draw at the jth iteration.

For the application we are studying here, maximum likelihood in GP state space models, the particle M step is actually more computationally expensive than the E step. Thus we don't expect much of a computational saving as a result of using SAEM methods. For this reason we will draw multiple samples ($K_j > 1$) at each EM iteration, which limits the number of M steps we perform. The alternative, as suggested in (Lindsten, 2013a), is to set $K_j = 1$ and then use the complete weighted system of particles from the particle filter to compute the new term in $\hat{Q}_j$. This is too computationally demanding for our M step to compute.

We define $\hat{\gamma}_{kj}$ to be the discount applied to the Q function evaluated on the sampled trajectory drawn at the kth EM iteration given that we are currently on the jth EM iteration, with $j \;\geq\; k$,

$$\hat{\gamma}_{kj} \;=\; \gamma_k \prod_{l=k+1}^{j} 1 - \gamma_l \tag{3.232}$$

With this definition we can rewrite equation 3.230 as,

$$\hat{Q}_j \;=\; \sum_{k=1}^{j} \hat{\gamma}_{kj}\, q_k \tag{3.233}$$

where,

$$\begin{aligned} q_k \;&=\; -\sum_{i=1}^{K_k} \log p(\boldsymbol{x}_{1:T}^i,\, \boldsymbol{y}_{1:T} \mid \boldsymbol{\theta}) \\ &=\; -\log p(\boldsymbol{x}_1^i \mid \boldsymbol{\theta}) \;-\; \sum_{t=2}^{T} \log p(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^i,\, \boldsymbol{\theta}) \;-\; \sum_{t=1}^{T} \log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t^i,\, \boldsymbol{\theta}) \end{aligned} \tag{3.234}$$

The transition term is,

$$\hat{Q}_{j,\text{trans}}^{\text{SAEM}} \;=\; \sum_{k=1}^{j} \hat{\gamma}_{kj} \left(\frac{1}{2K_k} \sum_{i=1}^{K_k} \sum_{t=2}^{T} \sum_{e=1}^{E} \left[\frac{(x_{t,e}^i \;-\; \mathfrak{m}_e(\boldsymbol{x}_{t-1}^i))^2}{\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) \;+\; \sigma_{\epsilon,e}^2} \;+\; \log(\mathfrak{s}_e(\boldsymbol{x}_{t-1}^i) \;+\; \sigma_{\epsilon,e}^2) \right] \;+\; \frac{E(T-1)}{2} \log 2\pi \right) \tag{3.235}$$

from which we can find β analogously to before,

$$\boldsymbol{\beta}_e^* \;=\; \left(\hat{\boldsymbol{\kappa}}_e^T\, \hat{\boldsymbol{\kappa}}_e \right)^{-1} \hat{\boldsymbol{\kappa}}_e^T\, \hat{\boldsymbol{\chi}}_e \tag{3.236}$$

where we have stacked discounted versions of $\boldsymbol{\kappa}$ and $\boldsymbol{\chi}$ from previous EM iterations,

$$\hat{\boldsymbol{\kappa}}_e \;=\; \begin{bmatrix} \sqrt{\hat{\gamma_{1j}}}\boldsymbol{\kappa}_e^1 \\ \vdots \\ \sqrt{\hat{\gamma_{jj}}}\boldsymbol{\kappa}_e^j \end{bmatrix}, \qquad \hat{\boldsymbol{\chi}}_e \;=\; \begin{bmatrix} \sqrt{\hat{\gamma_{1j}}}\boldsymbol{\chi}_e^1 \\ \vdots \\ \sqrt{\hat{\gamma_{jj}}}\boldsymbol{\chi}_e^j \end{bmatrix} \tag{3.237}$$

To find the observation noise, consider,

$$\hat{Q}^{\text{SAEM}}_{j,\text{observ}} = \sum_{k=1}^{j} \hat{\gamma}_{kj} \left(\frac{1}{2K_k} \sum_{i=1}^{K_k} \sum_{t=1}^{T} \sum_{e=1}^{E} \frac{(y_{t,e} - C_e \boldsymbol{x}_t^i)^2}{\sigma_{\nu e}^2} + T \sum_{e=1}^{E} \log \sigma_{\nu e} + \frac{TE}{2} \log 2\pi \right) \tag{3.238}$$

$$\begin{aligned} \frac{\partial \hat{Q}^{\text{SAEM}}_{j,\text{observ}}}{\partial \log \sigma_{\nu,e}} &= 0 \\ \Rightarrow T \sum_{i=1}^{j} \hat{\gamma}_{kj} &= \frac{1}{\sigma_{\nu e}^2} \sum_{k=1}^{j} \frac{1}{K_k} \sum_{i=1}^{K_k} \sum_{t=1}^{T} \hat{\gamma}_{kj} (y_{t,e} - C_e \boldsymbol{x}_t^i)^2 \\ \Rightarrow \log \sigma_{\nu,e} &= \frac{1}{2} \log \frac{\sum_{k=1}^{j} \sum_{i=1}^{K_k} \sum_{t=1}^{T} \frac{\hat{\gamma}_{kj}}{K_k} (y_{t,e} - C_e \boldsymbol{x}_t^i)^2}{T \sum_{i=1}^{j} \hat{\gamma}_{kj}} \end{aligned} \tag{3.239}$$

Due to time restrictions and because the M step is likely to be the most limiting stage in the PGAS-EM algorithm we do not include this variant in our comparisons. We would like to extend the experiments in the future though to thoroughly test this proposed method alongside the other approaches.

3.9 Comparison

In this section we compare the methods which we have described previously in this chapter along with the NIGP method from chapter 2, and a basic, order 1, GP auto-regressive model. This approach, denoted 'AR1-GP', fits a standard GP to the data set $\{\boldsymbol{y}_{1:T-1}, \boldsymbol{y}_{2:T}\}$.

3.9.1 Theoretical comparison

NIGP (chapter 2) is not a state space model and so should perform worse than the four methods outlined in the preceeding sections. However, it is much simpler and faster to run than any of them, which could make it attractive in some applications. The non-EM approaches, the 'Direct' method and the particle filter, do not have to perform smoothing over the latent state variables. The smoothing step is considerably more complex than filtering as we only have a forward dynamical model available — we cannot invert the GP. Thus it is a considerable advantage for the direct method and the particle filter not to have to perform smoothing. Not only does it saves computation but, for the analytic approach, it avoids having to make further approximations: the analytic EM algorithm makes pairwise joint Gaussian approximations to the latent state posterior whereas the analytic direct approach only makes marginal Gaussian approximations. The EM algorithm also makes an additional approximation in the M step, where the expectation of a ratio is replaced by the ratio of the expectations. For both EM algorithms, there is a strong dependency between the noise levels found in the M step and the posterior distribution found in the E step: a high initial noise level means the posteriors have high variance, which restricts the M step's ability to reduce the noise variances. We found they also struggle to differentiate between the two types of noise: process and observation. The direct gradient descent methods do not separate out inference and parameter optimisation, and thus they avoid these difficulties.

The EM approaches have the advantage of being able to solve for the pseudo-targets and observation noise variance parameters. This greatly reduces the number of parameters that need to be optimised

and hence improves the optimisation performance. For example, in the unicycle task we used fifty pseudo-points, which means that the direct and particle filter methods had 780 parameters to optimise (500 pseudo-targets), whereas the EM methods just had 270 parameters. Having said that, solving for the pseudo-targets proved difficult to code in a numerically stable manner, which limited the advantages which could be gained from this. The particle Gibbs approach also solves for these parameters but does not have to make any of the approximations that the analytic EM method does. It does have a potential scale issue in that one must sample a set of latent trajectories for every observed trajectory. The M step then makes a GP prediction for every sampled transition (around 8000 in our experiments), which could get costly. However, these predictions are very easily parallelised if need be as they only consist of matrix multiplications.

Whilst the derivatives computed by the particle filter were found to be fairly robust, the function values varied by a significant amount. This meant we couldn't use the function values for optimisation: our solution of using a derivative-only optimiser has the potential to go wrong if sufficient checks aren't put in place. We also found optimisation was very slow, even with BFGS gradient steps.

3.9.2 Computational Time

Each of the algorithms have a number of parameters which control their performance, for example the number of particles to use or the number of optimisation steps to run for. We explored the effect of various settings for these parameters in the relevant preceding sections. Here we select a setting for each of these parameters based on those results: we set the parameters which give the fastest run time whilst still being 'close' to the best performance possible for that algorithm. This is somewhat of a subjective choice which will be dependent on how a particular user needs to trade-off computational time and performance. The resulting settings are shown in table 3.4.

Table 3.4: Settings for various training parameters for each of the methods

	1D function	Cart & pendulum	Unicycle
Direct	750 gradient descent steps	350 gradient descent steps	750 gradient descent steps
ADS-EM	20 EM iterations	3 EM iterations	3 EM iterations
IS-EP-EM	10 samples, 4 EP iterations, 6 EM iterations	50 samples, 4 EP iterations, 3 EM iterations	1000 samples, 4 EP iterations, 3 EM iterations
PF	5000 particles, 250 gradient descent steps	1000 particles, 1000 gradient descent steps	1000 particles, 1000 gradient descent steps
PGAS-EM	10 samples, 20 EM iterations	25 samples, 60 Em iterations	25 samples, 15 EM iterations

Running time is an important metric for these algorithms if they are going to be used in real applications. It is hard to objectively judge however as it is very dependent on implementation. We report complete algorithm run times in table 3.5 for comparison, although these results should be treated carefully as they will vary a lot for different machines and implementations. We used up to six parallel computation threads where possible to increase speed, although the use of parallelisation could be greatly expanded if the necessary hardware is available. The times in table 3.5 are disappointingly slow in comparison to the standard GP. However, our algorithms are by no means optimised and there could be significant time gains to be made. The EP-EM method has a smaller than expected run time

due to its tendency to overfit. This meant we greatly reduced the number of EM iterations it used (to only three for the cart & pendulum, and unicycle tasks). The EM performance results in section 3.6.5 surprisingly showed the best performance came with only a small number of samples, which again greatly helps its running time. The particle filter is the slowest to run, although potentially could benefit the most from parallelisation.

Table 3.5: Training times in minutes. For a comparison of the E and M steps see figures 3.25 and 3.35 in the relevant sections.

Method	1D function	Cart & pendulum	Unicycle
AR1-GP	0.01	0.07	0.26
NIGP	0.04	1.02	3.59
Direct	4.62	26.5	366
ADS-EM	6.21	16.1	80.1
IS-EP-EM	1.95	18.7	81.5
PF	6.37	489	2016
PGAS-EM	0.504	63.4	100

3.9.3 Learnt Noise Levels

The state space models learn values for the process and observation noise variances (or rather the log of their standard deviations). The standard GP model and NIGP both learn a single set of noise variances, which can be thought of as observation noise. We can use the learnt values of the noise parameters to analyse the modelling performance of each method. Accurately deducing the amount of noise present in the system suggests that the model has accurately captured the true latent signal within the data, which implies it is a method performing well. If a model overestimates the amount of noise it is likely to be underfitting the signal, whereas a too small learnt noise variance indicates probable overfitting. Table 3.6 shows the percentage error in the learnt total noise standard deviations for each of the methods on the three test data sets. The total noise standard deviation is the square root of the sum of the process and observation noise variances. We combine the two types of noise for two reasons: firstly we are interested here in the split between noise and signal rather than between the two different types of noise. Secondly, the standard GP model and NIGP do not differentiate between process and observation noise and so it makes for a fairer comparison for these approaches.

In general the worst performing algorithm is the standard GP regression model, mapping directly from $\boldsymbol{y}_{t-1}$ to $\boldsymbol{y}_t$ without using the state space model. Because of this it dramatically overestimates the noise levels. NIGP still does not use a state space model but it does take into account noise on the inputs and thus is able to dramatically improve over the standard GP, it even occasionally finds the most accurate estimate of the noise levels. The direct gradient descent via moment-matching algorithm, mostly produces accurate estimates of the noise. It does tend to overfit slightly, however, more often than not underestimating the amount of noise present. The analytic EM methods tend to overestimate the noise, sometimes by a considerable amount. The sample-based version of EP does show improvement over the purely analytic version however; this is particularly noticeable for the cart and pendulum data set. The particle filter overall performs well, it finds the most accurate estimate for the 1D function, but tends to be slightly more unreliable for the 10D unicycle – achieving the best result for the yaw angular velocity but then dramatically overestimating the noise for the wheel angular

Table 3.6: Percentage errors in learnt total noise standard deviations (combining process and observation noise where they are learnt separately) for the 1D kink function, the 4D cart & pendulum system, and the 10D unicycle system. The top row of numbers shows the true noise standard deviations and the lower rows the percentage error in the learnt value for each of the methods. Negative values indicate the method underestimated the noise level.

		1D kink	Cart & Pendulum			
			$\dot{\theta}$	$\dot{x}$	x	θ
	True	0.5600	0.1000	0.4000	0.1745	0.0175
% Error	AR1-GP	**164**	109	**74.8**	**119**	**137**
	NIGP	16.1	3.72	7.57	**1.96**	**-2.40**
	Direct	-8.45	**0.232**	**-6.45**	-6.61	-14.0
	ADS-EM	16.4	**213**	28.8	69.3	133
	IS-EP-EM	13.8	-4.40	-11.4	-3.34	-3.99
	PF	**5.10**	1.24	-6.55	-7.17	-14.1
	PGAS-EM	9.52	-3.70	-10.9	-7.03	-12.5

		Roll	Yaw	Wheel	Pitch	Flywhl	x Pos.	y Pos.	Roll	Yaw	Pitch
	True	0.1164	0.1164	0.1164	0.1164	0.1164	0.0100	0.0100	0.0175	0.0175	0.0175
% Error	AR1-GP	**39.3**	16.1	**539**	**196**	**45.2**	**230**	**122**	**76.8**	**52.0**	**164**
	NIGP	22.3	25.1	-37.2	**1.12**	7.31	48.6	31.2	-13.3	**0.44**	-4.23
	Direct	**0.879**	51.2	-22.9	8.82	-5.71	**-0.448**	**-3.68**	-8.12	-41.1	**-1.49**
	ADS-EM	69.5	**46.2**	416	166	39.1	213	103	55.3	38.6	106
	IS-EP-EM	57.9	**46.5**	382	141	32.4	186	88.8	45.5	30.2	99.3
	PF	20.9	**7.48**	430	104	18.1	166	69.5	57.3	37.3	93.2
	PGAS-EM	4.52	39.7	**-13.9**	-3.40	**-0.761**	37.3	16.5	**0.829**	-21.7	9.55

velocity. PGAS provides the most reliable estimates of the noise, always finding values within 40% of the true values, a small range than the other approaches. It performs, comparatively, particularly well on the unicycle dataset, which is perhaps unexpected for a sampling based algorithm.

3.9.4 Predictive performance

We test the methods on the three data sets we have been using in this chapter. The 1D function, consisting of a close-to-linear rise followed by a sharp kink, is very difficult to learn with linearised methods such as the EKF: the result of a filtering/smoothing step is highly dependent on whether the system is linearised on one side of the kink or the other. This also causes significant problems for the E step of GPIL, as discussed in section 3.6. Figure 3.37 shows the learnt latent transition functions for the four 'parametric-GP' methods. The analytic direct optimisation approach is over-confident, underestimating the process noise. The two EM approaches suffer from the opposite problem, overestimating the process noise, particularly the particle EM method. Note however, that the estimated total noise level, process plus observation noise, as shown in table 3.6 has the analytic EM approaches overestimating the noise by a larger amount than the particle EM method. This implies that the analytic EM approaches particularly overestimate the observation noise whereas the particle EM approach overestimates the process noise. The particle filter gives an excellent estimate for the process

noise and is the only method to correctly fit the tail of the transition function on the right hand side.

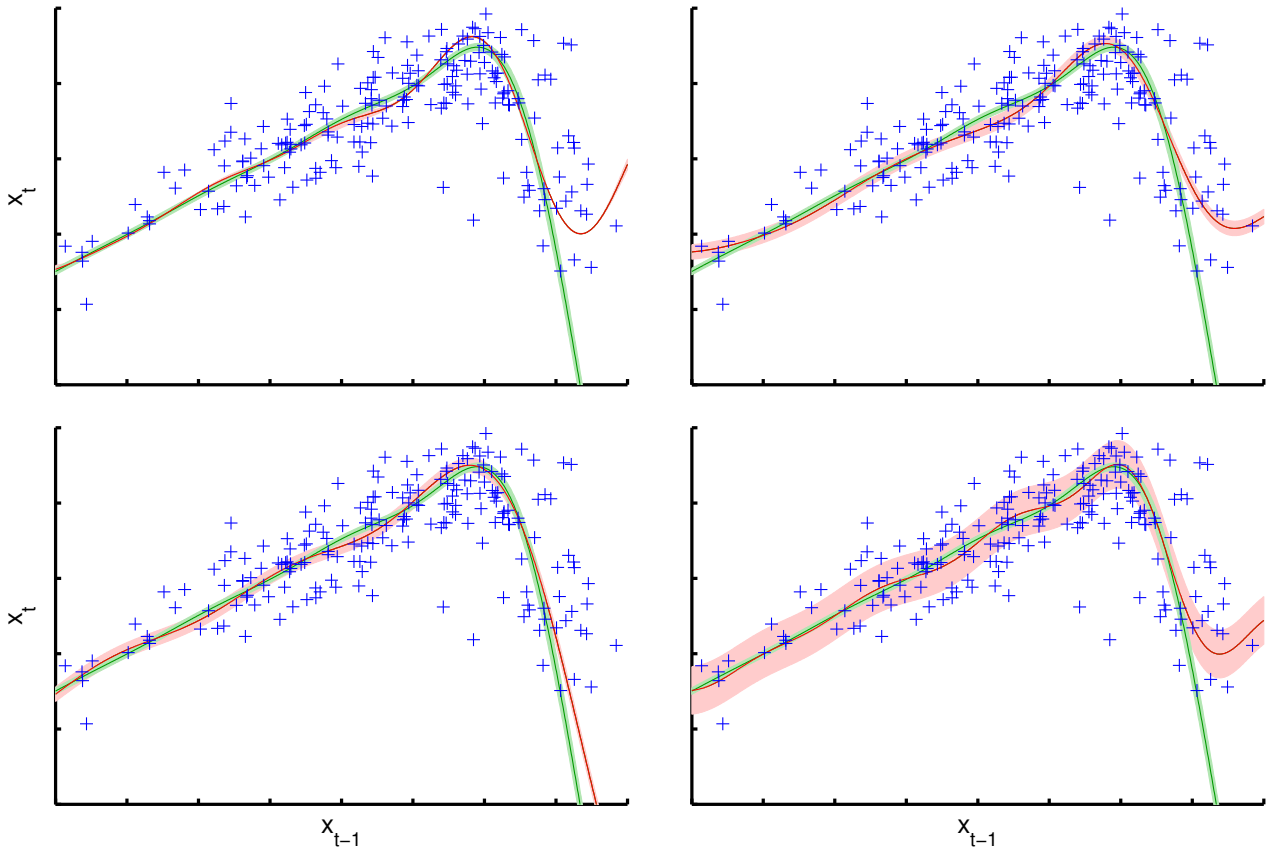


Figure 3.37: The learnt latent transition functions for the 1D data set for the four different methods. Top row: analytic methods, bottom row: sampling; left column: direct optimisation, right: EM. The narrow green 'kink' function is the true latent function, the shaded region showing two standard deviations of process noise. The red function is the learnt transition function, with the shaded region showing both uncertainty about the function and process noise. The blue crosses represent the *observed* transitions, shown to illustrate the task complexity.

To quantify predictive performance, we first look at one-step-ahead predictions using the same test metric as before (equation 3.75). Figure 3.38 shows the comparative results over each of the twenty test trajectories for each system, and over the repeats for each of the stochastic methods. As we would expect, the standard GP is outperformed in nearly all cases by each of the other methods (the only exception being the particle filter on the unicycle data set, which suffers from the high dimensionality of the system — see figure 3.29). Also as predicted, the NIGP method outperforms the classic GP but, in turn, cannot match the performance of the state space models (again excepting the particle filter on the unicycle data). Out of the state space models it is the direct, moment-matching, optimisation approach which most regularly produces the best results. The particle filter's performance is strongly dependent on the dimensionality of the system, achieving worse results as the dimension increases. It particularly struggles with the unicycle data set where it can only achieve comparative performance with the standard GP regression model. We expected it to perform better on the kink function dataset, given that it was the only method to correctly deduce the shape of the right side of the function and it produced the most accurate estimate of the noise variances. The variance in its fit is too large however, figure 3.38 shows that its comparative performance with the other state space

models straddles both sides of zero — it has instances of outperforming the other models but also of under-performing. Figure 3.39 shows the worst (left) and best (right) models learnt using the particle filter, with identical training data and set up. These models have a log test probability score of -1.06 and -0.638 respectively, with the direct method getting -0.684, which lies in the middle. Increasing the number of particles may increase the consistency of the particle filter although this method is already nearly 50% slower than the direct method (table 3.5). The two EM approaches struggle with consistency for the cart and pendulum data set, both have a large number of outliers where performance was very poor. It is not completely clear why the EM methods in particular have been affected in this way.

A typical use of a dynamical system model is to simulate the system over a number of time steps. Despite this, in the literature dynamical models are rarely tested in this way — mostly they are only tested using one-step-ahead predictions. Therefore, we also use a more rigorous testing regime by using each model to iteratively predict an entire system trajectory over a set horizon. To do this we make use of the GP predictive equations for a Gaussian distributed test input 1.28. Although the resulting predictive distribution is non-Gaussian we can compute the correct moments analytically. We thus moment-match a Gaussian to the predictive distribution, which allows us to predict the next transition with the same Gaussian test point predictive equations. Iterating these steps over the trajectory length gives us a complete set of Gaussian distributions on the state at every time step. This moment-matching approach, suggested in Girard et al. (2003), will be used in chapter 4, which makes these results all the more important. The predictive distributions are evaluated against the true distribution of observations in the same way as for the one-step-ahead predictions. Each model is given the same training set consisting of a number of observed trajectories. A separate set of trajectories is used at test time. We assume that the system starts each test trajectory in a known state; each model then simulates the system over the remaining time steps. For the systems with control variables, the control policy/law is provided to the models, from which they can generate controls at each time step based on their belief about the system state.

Figure 3.40 shows the frequency with which each method achieved a particular rank on the test trajectories for the three different test systems. Better performances are indicated by larger circles at ranks closer to 1. Table 3.7 summarises the figure by listing the average rank of each method for each data set. For the one dimensional kink function, the direct method achieves the best results most consistently, with the particle EM in second place. NIGP performs the worst, which is likely to be a consequence of the linearisation failing around the sharp kink in the function. The results are more mixed for the cart and pendulum system, although the direct method still comes out on top. NIGP performs very well on this dataset, as does the particle filter, although it again is very inconsistent — regularly being either the best or the worst model. This is similar to the particle EM model, which suggests that it is a feature of the sampling approaches rather than a lower level implementation issue causing this instability. The EP-EM method has a significant drop in performance on this dataset compared to how it does on the other two, although it is not completely clear why this is. The ten dimensional unicycle dataset proves to be very difficult for the two sampling methods to model, with the particle filter performing the worst of all the models. This is strongly related to the path degeneracy problem: although we have 1000 individual samples at the end of the particles' trajectories, if we trace these back to the first time step we find that we only have ten to fifteen separate paths at that point. This is far too few to perform reliable estimates of the expectations in ten dimensions.

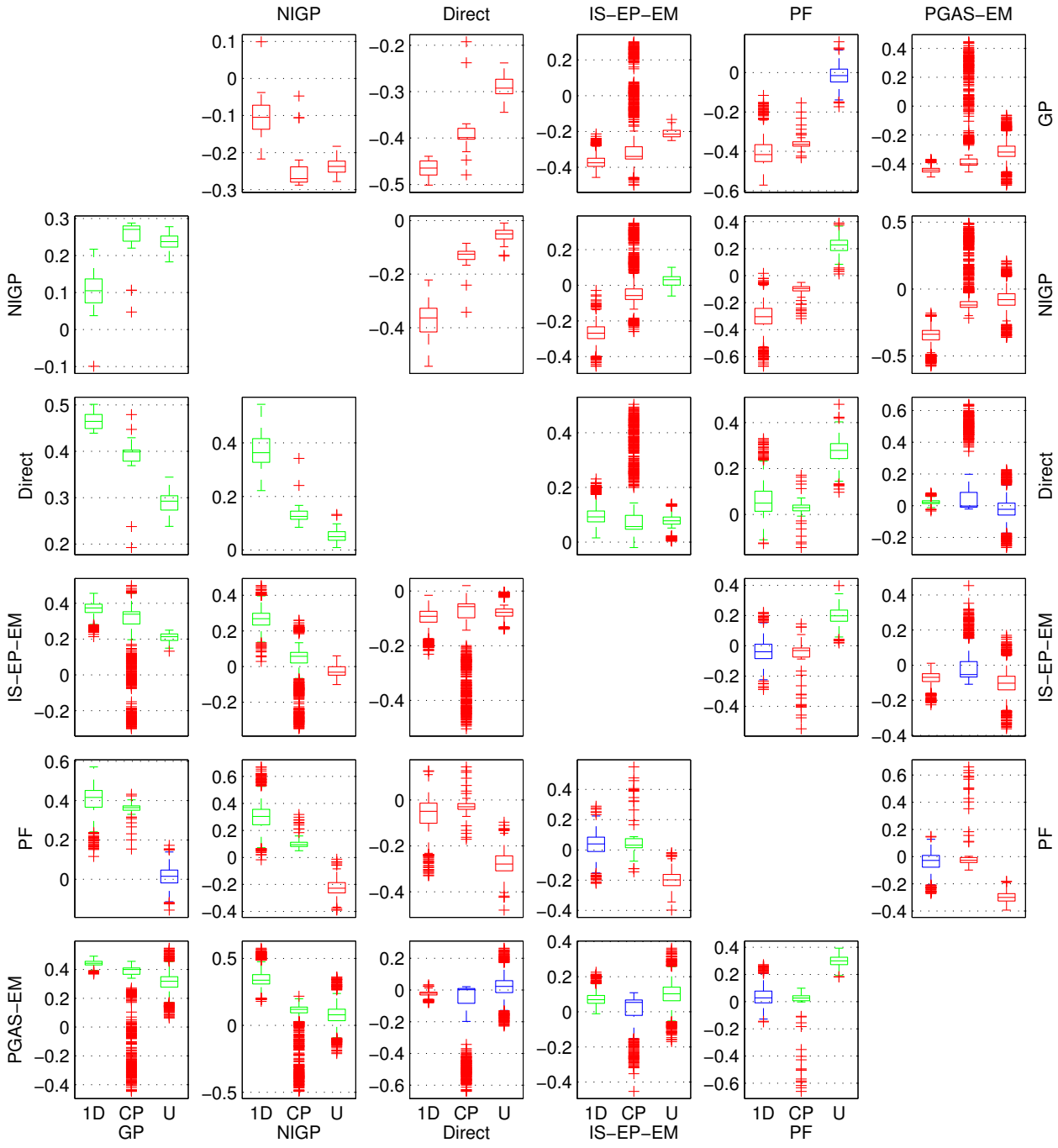


Figure 3.38: Comparative boxplots for one-step-ahead predictive performance. Each plot shows how the method listed on the row compares against the method listed on the column for the three test sets. A box plot coloured green indicates that the row method outperformed the column method in over 75% of trials, whilst a red box indicates the opposite. The test NLP values were normalised against the standard GP, such that the numbers on the y-axis indicate the difference between the approaches as a fraction of the standard GP's performance.

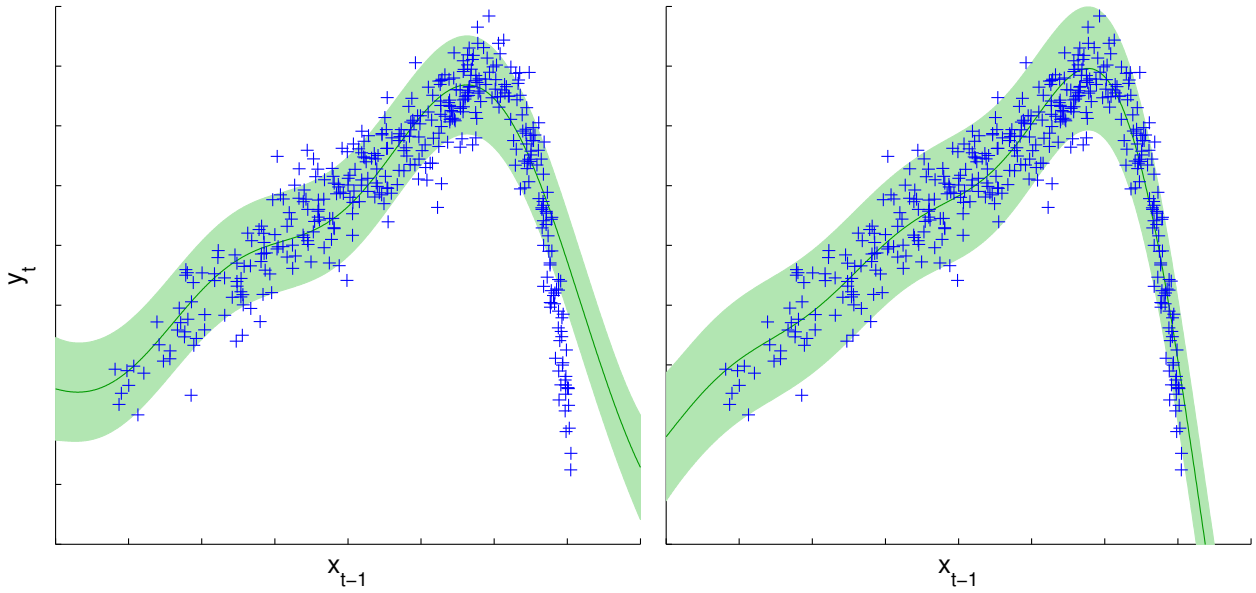


Figure 3.39: The worst (left) and best (right) models learnt by the particle filter approach using the same number of particles (5000), the same number of optimisation steps (250), and the same starting configuration. Best and worst is determined by the one-step-ahead test log probability statistic used throughout this chapter.

The direct method once again performs best, with the analytic EM approach in second place. This highlights the advantages of analytical approaches in high dimensions.

Table 3.7: The average rank (lower is better) for each method over all test trajectories on each data set.

Data set	GP	NIGP	Direct	IS-EP-EM	PF	PGAS-EM
1D	4.35	5.26	**2.00**	2.81	4.34	2.24
Cart & Pend.	4.16	3.39	**2.16**	4.12	3.57	3.60
Unicycle	4.05	3.15	**2.00**	2.63	5.03	4.14

3.10 Conclusion

In this chapter we have presented and compared a number of different approaches to learning in Gaussian Process state space models. All the state space models we investigated involve introducing a set of pseudo-points, often termed inducing points, to the GP formulation, which allowed us to treat the GP latent function values in a Bayesian manner. Apart from the 'fully Bayesian' approach (Frigola et al., 2013) all the methods we looked at use maximum likelihood optimisation to solve the GP hyperparameters. The variational approach we discussed in section 3.4.4 allowed for a Bayesian treatment of the pseudo-targets, finding a posterior distribution over them and integrating them out. Due to time constraints we were limited in our investigation of this approach, although preliminary results on the one dimensional function showed it was under-fitting. In the remainder of the chapter we treated the pseudo-targets as parameters and investigated four different methods for fitting the resulting state space model: direct gradient descent on an analytic approximation to the marginal

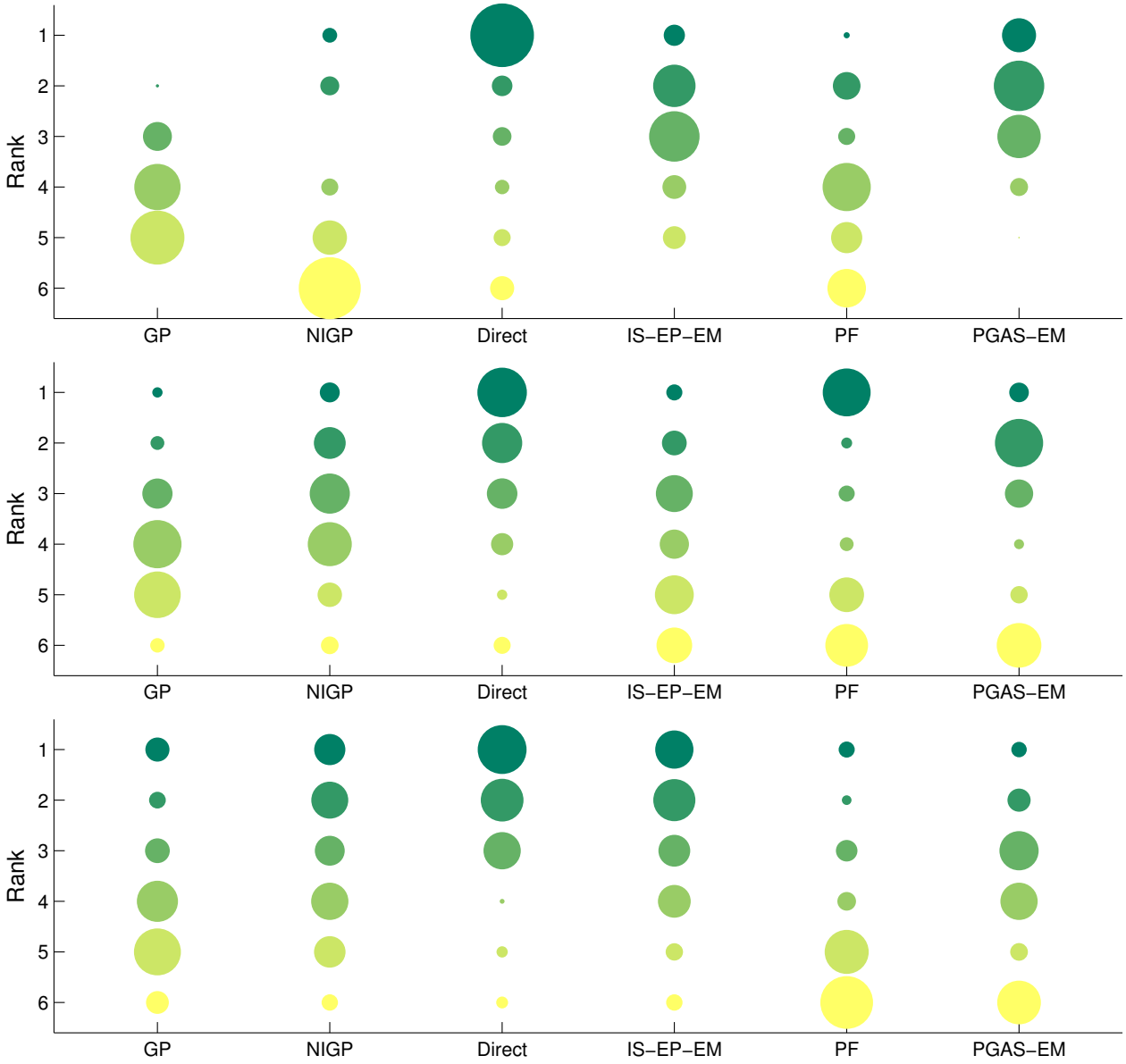


Figure 3.40: Simulation test performance on the three test data sets, in order from top to bottom: 1D function, cart & pendulum, unicycle. The circle size shows the number of times a method achieved a particular rank, relative to the other methods, over a set of 20 test trajectories and random repeats for the stochastic methods. Colour further denotes rank, thus a large green circle at rank 1 indicates good performance. See table 3.7 for average ranks.

likelihood, an approximate-analytic EM algorithm using a variety of inference methods including expectation propagation, direct gradient descent using a particle filter to estimate the derivatives of the marginal likelihood, and a particle EM method using a particle Gibbs algorithm in the E step. Each of these methods have their own strengths and weaknesses and their relative performances were found to be strongly dependent on the dataset to which they were applied.

The novel, analytic, direct optimisation method tended to underestimate the noise levels for lower dimensional, or simpler, problems. Where this is not a problem it performed consistently well and was the method which performed the best on the most number of data sets. The Gaussian moment-matching did not appear to cause a strongly detrimental effect that we could detect, although there may be other datasets where this becomes more apparent. The direct method also avoids having to

run smoothing over the latent states, which is a large advantage when compared to the EM methods. Of all the approaches we looked at, the analytic EM methods were the most complex: combining importance sampling, expectation propagation, and the EM algorithm. The introduction of sampling into the EP stage gave some significant improvements for the lower dimensional problem but caused numerous problems with numerical stability as well as a non-negligible computational cost. That said, the method performed particularly strongly on the kink function and the unicycle data set, suffering a currently unexplained drop in performance on the cart and pendulum system. Its splitting of inference and learning into two stages allowed us to solve for the pseudo-targets directly, which is a strong advantage, although this approach can also lead to slow convergence, particularly of the noise levels. However, it does make the most approximations of all the methods, and our results showed it had a very strong tendency to overfit, so care should be taken with this approach.

The particle filter can be very effective, producing the best performing models for lower dimensional systems. However, its performance is incredibly variable — it also produces the worst performing models on the same data set. It is not completely clear as to the cause of this variance, it does not seem to be just too few particles as there was no increase in performance from five thousand to ten thousand particles for the kink function (see figure 3.29). Its greatest difficulty is finding an appropriate search direction for optimisation as steepest descent leads to poor performance and the gradients are somewhat noisy, which limits the effectiveness of BFGS-style methods. It suffered a significant drop in performance for the ten dimensional unicycle system, indicating that its application is very problem specific and care must be taken with its use. It was also easily the slowest method to train (3.5), although it is easily parallelised. The particle EM approach combines the advantages of using EM, namely solving for many of the parameters, with the advantages of a particle approach (lack of approximations), while avoiding many of the disadvantages of both. The only approximation it has to make is that of using samples in the E step; these samples are then fixed in the M step, thus avoiding the gradient instability of the particle filter. Despite this the results were broadly similar to the analytic EM approach and not as strong as expected. It also proved to be fast to train, although it required a much larger number of EM steps to fit the cart and pendulum dataset than might have been expected. In section 3.8.5, we discussed an extension to this approach, the particle stochastic approximate EM algorithm (Lindsten, 2013b), which we did not compare to due to time restrictions but which may well improve results.

As well as introducing a powerful novel method of modelling GP state space models, we have demonstrated some of the strengths and weaknesses of a broad range of modelling approaches, something often lacking in the literature. Although the direct method produced the best results in our experiments, there is enough variability to suggest other methods may perform better on different data. Thus this chapter shows the benefits of a careful choice of a modelling strategy to suit the system of interest, rather than a single beats-all method.

Although the comparisons in this chapter have been somewhat lengthy, they are far from the definitive word on these models, and there is much still to be investigated. In particular we would like to extend the analysis of the variational approach and test it on the higher dimensional datasets, and run the Bayesian method of Frigola et al. (2013) on the test datasets. All the methods need an overhaul in their implementation to provide computational gains and we need to investigate the numerical stability and overfitting issues in the approximate-analytic EM approach. Given the power and general applicability of Gaussian Process state space models we expect this to be an important and fruitful area of research

for many years to come.

Chapter 4

Machine Learning for Control

4.1 Chapter Overview

In this chapter we look at a control policy learning framework developed over the last few years to optimise a control policy for a plant based purely on measurements. This framework has been published under the name of 'PILCO' (Probabilistic Inference for Learning COntrol) (Deisenroth and Rasmussen, 2011; Rasmussen and Deisenroth, 2008; Deisenroth et al., 2014; Deisenroth, 2009). We highlight a number of weaknesses in the framework as published, particularly when the system to be controlled has even a moderate level of observation noise. We then apply a GP state space model method from chapter 3 and show that it vastly improves the control learning algorithm. We also investigate two further aspects of the framework and show how these could be advanced in future research to bring additional improvements to the framework.

The main contribution of this chapter is improving PILCO to work on systems with significant observation noise, a common situation it could not tackle adequately before. The PILCO framework has been developed in collaboration with a number of others, however the work presented from section 4.5 onwards is entirely original.

4.2 Background

The field of control theory is often said to have started with James Clerk Maxwell's 1868 paper 'On Governers' (Maxwell, 1867). Since then the design and implementation of control systems has become essential to our everyday lives: manufacturing and mass production, power generation and distribution, transport, communications, and computing, among many, many other applications, all rely on numerous controllers. As our processes become ever more complex the task of designing satisfactory controllers becomes increasingly difficult. The vast majority of controller design techniques in the control field revolve around the mathematical analysis of a parametric model of the plant to be controlled. This model is most commonly built from first principles and knowledge of relevant laws and equations describing the plant dynamics, which represents a significant barrier to overcome before controller design can even start. An incredible amount of time and effort is invested in building increasingly complicated models of modern processes in order to facilitate controller design.

Once a model has been built, common controller design methods include loop-shaping and 'optimal' control methods such as the linear quadratic Gaussian regulator (LQG), or model predictive control (MPC) (Maciejowski, 2002). In 'optimal' control, the control policy is designed to minimise a specified loss function given the model of the plant's behaviour. For example the time-invariant, discrete-time, linear quadratic Gaussian regulator assumes a linear model of the form,

$$\begin{aligned} \boldsymbol{x}_t &= A\,\boldsymbol{x}_{t-1} + B_{t-1}\,\boldsymbol{u}_{t-1} + \boldsymbol{\epsilon}_t \\ \boldsymbol{y}_t &= C\,\boldsymbol{x}_{t-1} + \boldsymbol{\nu}_t \end{aligned} \tag{4.1}$$

where $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\nu}_t$ are Gaussian noise vectors. It then finds a controller, $\boldsymbol{u}_t = -L\,\hat{\boldsymbol{x}}_t$, where $\hat{\boldsymbol{x}}_t$ is the Kalman filtered state, which minimises the quadratic loss,

$$J = \mathbb{E}\left[\boldsymbol{x}_T^T F\,\boldsymbol{x}_T + \sum_{t=1}^{T-1} \boldsymbol{x}_t^T Q\,\boldsymbol{x}_t + \boldsymbol{u}_t^T R\,\boldsymbol{u}_t\right] \tag{4.2}$$

If the plant's dynamics are perfectly described by equation 4.1, the true loss is quadratic, and there are no constraints or restrictions on either the state or the controls then the controller so designed is optimal. As one might expect, these criteria are rarely met for anything other than very simplistic systems.

Model predictive control is a more complex form of 'optimal' control, introduced in particular to handle constraints and nonlinear dynamical systems. Once again a particular parametric model for the plant is assumed but this time it is used to predict the system trajectory from the current state for a set number of time steps into the future, termed 'receding horizon'. The control actions at each time step are then optimised to reduce a specified loss function. Once the optimisation has finished the resulting control action for the current time step is actuated and then the optimisation begins again at the next time step with new measurements. The optimised controls for future time steps are discarded, or used to initialise optimisation at the next step. As the optimisation must run online within one time step there is a computational limitation in the complexity of the dynamical model used and in the simulation horizon length. With modern computing power however, MPC has been successfully implemented in industry for control of complex plants.

A weakness of all the above methods is that they require a model of the plant to be pre-specified and then use this model to design the controller. Designing these models often requires significant domain knowledge to understand the underlying principles of the plant's dynamics and thus model them. Even with such knowledge the almost fractal-like complexity of real systems necessitates making modelling assumptions and idealisations. For example, consider the simple cart and pendulum system (see section 1.7.1) we can model the high level dynamics simply enough with Newtonian mechanics. However, we could then model the dynamics of the motor and its shaft, the response of the cart sliding along the rail, the flexibility in the cable attached to the cart and in the pendulum arm, or the effects of friction and 'stiction'. On top of this, will usually assume the pendulum, mass, and cart are each of a uniform density and are attached to one another perfectly symmetrically. All of these assumptions can be wrong and these errors will cause the true plant to behave differently to the model. In fact, it will rarely be the case that the model captures the true dynamics of the plant and these model errors can cause severe problems — controllers which successfully stabilise the model might not stabilise the real plant. The area of robust control attempts to reduce these problems by considering disturbances to the state and perturbations to the model and ensuring stability for a certain size of model error.

The weakness with this approach is that it usually only considers errors with a certain form or that are within the model class, i.e. if the model is linear then a controller might be found to stabilise the system,

$$\boldsymbol{x}_t \;=\; A\,\boldsymbol{x}_{t-1} \;+\; B\,\boldsymbol{u}_{t-1} \;+\; \boldsymbol{\epsilon}_t \;+\; F\,\boldsymbol{w}_t \tag{4.3}$$

where $\boldsymbol{w}_t$ is some additional noise which can be correlated with the state and F is how this noise is assumed to affect the dynamics. Alternatively we can consider the structured error

$$\boldsymbol{x}_t \;=\; (A \;+\; \Delta_A)\,\boldsymbol{x}_{t-1} \;+\; (B \;+\; \Delta_B)\,\boldsymbol{u}_{t-1} \tag{4.4}$$

however, if the true dynamics are nonlinear then the size of perturbations Δ_A and Δ_B required to capture the true plant is likely to be very large. For example, consider figure 4.1, which shows a true plant transition function and a linear model of it. By eye the model looks close to the true plant's dynamics, however, the right hand plot shows a trajectory from each of these transition functions which are completely different. In order to capture all possible true trajectories within this input space by perturbing the offset term in the linear model we would have to consider all linear models within the dashed red bounds. This can lead to a very conservative controller.

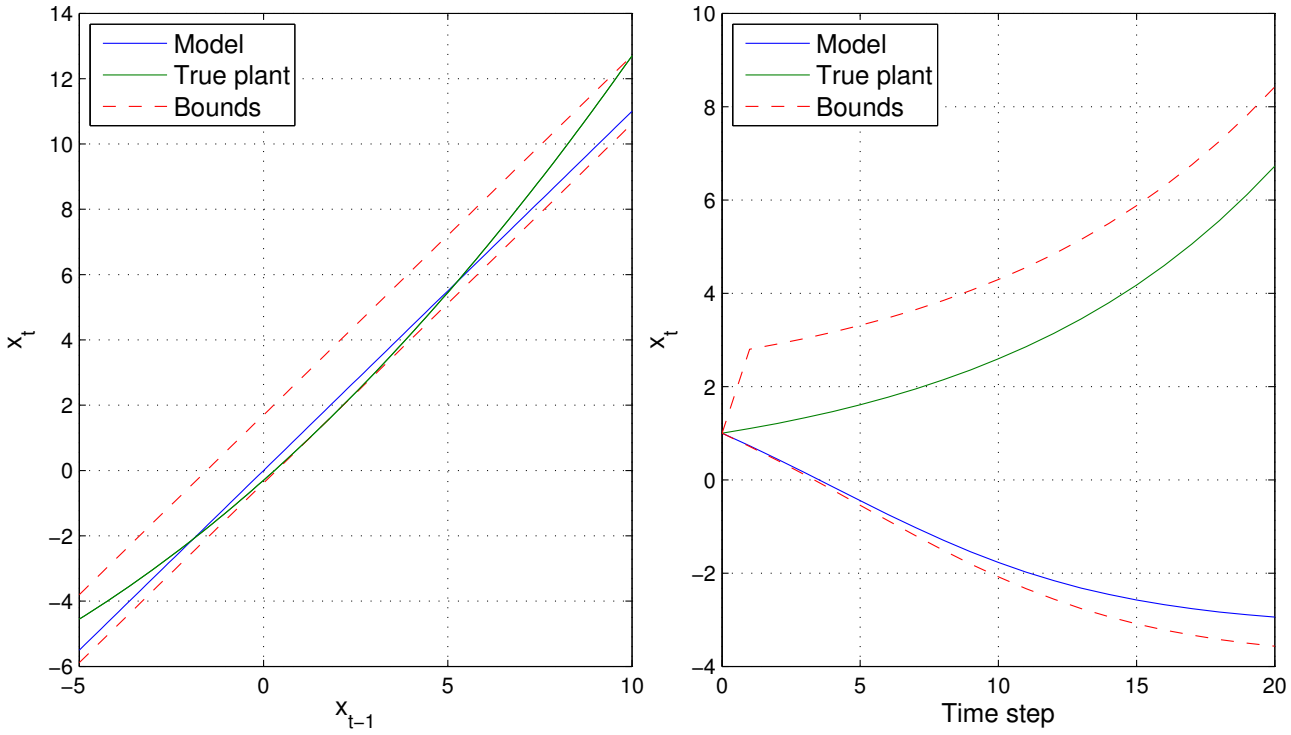


Figure 4.1: Illustration of the effect of model errors on trajectory prediction. The left plot shows the true nonlinear transition function in green and the linear model of the plant in blue. Whilst these look visually similar they lead to very different trajectories as shown in the right hand plot. In order to capture the true trajectory in a predicted set by considering perturbations to the linear model's offset parameter we would need to use all models within the red dashed lines.

Model predictive control has some inherent robustness as the controls to be applied are optimised in an online fashion, at each time step, and are based on the most recently acquired observations. Model errors have the largest effect over a long time horizon where they are compounded by making multiple predictions from the model. MPC only requires the model to be accurate for a short time horizon and

thus it can avoid the difficulties that an offline designed controller suffers from. However, model errors can still cause its performance to degrade and it should be noted that its claim to be an 'optimal' control method is based on the accuracy of its dynamical model.

A different approach is that of adaptive control (Landau et al., 2011), which seeks to adapt either the control policy or the plant model online in order to improve performance. Adaptive control methods can be broadly split into two approaches: indirect and direct. Direct, or model-free, methods attempt to adapt the control policy directly as actions are taken and new measurements are made (Kaufman et al., 1998). The key method in direct adaptive control is Model Reference Adaptive Control (Whitaker et al., 1958; Parks, 1966), which attempts to adjust policy parameters to reduce the error between the actual plant output and a desired output. Whilst direct adaptive control methods have been applied to numerous industrial applications, they cannot always be used: for example, many of the elegant theorems are based upon the use of a minimum-phase linear model of the plant, which can be very restrictive (Landau et al., 2011). Indirect adaptive control attempts to adjust the parameters of the dynamical model as measurements are made in order to better capture the plant's dynamics. This idea has been around for a long time, for example an early reference is Kalman (1958). A popular approach for many years is that of self-tuning regulators (Åström and Wittenmark, 1973); these adjust the model's parameters so as to minimise a predictive metric (e.g. the squared distance between predicted and observed trajectories) and then redesign the controller based on a regularisation scheme such as LQG or minimum-variance control. In all these adaptive schemes there is a trade off between controlling the plant satisfactorily and exploring the state space so as to improve the dynamical model and advance the control policy, this is often termed the exploitation-exploration trade-off. In the control field, this trade-off was formalised with the introduction of dual control theory (Feldbaum, 1960). Dual control methods are often intractable to apply directly and thus rely on approximate approaches.

The field of adaptive control is strongly related to the field of reinforcement learning (Sutton et al., 1992; Sutton and Barto, 1998). Like adaptive control, reinforcement learning can be broadly split into two categories: model-free and model-based learning. In model-free learning we attempt to learn a control strategy without building a model of the state dynamics. This is most commonly achieved by instead modelling the value function, which maps from a state (or state-action pair) to the long-term loss that would result from being in that state (and taking that action). There is a distinction here between the immediate loss (or reward) that might be gained from being in a state and the cumulative, long-term value that results from being in a state at the current time. It is this longer term value which is captured by the value function. The most well known model-free method is that of temporal difference (TD) learning (Sutton, 1988), made famous by the successful backgammon playing algorithm 'TD-Gammon' (Tesauro, 1995). These approaches can be very successful for complex, branching trajectories such as those found in backgammon but where the total number of states and actions is manageable. However, for very large state and input spaces it can become extremely difficult to learn the value function without requiring inordinate amounts of data and computation time (tens or hundreds of thousands of trials are not uncommon). In such cases model-based learning can be more effective (Atkeson and Santamaria, 1997). Model-based learning generally requires more offline computation as a dynamical model must be learnt as well as a policy, however this (usually) allows effective policies to be learnt from fewer interactions with the plant.

The success or failure of model-based learning is strongly dependent on the accuracy of the model used

to capture the plant's dynamics. If this is chosen to be a deterministic model then we once again suffer from the model-errors problem illustrated in figure 4.1 — the model is biased and thus resulting control policies can perform poorly. It is therefore essential that any model trained on data is able to represent its uncertainty rather than just making a prediction. This strongly suggests the use of a probabilistic model inside model-based methods. The strengths of this approach were emphatically demonstrated when it was used to learn to fly a helicopter through numerous acrobatic manoeuvres (Kim et al., 2003; Ng et al., 2006). Two proposed approaches for policy learning, which both use stochastic nonlinear models, are PEGASUS (Ng and Jordan, 2000) and PILCO (Deisenroth and Rasmussen, 2011). In PEGASUS, a number of transition functions are sampled from the stochastic model and used to predict future trajectories under the influence of a control policy. This policy is then optimised to produce the lowest cost, averaging over the trajectories. PILCO uses Gaussian Processes to model the dynamics and builds a Gaussian distribution over the predicted state trajectories before optimising the control policy to find the minimum expected cost.

In this chapter we examine the PILCO framework and build upon it to improve some of its key weaknesses, particularly in the area of observation noise.

4.3 The Control Learning Framework

In this section we describe the PILCO control learning framework. The core components of the framework are:

1. **The system state**, $\boldsymbol{x}$ In classical control theory, the system state is defined to be the set of system variables which renders the transitions Markov i.e. those variables which capture all the historical information necessary to determine the system's future behaviour. If we already have a mathematical model of the system's dynamics then, in most cases, the state can be derived from the properties of this model. However, in the machine learning approach we do not have such a model available and are unlikely to know which variables we need to include in the state (there might also be prohibitively many of them). We therefore use a less precise definition: the system state is the collection of variables required to predict the system's evolution to within a required accuracy, in union with the variables required to compute the control action(s), and to compute the loss. Even once we have chosen a set of variables to be the state, in most cases we won't know their values exactly and so we will represent our belief over the state's value with a Gaussian distribution,

$$\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_{x_t}, \Sigma_{x_t}) \tag{4.5}$$

2. **The loss function**, $L(\boldsymbol{x})$ The loss function contains the goal of the control task and returns a value depending on how far away the current state is from that goal. Given that the system state can be uncertain we require that the loss function be chosen such that we can compute the expectation over the state in closed form,

$$\bar{L}_t = \mathbb{E}_{x_t \sim \mathcal{N}(\mu_{x_t}, \Sigma_{x_t})}[L(\boldsymbol{x}_t)] \tag{4.6}$$

We also require that the loss function be differentiable so that we can optimise the control policy parameters via gradient descent. Suitable loss functions include linear, quadratic, a saturating

loss function such as $1-\exp(-(\boldsymbol{x}_t-\boldsymbol{z})^T W (\boldsymbol{x}_t-\boldsymbol{z}))$ (where $\boldsymbol{z}$ is the target state and W is a weight matrix), RBFs or, for a one dimensional state, a hinge loss function. The exact choice of loss function form will depend on the problem being solved: for example, a regularisation problem (such as the unicycle discussed later) is suited to the saturating loss function centred on the target state; whereas a maximisation-type control problem, such as trying to drive a car as fast as possible, would be suited to a linear loss function on speed.

The above loss function is placed on the system state, $\boldsymbol{x}$, however we can also place loss functions on the applied control actions or the parameters of the control policy. We can also introduce an exploration-exploitation trade-off by adding a term based on the uncertainty, $\mathbb{V}_{x_t\sim\mathcal{N}(\mu_{x_t},\Sigma_{x_t})}[L(\boldsymbol{x}_t)]$. This is discussed in more detail in section 4.4.

3. **The control policy**, $\pi(\boldsymbol{x}, \boldsymbol{\theta})$ The control policy maps the current state, $\boldsymbol{x}_t$ to the control action to be applied, $\boldsymbol{u}_t$. It can take the form of any parametric function, linear or nonlinear, providing that we can compute in closed form the output moments for a Gaussian distributed state vector,

$$\boldsymbol{\mu}_{u_t} \;=\; \mathbb{E}_{x_t}\left[\pi(\boldsymbol{x}_t,\boldsymbol{\theta})\right], \qquad \Sigma_{u_t} \;=\; \mathbb{V}_{x_t}\left[\pi(\boldsymbol{x}_t,\boldsymbol{\theta})\right] \tag{4.7}$$

and the mapping is differentiable. The goal of the entire framework is to find the setting of the control policy parameters $\boldsymbol{\theta}$, which minimises the sum of the expected losses over a fixed time horizon, T,

$$\boldsymbol{\theta}^* \;=\; \arg\min \sum_{t=1}^{T} \mathbb{E}_{x_t\sim\mathcal{N}(\mu_{x_t},\Sigma_{x_t})}\left[L(\boldsymbol{x}_t)\right] \tag{4.8}$$

Typically we use either a linear control policy, a Gaussian RBF, or a mixture of the two, although there are numerous other suitable forms.

4. **The dynamical model**, $f(\boldsymbol{x}, \boldsymbol{u})$ The dynamical model is used to simulate the plant to allow a control policy's effectiveness to be evaluated without needing to interact with the real system. The dynamical model maps from the current state and an applied control action to the state at one time step later. Complete predicted trajectories can be found by chaining several such predictions together. As we are using a data-driven approach we fit the dynamical model directly from the observed data, although this does not preclude us from encoding prior knowledge of the plant into the model. The dynamical model needs to satisfy a number of requirements; it must be able to:

   - *Handle nonlinear dynamics* — the motivation behind this approach is to be able to find controllers for complex nonlinear systems, thus the dynamical model must be able to fit such systems.
   - *Quantify its predictive uncertainty* — as the model is fit to data it will only be accurate in the region of the observations. However, during policy optimisation (described below) the model may be required to make predictions at points far away from any observed data. If the model is not able to indicate that its resulting prediction is very uncertain then control policy training can be extremely slow or fail altogether.
   - *Make predictions with an uncertain test point* — we will nearly always be uncertain about

the exact current state (as a direct result of the above point) and so the dynamical model must be able to handle and propagate this uncertainty correctly. In most cases this means we need to be able to analytically compute,

$$\boldsymbol{\mu}_{x_t} = \mathbb{E}_{x_{t-1},u_{t-1}}\left[f(\boldsymbol{x}_{t-1},\boldsymbol{u}_{t-1})\right], \qquad \Sigma_{x_t} = \mathbb{V}_{x_{t-1},u_{t-1}}\left[f(\boldsymbol{x}_{t-1},\boldsymbol{u}_{t-1})\right] \tag{4.9}$$

- *Be differentiable* We wish to optimise the policy parameters via gradient descent and thus we must be able to compute the derivative of the next state prediction w.r.t. the input state.

A principled and powerful model which satisfies these requirements is the Gaussian Process (Rasmussen and Williams, 2006). A GP can model highly nonlinear functions, it is a stochastic model which provides Gaussian predictive distributions, with some restrictions on the allowed covariance functions we can compute predictive moments for a Gaussian distributed test point Girard et al. (2003), and the resulting predictive moments are differentiable w.r.t. the input test point (or its moments). The framework does not require the use of a GP but we have yet to find a more suitable model.

Figure 4.2 illustrates the high level steps in the control framework.

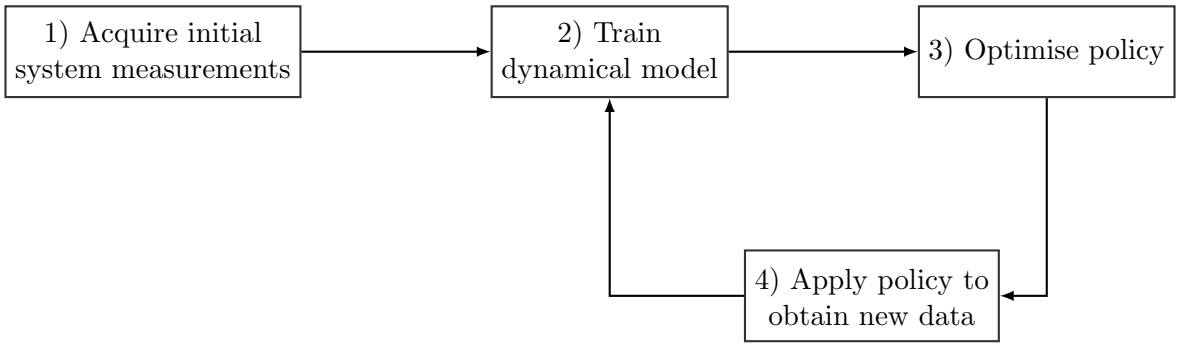


Figure 4.2: Diagram of the high level steps of the control learning framework. Once we have some initial data we can train a dynamical model and optimise the control policy. From this point we can collect more data and then iterate the procedure to improve the dynamical model and control policy.

1. **Acquire initial system measurements** We need some initial data from the system in order to start training. This data can be acquired in a number of ways: for example by application of random controls or a random control policy, or by use of an existing controller.

2. **Train dynamical model** As discussed above the model of the plant's dynamics is a key component of the framework. In this step the observed data is used to construct this dynamical model.

3. **Optimise policy** The dynamical model is used to simulate the plant's behaviour under the current control policy. The policy parameters are then updated to reduce the expected loss incurred.

4. **Apply policy to obtain new data** After the policy parameters have been optimised the new policy is applied to the plant to obtain further training data. This data is used to update the dynamical model and then find a new policy.

We now look at steps 2 and 3 in more detail.

4.3.1 Training the dynamical model

The dynamical model maps the current state and control action to the state at the next time step,

$$\boldsymbol{x}_t \;=\; f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) \;+\; \boldsymbol{\epsilon}_t \tag{4.10}$$

where $\boldsymbol{\epsilon}$ is a zero-mean Gaussian noise vector (process noise). We will model the dynamics with a Gaussian Process,

$$f \;\sim\; \mathcal{GP}(m,\; k) \tag{4.11}$$

where $m(x)$ is the GP mean function and $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the covariance function. We will mostly be modelling multi-dimensional systems, $\boldsymbol{x} \in \mathcal{R}^D$ and thus we will use D independent GPs to model the mapping from the inputs to each output dimension,

$$x_t^i \;=\; f_i(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) \qquad \text{for } i \;=\; 1 \ldots D \tag{4.12}$$

$$f_i \;\sim\; \mathcal{GP}(m_i,\; k_i) \qquad \text{for } i \;=\; 1 \ldots D \tag{4.13}$$

We will usually set the GP mean function to be the identity function such that the the relevant input variable is passed to the output,

$$m_i(x_j) \;=\; \begin{cases} 1 & \text{if} \quad i \;=\; j \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

which means that the GP models the change in state rather than the absolute state value. Providing the time step isn't too long compared to the system time constants, the current state is usually a reasonable estimate for the next state and so this choice of mean function tends to lead to better performance than using a zero mean function. As stated above, we will need to evaluate the predictive output moments of the dynamical model for the case of Gaussian input points, which limits the choice of covariance function. We will use the 'squared exponential' function in this chapter as it leads to a flexible class of models whilst keeping the mathematics simple. For future work we would like to investigate more widely the class of covariance functions we could use to see if there is a better choice — the squared exponential kernel imposes smoothness constraints which are mostly too strict for the systems we are modelling.

In real-world systems we will rarely be able to observe the true state directly. We therefore define the observations made at time t to be $\boldsymbol{y}_t$, and introduce an observation function to account for this,

$$\boldsymbol{y}_t \;=\; g(\boldsymbol{x}_t) \;+\; \boldsymbol{\nu}_t \tag{4.15}$$

where $\boldsymbol{\nu}_t$ is a zero-mean Gaussian noise vector, referred to as the observation noise. We argued in

section 3.3 that we can always define our state $\boldsymbol{x}$ such that the observation function is linear,

$$g(\boldsymbol{x}_t) \;=\; C\,\boldsymbol{x}_t \tag{4.16}$$

In fact, in all the systems we will consider in this chapter we observe all the state variables up to noise corruptions, and so we will set $C = I_D$, where I_D is the D-dimensional identity matrix. Considering systems which include completely unobserved state variables is an extension we are keen to explore in the future.

We are now faced with training a Gaussian Process dynamical model based on noisy data, which is exactly the problem considered in chapter 3. In the previously published work on this control framework the noisy observations $\boldsymbol{y}$ were simply substituted in for the latent states $\boldsymbol{x}$ when training the GP. That is, a training set was produced such that,

$$\begin{bmatrix} \boldsymbol{y}_1, \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{y}_{T-1}, \boldsymbol{u}_{T-1} \end{bmatrix} \quad \xrightarrow{\mathcal{GP}} \quad \begin{bmatrix} \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_T \end{bmatrix} \tag{4.17}$$

As was discussed in the previous two chapters, ignoring the effect of noise in the inputs of the GP can lead to very poor modelling performance. In section 4.5 we will use techniques from chapter 3 to improve the performance of the control learning framework.

4.3.2 Optimising the policy

Optimisation of the policy is based on simulation of the plant using the dynamical model. There are two compelling reasons for taking this approach,

- Using a data efficient model to simulate the true plant means that a controller can be learnt with very few interactions with the real system — something which can be costly and time consuming.
- Use of a differentiable model means that the policy can be learnt using gradient optimisation methods, which are greatly preferable to gradient-free methods. This would not be possible if we evaluated a policy by applying it to the real plant.

Simulation of the plant involves chaining together numerous one-step-ahead predictions using the trained dynamical model. We start by specifying an initial Gaussian distribution over the state,

$$\boldsymbol{x}_0 \;\sim\; \mathcal{N}(\boldsymbol{\mu}_{x_0},\; \Sigma_{x_0}) \tag{4.18}$$

This is useful for specifying a set of start states from which the controller must solve the required task. If the plant will always start in exactly the same state then this distribution can be collapsed to a point. The current setting of the control policy is used to calculate the mean and variance of the control variables to be applied along with their covariance with the state. However, we cannot simply apply the control policy to the distribution over the latent state $\boldsymbol{x}_0$ because during a real trial we do not have access to this, we only have an observation. Thus we must first find the distribution over the

observation $\boldsymbol{y}_0$, which we can do by using our estimate of the observation noise variance,

$$\boldsymbol{y}_0 \;=\; \boldsymbol{x}_0 \;+\; \boldsymbol{\nu}_0 \quad \Rightarrow \quad \boldsymbol{y}_0 \;\sim\; \mathcal{N}(\boldsymbol{\mu}_{x_0},\; \Sigma_{x_0} \;+\; \Sigma_\nu) \tag{4.19}$$

The control moments are therefore,

$$\boldsymbol{\mu}_{u_0} \;=\; \mathbb{E}_{y_0}\left[\pi(\boldsymbol{y}_0,\, \boldsymbol{\theta})\right], \qquad \Sigma_{u_0} \;=\; \mathbb{V}_{y_0}\left[\pi(\boldsymbol{y}_0,\, \boldsymbol{\theta})\right], \qquad C_{x_0,u_0} \;=\; \mathbb{C}_{x_0,u_0}[\boldsymbol{x}_0,\, \boldsymbol{u}_0] \tag{4.20}$$

If the policy is linear then the distribution over the controls $\boldsymbol{u}_0$ are Gaussian, however for a nonlinear policy this won't be the case. To maintain analytic tractability we moment-match a Gaussian to the distribution over the controls. We can thus find an approximate joint Gaussian distribution over the latent state and the controls,

$$\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{u}_0 \end{bmatrix} \;\sim\; \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{x_0} \\ \boldsymbol{\mu}_{u_0} \end{bmatrix}, \begin{bmatrix} \Sigma_{x_0} & C_{x_0,u_0} \\ C_{x_0,u_0}^T & \Sigma_{u_0} \end{bmatrix}\right) \tag{4.21}$$

We use the dynamical model to find the moments over the state at the next time step. Once again we use moment-matching to approximate the distribution over the next state with a Gaussian.

$$p(\boldsymbol{x}_1 \mid \boldsymbol{\mu}_0, \Sigma_0, \boldsymbol{\theta}) \;\approx\; \mathcal{N}(\boldsymbol{\mu}_{x_1},\; \Sigma_{x_1}) \tag{4.22}$$

We then compute the first expected loss,

$$\bar{L}_1 \;=\; \mathbb{E}_{x_1 \sim \mathcal{N}(\mu_{x_1},\, \Sigma_{x_1})}[L(\boldsymbol{x}_1)] \tag{4.23}$$

These steps are repeated until we have a complete predicted trajectory and associated losses. This process is illustrated in figure 4.3. The total loss is formed by summing the individual expected losses. At each step we also compute the derivatives and use the chain rule to find the derivative of the total loss with respect to the policy parameters. With this computed we can use gradient descent to optimise the policy parameters.

4.4 Framework Evaluation

The control learning framework described in the previous section has been applied to a number of different mechanical systems, both real and simulated, with great success. Figure 4.4 shows a series of photos from Deisenroth and Rasmussen (2011) demonstrating a trained control policy swinging-up a pendulum and holding it in an inverted position. This task cannot be achieved with a linear controller, nor with a linear dynamical model (as the pendulum moves through more than π radians).

Figure 4.5 shows a still from McHutchon and Rasmussen (2010) where the task was to balance a pitch-only unicycle (a unicycle with stabilising wheels preventing it falling over in roll). This task was completed satisfactorily with only nine policy optimisation iterations. This task was then extended to the full, unrestricted unicycle in Queiro et al. (2011) (figure 4.6) who trained a controller which stabilised the system for around two seconds. However, there were strong hardware constraints, which restricted testing on the actual system and made the task significantly more difficult. The unrestricted unicycle task has been successfully learnt in simulation, using a very complex model of the dynamics

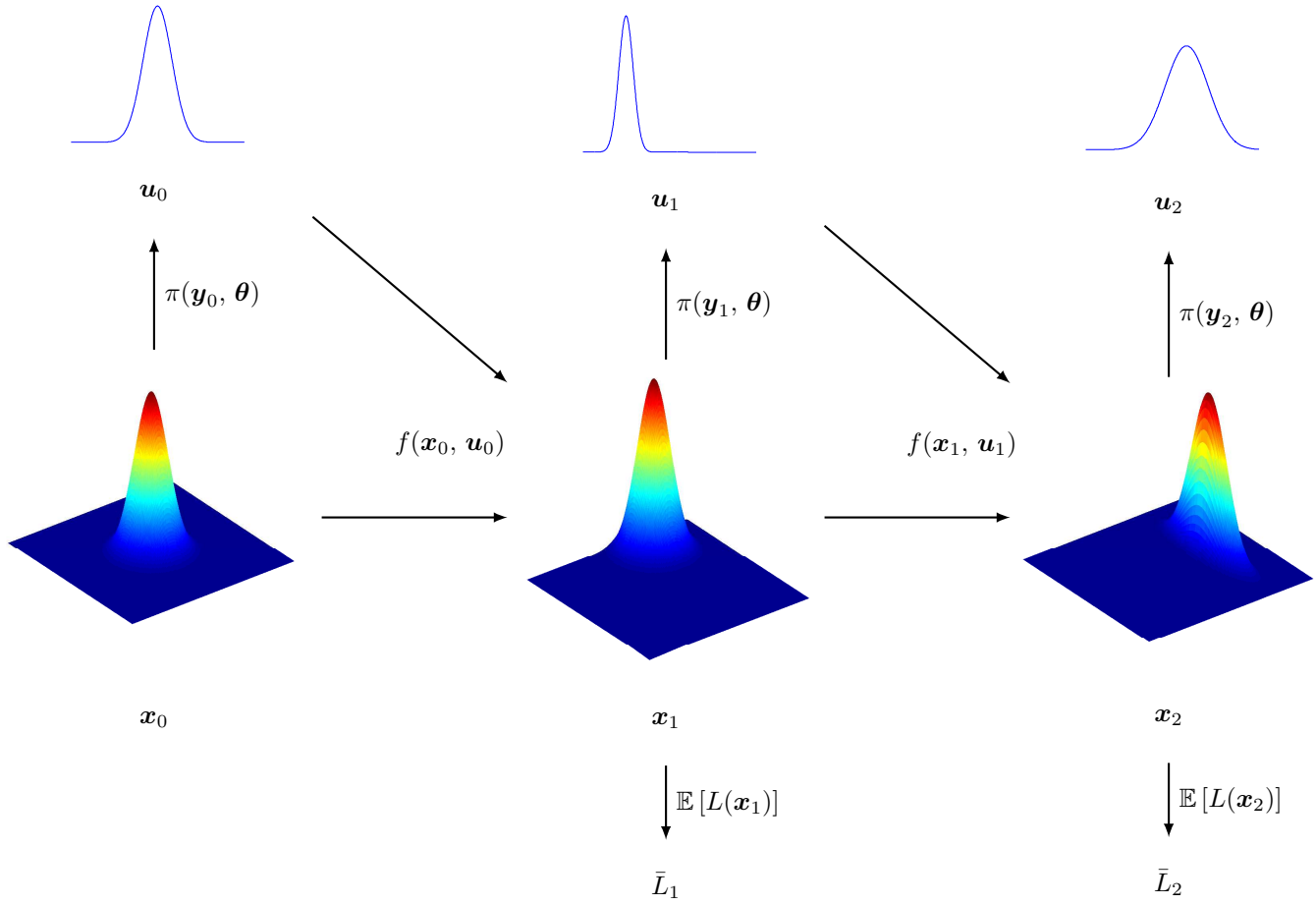


Figure 4.3: An illustration of the plant simulation step under a control policy π, for a 2D system with 1 control variable and 2 time steps.

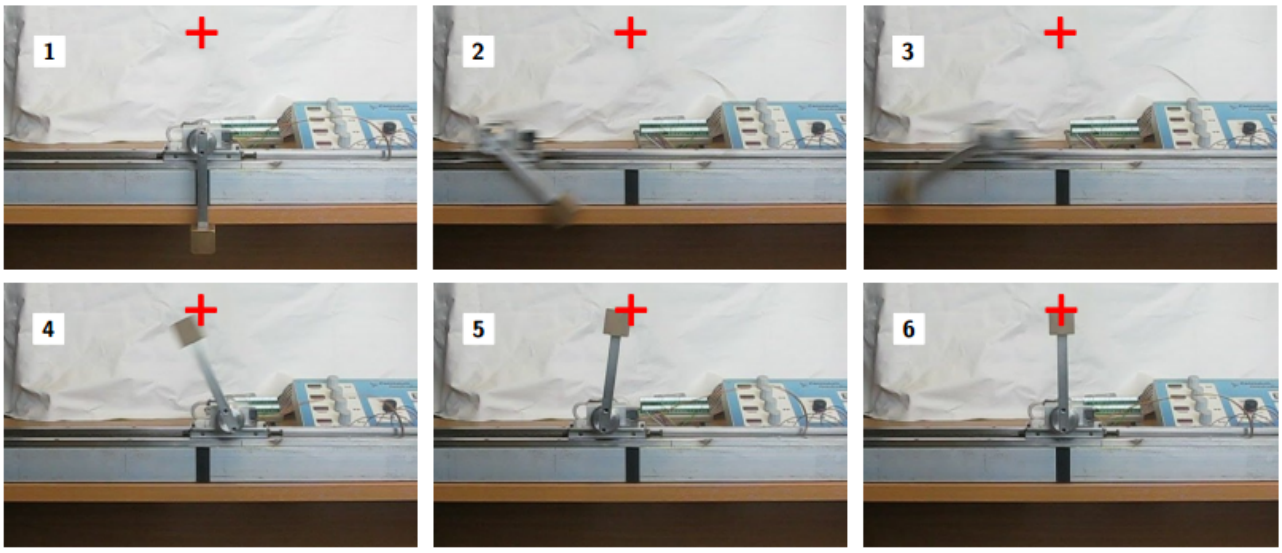


Figure 4.4: A series of stills from a video of a control policy successfully swinging up the pendulum.

as the plant.

In Bischoff et al. (2014) the learning framework was applied to a mobile robotic arm, which was

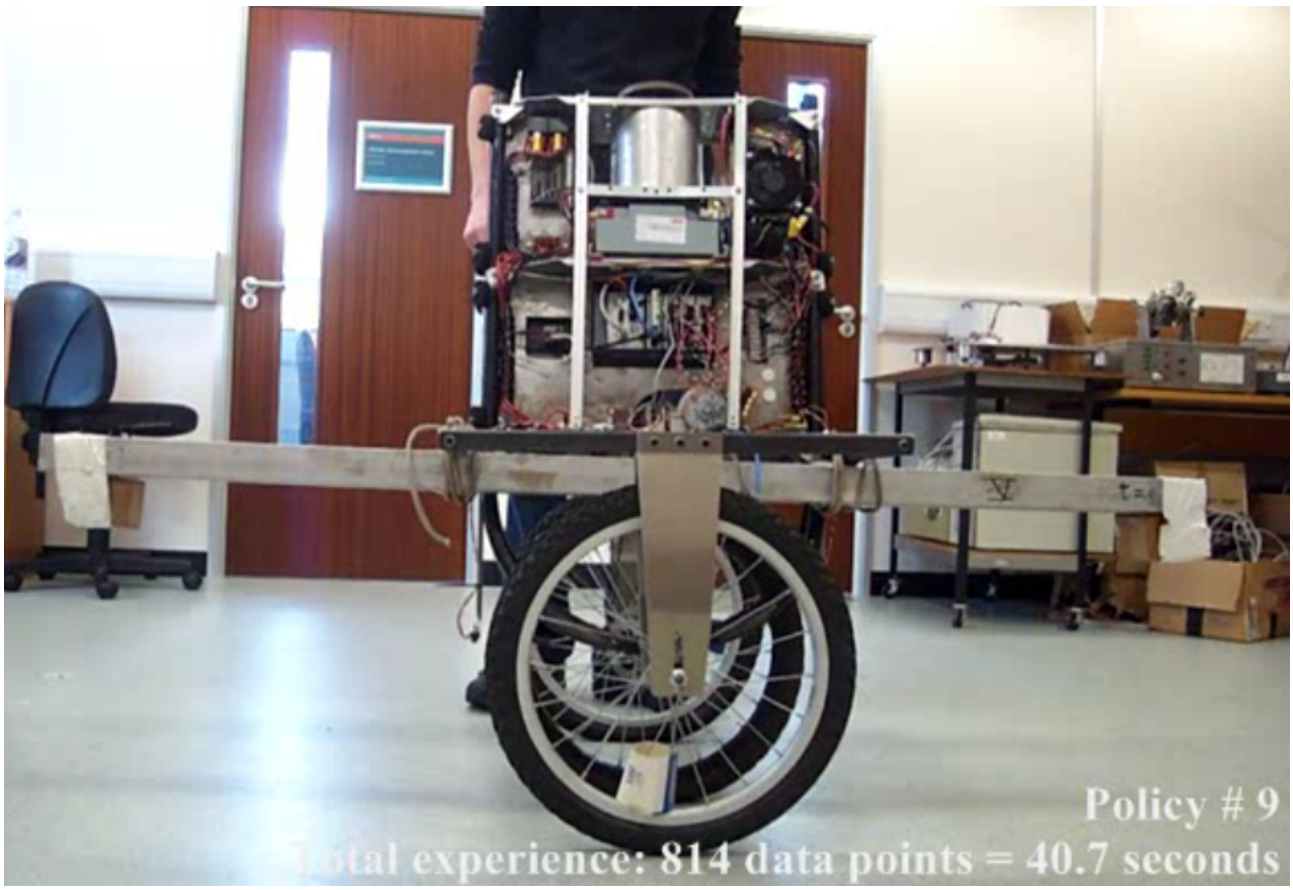


Figure 4.5: A still from a video of training performance on the restricted, pitch-only unicycle.

Figure 4.6: A still from a video of training performance on the unrestricted robotic unicycle.

required to move to an object and pick it up. Figure 4.7 shows a series of photos of the final trained policy applied to the real system. In addition to these real-world examples, the framework has been

applied to a number of simulated tasks, including cars, quad-copters, and a double pendulum version of the cart and pendulum swing-up task.

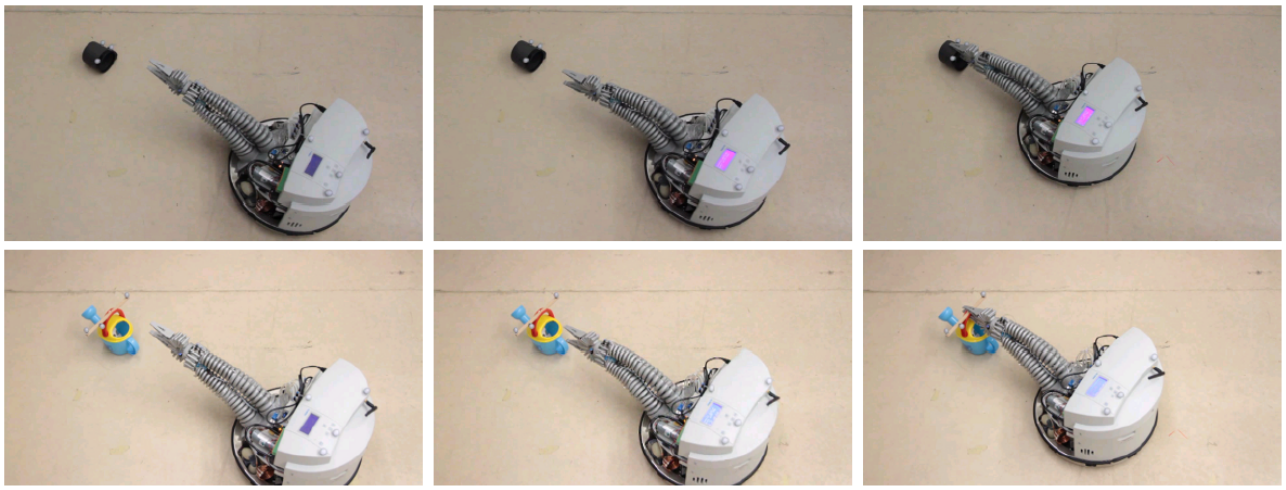


Figure 4.7: A series of photos demonstrating the optimised policy in action on the mobile robotic arm from Bischoff et al. (2014). The robot approaches the item, extends the arm and manipulates the claw so as to successfully pick up the object.

These myriad applications strongly demonstrate how powerful and approach to controller design this framework is. Despite this there are a number of areas of weakness in the framework, which could be improved upon. The first of these, which must be solved before the framework can be applied to a wide variety of real-world tasks, is the problem of observation noise. In section 4.3.1 we stated that in the published version of the framework the dynamical model was trained by fitting a GP directly to the observed transitions, from $\boldsymbol{y}_{t-1}$, $\boldsymbol{u}_{t-1}$ to $\boldsymbol{y}_t$, in an auto-regressive manner. However, as chapter 3 has shown, this can lead to a very poor dynamical model for even moderate amounts of observation noise. This weakness therefore must be addressed. To do so we propose to use methods from chapter 3 to train the GP dynamical model rather than just using simple GP-regression on the observations.

A second weakness is the Gaussian moment-matching approximations we are forced to make at each time step of the simulation stage, in order to keep the equations analytically solvable. It is hard to evaluate the effect that this approximation has. For some tasks it might seem clear than a Gaussian would be a very poor approximation to the true state distribution; for example consider the distribution the angle of a pendulum, a few time steps after it has been released from an initial inverted position. Presumably the pendulum will have fallen to either the left or the right, it is very unlikely to have remained perfectly balanced upright. Thus the true distribution on the angle would be bi-modal, but, under a Gaussian approximation (assuming symmetric dynamics) the most likely state would actually be upright. However, this analysis is complicated by the fact that the dynamics are learnt with the current controller in-the-loop, that is the actual state transitions are affected by the control signals applied. This means that, long before the controller learns to balance the pendulum, it can learn to push the pendulum always in one direction, thus collapsing the state distribution to a uni-modal distribution. In fact, the policy learning framework will favour controllers which drive the system along trajectories which are well modelled by the dynamical model.

A mixture of Gaussians could model more complex distributions but it is hard to see how to manage the transitions — each Gaussian would have to be propagated to the next time step in isolation,

which doesn't solve the problem. An alternative is to use a set of particles to approximate the state distribution at each time step. These particles can be individually propagated through the dynamical model to maintain a point cloud at every time step, which can approximate complex distributions with much greater accuracy than a simple moment-matched Gaussian. However, this approach comes with its own problems, particularly in regard to derivatives, which are now noisy. These methods, and others, are discussed in section 4.6.

A third weakness is that during the simulation stage all state transitions are considered to be independent from each other. That is, we neglect the covariance between the state at a particular time step and the dynamical model itself, which arises from previous transitions. This extra covariance is neglected because it cannot be computed analytically with a GP dynamical model. It can lead to the control framework being either over-confident or under-confident about its performance, both of which can prevent policy learning from proceeding. This area is discussed in further detail in section 4.7.

Finally, we list some other areas of potential weakness, which we do not discuss in any further detail here but which may be fruitful areas for future research.

- **Exploration–exploitation** Currently policy optimisation proceeds greedily — the framework attempts to find the control policy which leads to the lowest expected loss given the current model of the plant dynamics. However, if we know we have a budget of I policy training iterations then the optimal tactic may not be greedy optimisation, but rather to spend some initial iterations exploring the state-space. One simple method for doing this would be to make use of the variance of the loss and not just the expected loss at each time step. By subtracting some fraction of the variance from the expected loss the control policy would be rewarded for visiting areas of high uncertainty. The relative amounts of expected loss and loss variance included acts as a trade-off between exploitation and exploration. How should this parameter be set? Is there a better approach to encouraging exploration? These are open questions.

- **GP covariance function** In all current and past work we have used the squared exponential covariance function as it leads to analytically solvable equations. There are, however, other choices of covariance function which also lead to tractable equations, for example scale mixture of Gaussians, or Fourier based functions. The squared exponential kernel leads to transition functions which are infinitely differentiable; this is usually excessively smooth and so better performance may be gained by using a different covariance function.

- **Training time** The framework can take a long time to train, especially if large amounts of data are needed to fit an accurate dynamical model. There are various training parameters which can affect the training time but it is not clear how these should be set. For example, is it quicker to use short optimisation runs at each training iteration, which are quick but may we may then need more training iterations in total, or to use long optimisation runs in the hope that we need fewer iterations. Similar questions surround parameters such as the number of pseudo-points we use in the dynamical model and how long we spend training this model compared to optimising the policy. Are there further, or better, approximations we can make to speed up training at little cost to performance? Can we make better use of the increasingly parallel architecture of modern computers?

- **Time discretisation** Currently, for each task we choose a time step that roughly co-

ordinates with the time constants observed in the dynamics of the system in question. Longer time steps are usually to be preferred as this means we need fewer transition steps to simulate a trajectory of a particular length, and that we collect a smaller amount of data for each test trajectory. Both of these factors play a significant role in training time. However, if we set the time step to be too long then the framework cannot learn a controller, either because the dynamics are too complex on such a time scale, the observed states are too noisy, or the current zero-order-hold control methodology is too simplistic: if the controller could vary the applied control signal within a time step could we use a lower sampling frequency?

4.5 Handling Observation Noise

Observation noise causes two problems in the control learning framework. The first of these is in learning the dynamical model as discussed previously. The second of these occurs when we compute the control values using the policy, which is applied to the observed values. This can inject a large amount of noise into the controlled system. For example, for a one control variable, linear policy with $D \times 1$ weight vector $\boldsymbol{w}$, and an observation noise variance Σ_ν the controls will have a noise variance of $\boldsymbol{w}^T \Sigma_\nu \boldsymbol{w}$, potentially magnifying the effect of the observation noise. This effect is classically overcome by the use of a filter running inside the control feedback loop. Given that we have trained a model of the plant dynamics we can extend the framework to include a filter. However, this is not as straightforward as one might assume as, if we want to see the full effect, we cannot just include the filter during the trials on the real plant, we must also include it during the simulation state when optimising the policy. If we don't include it in this stage then the policy will not be able to take advantage of the filter, it is even possible that performance could be degraded as the policy would be applied to the real plant with an extra component that was not present during training. Including the filter in the simulation state is made difficult by the fact that we don't have any observations available, we must use distributions on $\boldsymbol{y}_t$ based on our belief over the latent state and the observation noise. Combining these distributions correctly requires several careful steps of probability mathematics, but can be done analytically. This is ongoing work and we do not report any further on the filter here, instead we focus on improving the dynamical model learning.

We could potentially use any of the methods discussed in chapter 3 to replace the simple GP regression model in the framework. We will be dealing with relatively high dimensional systems, $D \sim 10$, which suggests we should avoid the sampling based methods. We therefore choose the 'direct' method from section 3.5, although in the future we aim to test other methods, particularly the variational approach. We can take the 'direct' method as described in chapter 3 and immediately apply it to the control learning framework without any further modifications. We test the learning framework with this state space model in place on the cart and pendulum swing-up and the unicycle balancing tasks, using differential equations to simulate both real systems so that we can make multiple runs quickly. Tables 4.1 and 4.2 show the settings of various training parameters for the tasks. We ran ten complete policy training runs using different random seeds. After each iteration the current policy was tested ten times from different start states. As a performance metric we used the total loss for each of the test trials over the ten repeats and the ten different training runs.

Figure 4.8 shows the results for the cart and pendulum swing-up task for four different noise levels. The figure clearly shows a significant performance increase from using the state space model over the

Table 4.1: The settings of various training parameters for the cart and pendulum swing-up task. The velocity noise levels are set such that an Euler integration of the velocities would lead to the same noise levels as on the corresponding position or angle variable.

Parameter	Setting
Time step	0.1 seconds
Simulation horizon	5 seconds, 50 time steps
Policy function	Gaussian RBF with 50 basis functions
Loss function	saturating loss function about the stationary, inverted pendulum state
True observation noise standard deviation	
• cart position	0.01 m
• cart velocity	0.1 m/s
• pendulum angular velocity	10 degree/s
• pendulum angle	1 degree

Table 4.2: The settings of various training parameters for the unicycle balancing task. The angular velocity noise levels are set such that an Euler integration of the velocities would lead to the same noise levels as on the angles.

Parameter	Setting
Time step	0.15 seconds
Simulation horizon	10.05 seconds, 67 time steps
Policy function	linear with bias term
Loss function	saturating loss function about the zero state vector (all angles, velocities, and positions)
True observation noise standard deviation	
• roll angular velocity	6.67 degree/s
• yaw angular velocity	6.67 degree/s
• wheel angular velocity	6.67 degree/s
• pitch angular velocity	6.67 degree/s
• turntable angular velocity	6.67 degree/s
• x position	0.01 m
• y position	0.01 m
• roll angle	1 degree
• yaw angle	1 degree
• pitch angle	1 degree

simple GP regression model used previously. Figure 4.9 shows the results for the unicycle balancing task where the differences are even more marked. For all four noise levels the simple GP dynamical model does not satisfactorily solve the task, for the two higher noise levels the framework never makes any noticeable progress. With the state space model, however, the framework is able to learn a control policy which can stabilise a significant proportion of the trials for the first three noise levels and makes clear progress for the highest noise level. It seems likely that the observation noise is still having a strong effect on the system via the control policy as discussed earlier, resulting in the controller being unable to stabilise a number of trajectories, as seen in figure 4.9. It is therefore strongly desirable to implement a filter-in-the-loop to tackle this problem.

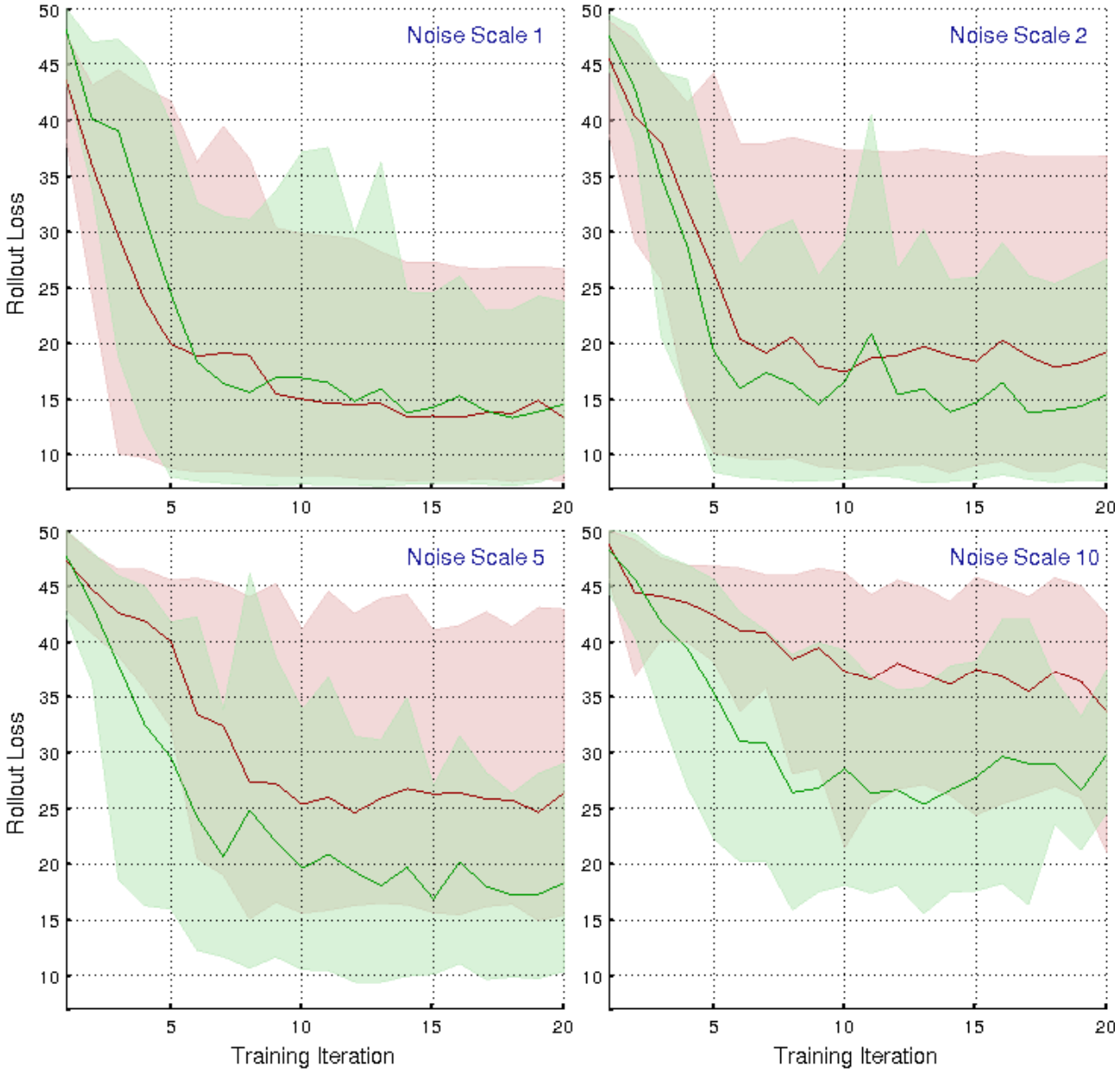


Figure 4.8: Training performance on the cart and pendulum swing-up task with the simple (non-state space) GP dynamical model in red, and the 'direct' GP state space model in green. The x axis shows policy training iteration number and the y axis shows the total loss over a 20 step trial when the policy was applied to the plant. The solid lines show the mean performance over ten different training runs and ten different trials for each run, the shaded region shows the 10 - 90% interval. The four panels show the results for four different observation noise levels: the noise variances were scaled by the factor indicated in the top right hand corner of each plot. Thus the bottom right plot has ten times the observation noise variance as the top left plot.

4.6 Simulation with Particles

In the control learning framework as described at the beginning of this section we moment-match Gaussian distributions to the true distribution over the state at each time step in order to maintain tractability. An alternative is to use a particle simulation approach, and approximate the state

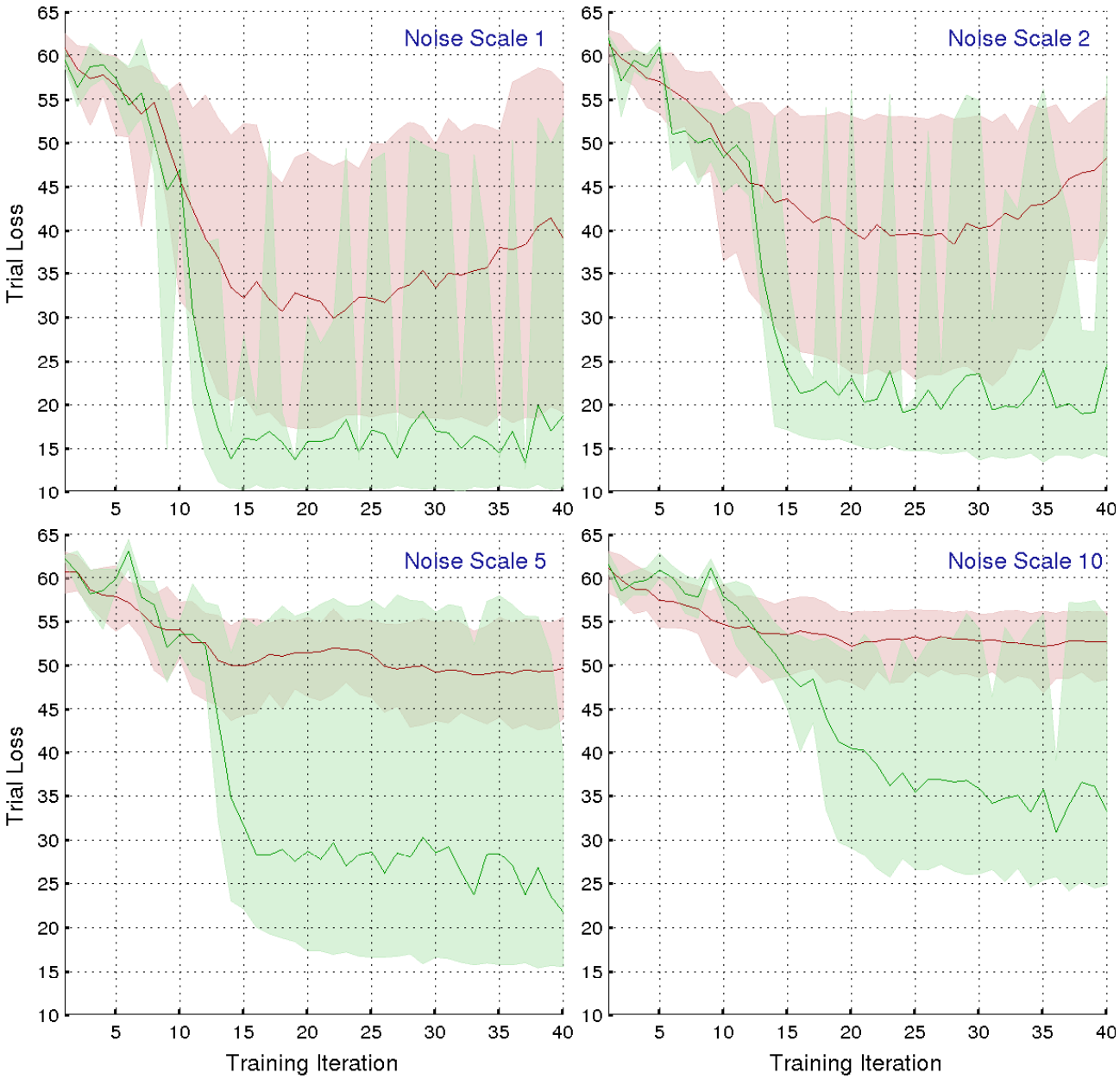


Figure 4.9: Training performance on the unicycle balancing task with the simple (non-state space) GP dynamical model in red, and the 'direct' GP state space model in green. The x axis shows policy training iteration number and the y axis shows the total loss over a 67 step trial when the policy was applied to the plant. The solid lines show the mean performance over ten different training runs and ten different trials for each run, the shaded region shows the 10 - 90% interval. The four panels show the results for four different observation noise levels: the noise variances were scaled by the factor indicated in the top right hand corner of each plot. Thus the bottom right plot has ten times the observation noise variance as the top left plot.

distribution by a set of delta functions at the sample locations,

$$p(\boldsymbol{x}_t \mid \mu_{x_0}, \Sigma_{x_0}, \pi, \boldsymbol{\theta}) \;=\; \sum_{i=1}^{N} \delta(\boldsymbol{x}_t^i) \tag{4.24}$$

Trajectories can be simulated as follows,

1. Sample from the initial distribution, $\{\boldsymbol{x}_0^i\}_{i=1}^N \;\sim\; \mathcal{N}(\boldsymbol{\mu}_{x_0},\; \Sigma_{x_0})$

2. Compute controls for each sample, $\{\boldsymbol{u}_0^i\}_{i=1}^N = \pi\left(\{\boldsymbol{x}_0^i\}_{i=1}^N, \boldsymbol{\theta}\right)$

3. Make a standard GP prediction at each sample to find the set of moments $\{\boldsymbol{\mu}_{x_1}^i, \Sigma_{x_1}^i\}_{i=1}^N$

4. Draw samples of the next state from the predictive distributions, $\left\{\boldsymbol{x}_1^i \sim \mathcal{N}(\boldsymbol{\mu}_{x_1}^i, \Sigma_{x_1}^i)\right\}_{i=1}^N$

5. Return to step 2 and repeat for the next time step until desired trajectory length has been reached.

Figure 4.10 shows a histogram of the pendulum angle state variable for the cart and pendulum balancing system over twenty time steps starting at thirty different positions near the inverted position. The figure shows strongly non-Gaussian distributions as the pendulum begins to fall either to the left or to the right. The red distributions plotted on top of the histograms show the predicted state distribution using the moment-matching simulation approach. Likewise, in green we show a kernel-smoothed density estimate based on the sampling approach. It is clear to see that the particles do a significantly better job of simulating the system. The bimodal structure of the true distributions forces the moment-matching Gaussians to have far too much mass beyond the range of the data and from this point it cannot recover.

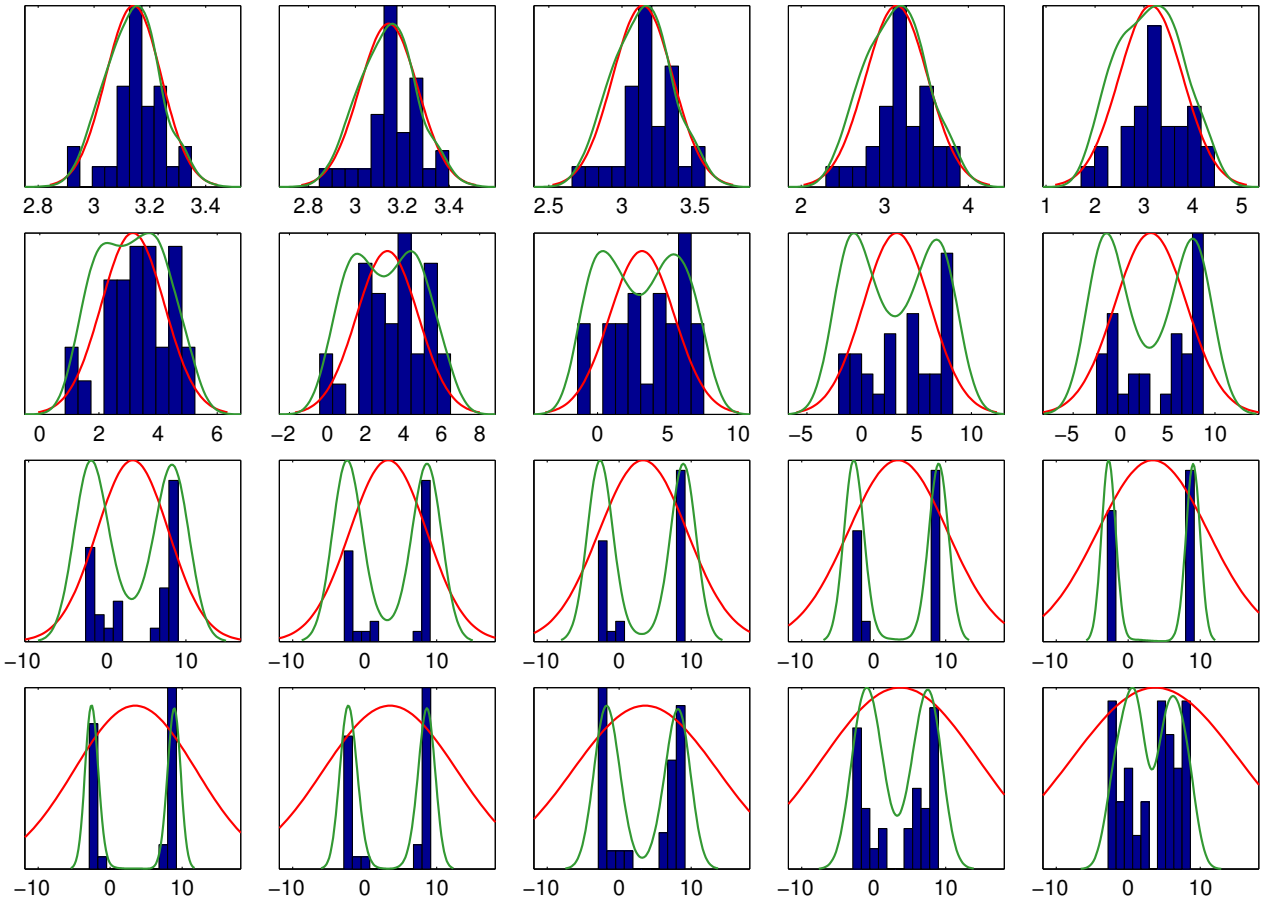


Figure 4.10: Distributions over the pendulum angle as it falls from upright over 20 time steps. The first plot in the top left corner shows the initial distribution over start positions. The histogram shows the true distribution from thirty repeats, the red Gaussians show the predicted distributions using Gaussian moment-matching at each time step, the green lines show a kernel-smoothed density estimate from the particle simulation method.

Despite this poor simulation performance the moment-matching approach can still optimise a control policy because the goal of the task itself is to hold the state around a fixed value. This means that the control actions taken to solve the task also prevent the multi-modal behaviour of the state distribution, see figure 4.11. Furthermore, the Gaussian moment-matching causes the simulation to be conservative, that is it predicts a worse performance than the truth. This is not so dangerous as predicting the controller will do better than it actually does as the optimisation approach can attempt to remedy the situation. A conservative simulator may however lead to significantly slower policy learning and to a poorer control performance. In the extreme case, the simulation might be so pessimistic that the optimiser cannot find any setting of the policy which it believes will improve task performance and so policy training stops. This situation is made more likely if the loss function contains any sharp edges. For example, with the cart and pendulum system perhaps we do not mind if the cart moves around within a specified window but we have a very strong loss if it moves beyond certain boundaries. In this case the controller may have to keep the cart far from the constraint boundaries in order to reduce the loss associated with the too-wide tails of the state distribution. Finally, note that the difference between the particle approach and the moment-matching approach is dependent on the amount of uncertainty in the dynamical model, as we collect more data and the model becomes more accurate the difference reduces. This again indicates that the moment-matching approach might be slower to learn a control policy as it requires more data to shrink the uncertainty in the simulated trajectories.

Whilst particles are effective for simulating the plant and controller the primary goal of our framework is to optimise a control policy to complete a task. We therefore need a way of optimising the policy whilst using particles to simulate the plant. We can't immediately take derivatives for each of our samples and then average them together because we cannot differentiate the sampling steps. One method for solving this problem is to freeze the random seed at the start of optimisation. This means that if we keep the policy parameters the same the particles will also take on the same values — the simulation stage becomes deterministic again. This approach is taken in the PEGASUS framework (Ng and Jordan, 2000). If we take this approach however we encounter a problem: figure 4.12 shows the loss function evaluated along the direction of the gradient in policy parameter space. The figure shows that the moment-matching approach has the unintended consequence of smoothing the loss function over the policy parameter space, which is necessary for gradient descent to work. The loss surface for the particle simulation approach has many local minima and indeed can often be not smooth. This implies that we cannot use gradient descent coupled with the particle approach due to the jaggedness of the loss function. We are therefore left with either more complex gradient methods or non-gradient based optimisation methods which perform significantly worse, especially when we can have hundreds (or even thousands) of policy parameters. Figure 4.12 shows that the loss function can be reduced by moving in the direction of the gradient, which suggests that we could still use gradient information if we had a more robust linesearch algorithm, able to take into account local minima. Whilst this approach can indeed work at the start of optimisation, once the initial large steps have been taken in policy improvement it becomes harder and harder to find directions which lead to any significant decrease in loss. This results in optimisation terminating much earlier in particle based simulations than for the moment-matched approach.

There seem to be a variety of causes for the lack of smoothness when simulating with particles, and these are not all understood. It is, however, strongly linked to the iterative use of the transition

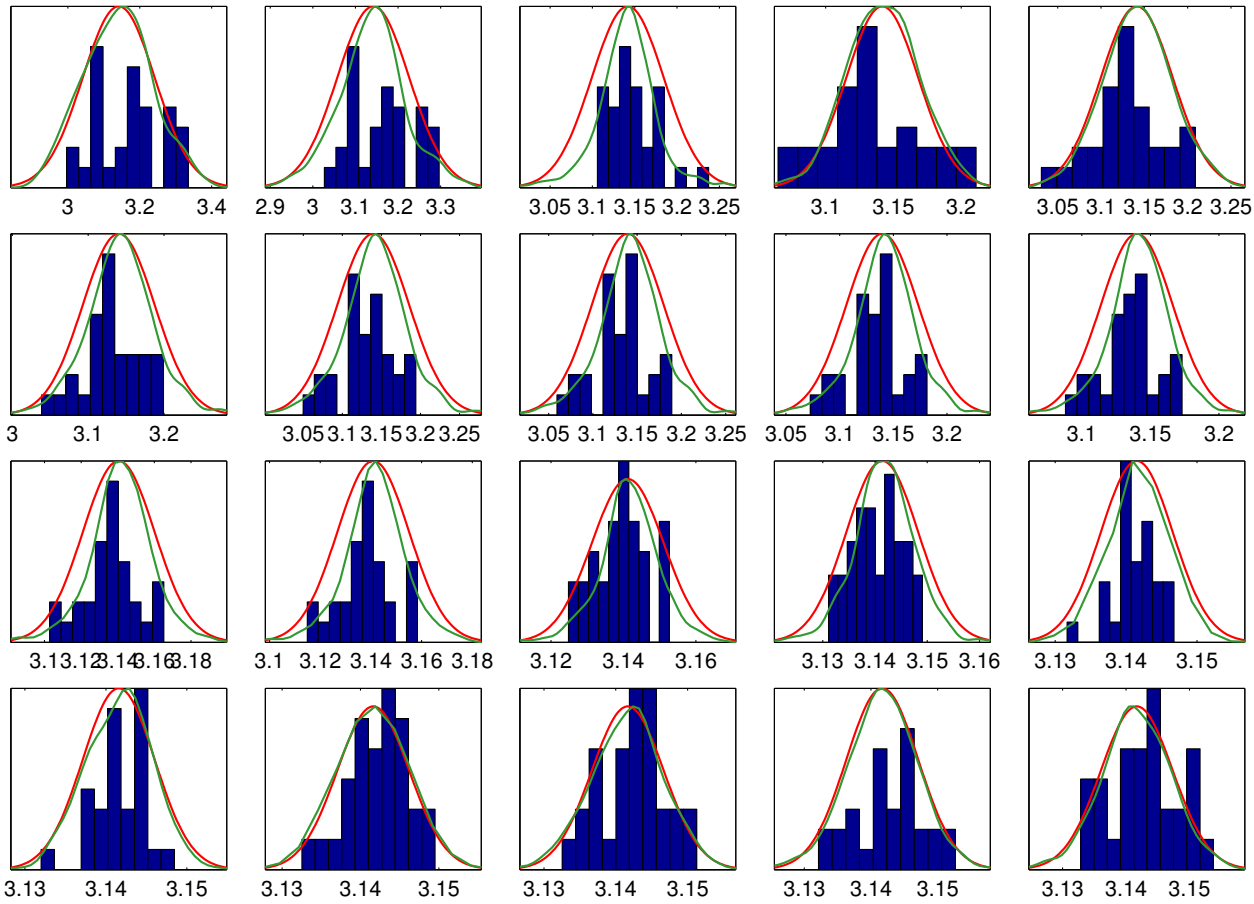


Figure 4.11: Distributions over the pendulum angle using a trained policy which balances the pendulum over 20 time steps. The first plot in the top left corner shows the initial distribution over start positions. The histogram shows the true distribution from thirty repeats, the red Gaussians show the predicted distributions using Gaussian moment-matching at each time step, the green lines show a kernel-smoothed density estimate from the particle simulation method.

function to simulate trajectories over a long horizon: the loss function qualitatively becomes more smooth as we decrease the length of the simulation horizon. Consider making a small change in the control policy parameters and then simulating a particle trajectory over many time steps: the first control variable computed by the policy $\boldsymbol{u}_0$, will be only slightly different to what it was previously as we only made a small difference to the policy. This means that the particle at the next time step $\boldsymbol{x}_1$ is still close to its previous location and the loss is not changed by very much. However, when we compute the next control signal the policy parameters have not only changed but also the state, thus the second control variable will differ from its previous value by a greater amount. This leads to a larger difference in predicted next state, which in turn further increases the difference in subsequent control variables. In this way, with a long enough prediction horizon, particle trajectories can diverge significantly for only a very small change in policy parameters. For the moment-matched approach, the light-tails of a Gaussian distribution seem to keep the state distribution more constricted than for the particles: individual particles can move a lot further around in state space than the mass of a Gaussian distribution. A possible solution based on these insights is to fit a Gaussian distribution to the particles and then resample them. This need not be done every time step but only after a fixed interval. In some preliminary experiments this approach showed promise in smoothing the loss function. Fitting a Gaussian will destroy the potentially non-Gaussian distribution of particles, which

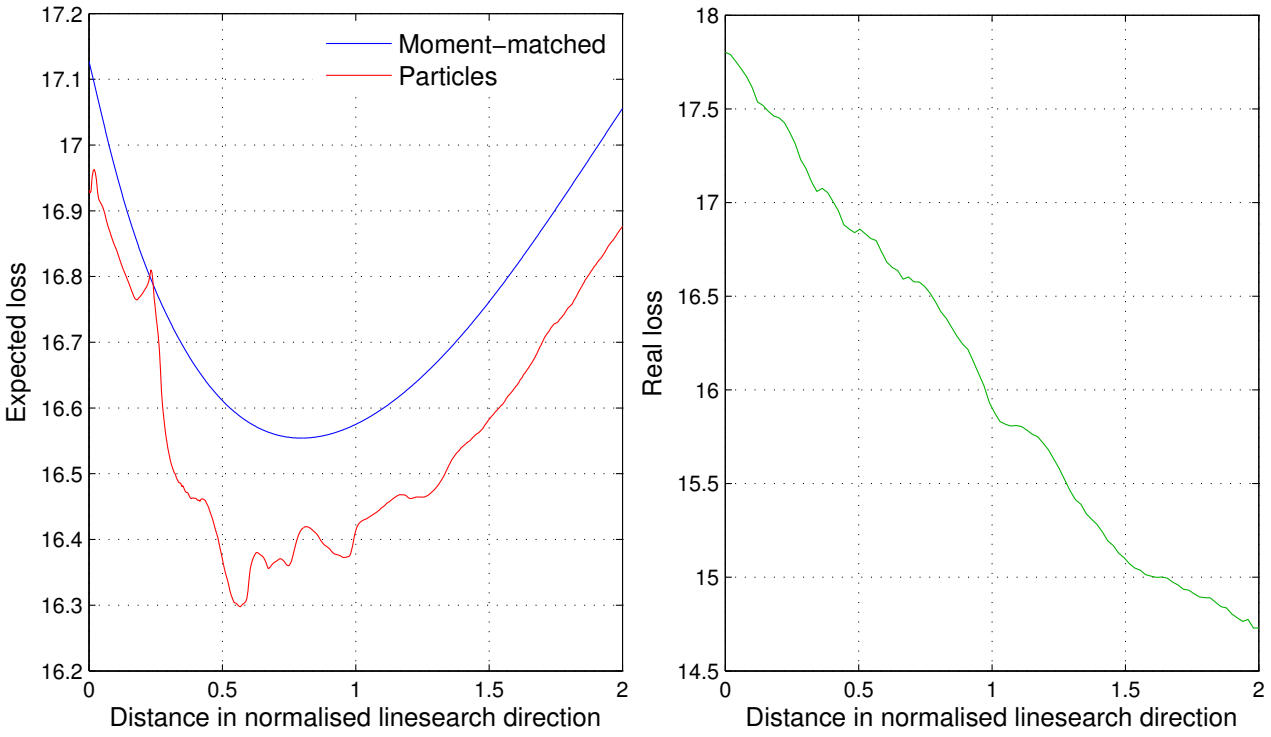


Figure 4.12: A 1D slice through the loss surface in the direction of steepest gradient descent for the cart and pendulum balancing task (starting with the pendulum around the inverted position). The blue line shows the loss for different parameter settings as calculated by the moment-matching approach and the red line for the particle simulation approach with 1000 particles. The green line shows the true loss using the differential equations to simulate the state. The simulation horizon was 20 steps, with a maximum loss of 1 at each time step. The losses in red and blue are based on a model without much training data which is why they do not match the truth.

was one of the motivations for using particles in the first place. We could potentially fit a mixture of Gaussians instead, which would allow the more complex distribution to be maintained. We are restricted in that we need derivatives of the new particles w.r.t. the old particles and so care needs to be taken in this step to ensure we can calculate this derivative.

Another part of the problem is contributed by loss functions in which there is an 'edge', e.g. the pendulum falling beyond the point of recovery: there can be a large step-change in loss as a particle moves from one side of this boundary to another. This can happen in terms of movement in the state space where the particle trajectory either crosses this boundary or not; it can also happen in terms of time, whereby a particle crossing the boundary at time $t + \tau$ can have a very similar total loss whilst $\tau < \delta_t$, the discrete time step. As an example of this, one can imagine the situation where a trajectory passes across an area of very high loss but the sample times catch the particle just before and just after this region and thus the whole trajectory has a low loss. Any movement of particles however would lead to a jump in loss. This can be mitigated somewhat by increasing the number of particles although this can become computationally infeasible long before the loss function is sufficiently smooth. Another approach to pursue would be to consider whether we could integrate the loss over the length of the trajectory to avoid the time step problem, although it is not clear at present how this could be done. We could try to avoid loss functions which exhibit these characteristics, however they are inherent to all tasks attempting to stabilise open-loop unstable systems as the plant dynamics will inevitably drive particles which fail to be stabilised into areas of high loss.

Given that sampling has been successfully used in the PEGASUS algorithm we are hopeful of finding a solution such that training can be carried out successfully. As has been discussed, using particles promises improvements in a number of areas: from more accurately representing the state uncertainty to compuational speed-ups via parallelisation. However, further research is required here in order to unlock this potential.

4.7 Simulation with Dependent Transitions

In the control learning framework discussed so far all the state transitions during the simulation stage are assumed to be independent. That is, the transition prediction of $\boldsymbol{x}_t$ is only dependent on the state $\boldsymbol{x}_{t-1}$, the controls $\boldsymbol{u}_{t-1}$, and the previously observed training data. It is not affected at all by any previous transitions, for example from $\boldsymbol{x}_{t-2}$ to $\boldsymbol{x}_{t-1}$. In state space theory this Markov transition is justified based on the definition of the state: the state $\boldsymbol{x}_{t-1}$, along with the control $\boldsymbol{u}_{t-1}$, is required to contain all information required to fully specify the transition to the next time step. However, a state vector which is sufficient for a deterministic transition function is not necessarily sufficient for an uncertain transition function. As we will show, making transitions in the presence of uncertainty leads to covariance between the state and the uncertainty in the model. Neglecting this extra covariance term can potentially have a large effect on predictions. This extra covariance term is most easily seen for a linear transition function with an uncertain bias term,

$$\boldsymbol{x}_t \;=\; A\,\boldsymbol{x}_{t-1} \;+\; \boldsymbol{b}, \qquad \boldsymbol{b} \;\sim\; \mathcal{N}(\boldsymbol{\mu}_b,\; \Sigma_b) \tag{4.25}$$

Figure 4.13 illustrates this for a one dimensional state. Suppose that we start at the point $\boldsymbol{x}_0$, after one time step we have,

$$\boldsymbol{x}_1 \;\sim\; \mathcal{N}(A\,\boldsymbol{x}_0 \;+\; \boldsymbol{\mu}_b,\; \Sigma_b) \tag{4.26}$$

It is tempting to write that after two time steps the state distribution would be,

$$\boldsymbol{x}_2 \;=\; A\,\boldsymbol{x}_1 \;+\; \boldsymbol{b} \;=\; A\,(A\,\boldsymbol{x}_0 \;+\; \boldsymbol{b}) \;+\; \boldsymbol{b} \tag{4.27}$$

$$\boldsymbol{x}_2 \;\sim\; \mathcal{N}(A\,(A\,\boldsymbol{x}_0 \;+\; \boldsymbol{\mu}_b) \;+\; \boldsymbol{\mu}_b,\; A\,\Sigma_b\,A^T \;+\; \Sigma_b) \tag{4.28}$$

However, if we consider figure 4.13 then we see that smaller values of $\boldsymbol{x}_1$ result from following a transition function similar to the red line, and larger values from following a transition function similar to the blue line. In other words there is covariance between $\boldsymbol{x}_1$ and $\boldsymbol{b}$ which we have neglected from our calculation of the moments in equation 4.28.

An alternative way of thinking about this is in terms of sampling a number of transition functions at the start of the simulation stage. Our start state can be propagated to the next time step using each of the sampled transition functions. Given that we are modelling time-invariant dynamics, we should then use the same sampled transition function for the second transition as we did for the first. In figure 4.13, suppose we draw the mean transition function (green line) and the red and blue transition functions. Starting from $\boldsymbol{x}_0$ we will end up with three values of $\boldsymbol{x}_1$ as indicated by the red, green, and blue dots in the figure. When transitioning to $\boldsymbol{x}_2$ we should use the red transition function to propagate the red dot, the green to propagate the green dot, and the blue function for the blue dot.

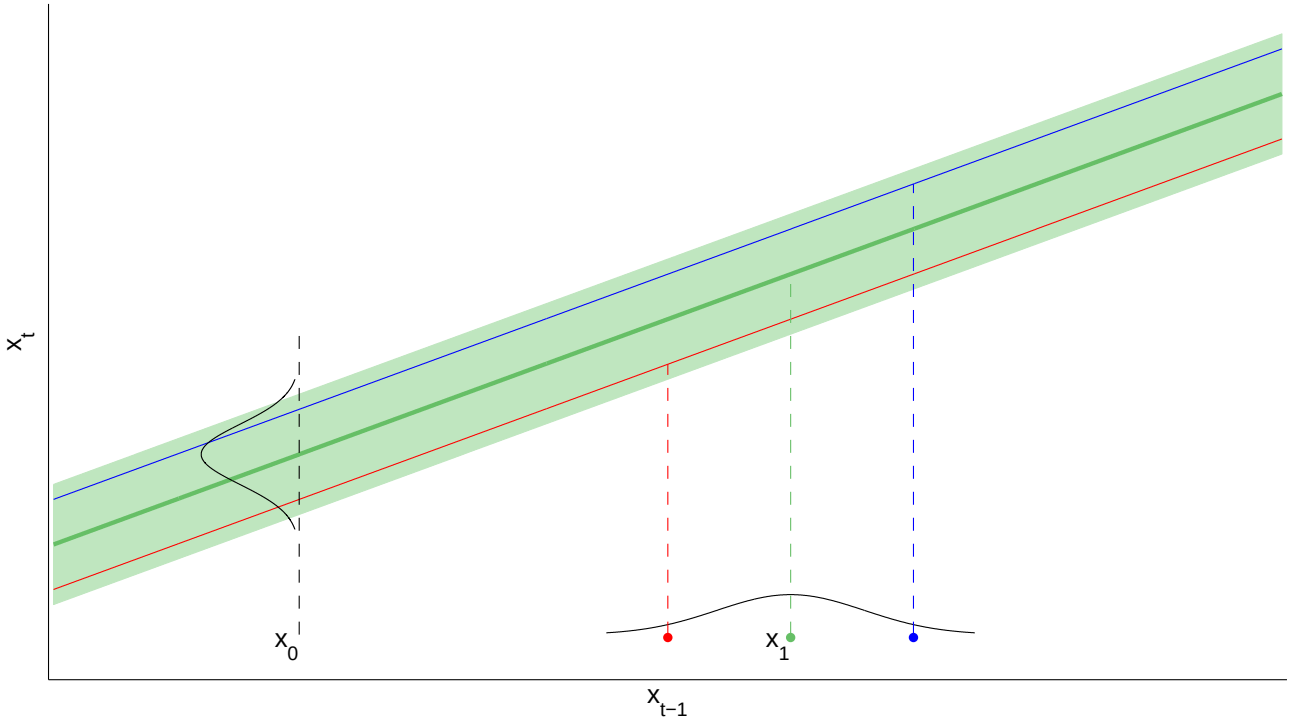


Figure 4.13: Illustration of the dependent transition problem for a linear transition model with an uncertain bias term. The green line and shaded area shows the distribution over the next state for a particular current state: the solid line represents the mean and the shaded area shows ± 2 standard deviations. The red and blue lines indicate two sampled transition functions from the model. The value of $\boldsymbol{x}_1$ indicated by the red dot can only be reached by using the red transition function, and similarly for the green and blue dots. This means that the uncertainty in $\boldsymbol{x}_1$ covaries with the uncertainty over the transition function: functions closer to the red line will result in states closer to the red dot. This covariance should be taken into account when propagating the state further.

The current propagation methodology in the framework is equivalent to resampling the transition functions at each time step. This can lead to poor simulation performance. For example, in figure 4.13, the red state should always use the red transition function, below the mean dynamics, whereas if we neglect this covariance we will integrate over the full distribution on $\boldsymbol{x}_2$, evaluated at the red state, which includes transition functions above the mean dynamics. This means that if we were to trace the path of the red state through multiple transitions we would see it, on average, using the mean transition function. We can see that this will lead to a distribution on the state trajectory with a too small variance — extremal states should always continue to use the extremal transition functions, pushing them even further apart than if they use, on average, the mean transition function.

4.7.1 The uncertain linear model

We will first derive the correct moments for the fully uncertain linear model and then consider the GP. For simplicity we will restrict ourselves to looking at a one dimensional model, the multi-dimensional case is also analytically tractable but requires a lot more algebra (or a more elaborate notation). We

extend the linear model of equation 4.25 by also considering the weights to be uncertain,

$$x_t \;=\; a\, x_{t-1} \;+\; b \tag{4.29}$$

$$\begin{bmatrix} a \\ b \end{bmatrix} \;\sim\; \mathcal{N}\left(\begin{bmatrix} \mu_a \\ \boldsymbol{\mu}_b \end{bmatrix}, \begin{bmatrix} \Sigma_a & C_{ab} \\ C_{ba} & \Sigma_b \end{bmatrix} \right) \tag{4.30}$$

Also, let x_0 have a Gaussian distribution,

$$x_0 \;\sim\; \mathcal{N}\left(\mu_{x_0},\, \Sigma_{x_0}\right) \tag{4.31}$$

Under this setup we can compute,

$$\begin{aligned} \mathbb{E}[x_1] \;&=\; \mathbb{E}_{x_0,a,b}[a\, x_0 \;+\; b] \\ &=\; \mu_a\, \mu_{x_0} \;+\; \mu_b \end{aligned} \tag{4.32}$$

and

$$\begin{aligned} \mathbb{V}[x_1] \;&=\; \mathbb{V}_{x_0,a,b}[a\, x_0 \;+\; b] \\ &=\; \Sigma_a\, \Sigma_{x_0} \;+\; \mu_a^2\, \Sigma_{x_0} \;+\; \mu_b^2\, \Sigma_a \;+\; 2\, \mu_{x_0}\, C_{ab} \;+\; \Sigma_b \end{aligned} \tag{4.33}$$

but we also have the covariances,

$$\begin{aligned} \mathbb{C}[x_1,\, a] \;&=\; \mathbb{E}_{x_0}\left[\mathbb{C}_{a,b}[a\, x_0 + b,\, a]\right] \;+\; \mathbb{C}_{x_0}\left[\mathbb{E}_{a,b}[a\, x_0 + b],\, \mathbb{E}[a]\right] \\ &=\; \mathbb{E}_{x_0}\left[\Sigma_a\, x_0 + C_{ba}\right] \;+\; 0 \\ &=\; \Sigma_a\, \mu_{x_0} + C_{ba} \end{aligned} \tag{4.34}$$

and

$$\begin{aligned} \mathbb{C}[x_1,\, b] \;&=\; \mathbb{E}_{x_0}\left[\mathbb{C}_{a,b}[a\, x_0 + b,\, b]\right] \;+\; \mathbb{C}_{x_0}\left[\mathbb{E}_{a,b}[a\, x_0 + b],\, \mathbb{E}[b]\right] \\ &=\; \mathbb{E}_{x_0}\left[C_{ab}\, x_0 + \Sigma_b\right] \;+\; 0 \\ &=\; C_{ab}\, \mu_{x_0} + \Sigma_b \end{aligned} \tag{4.35}$$

The product of Gaussian variables in equation 4.29, $a\, x_0$, means that x_1 is not Gaussian, which spells the end for analytic tractability. We are trying to parallel the simulation stage in the control learning framework and so we will proceed as we do there and approximate the distribution with a Gaussian. Doing so here allows us to write a joint probability distribution on the state and the model,

$$p(x_1,\, a,\, b \mid \mu_{x_0},\, \Sigma_{x_0}) \;\approx\; \mathcal{N}\left(\begin{bmatrix} \mu_a\, \mu_{x_0} \;+\; \mu_b \\ \mu_a \\ \mu_b \end{bmatrix}, \begin{bmatrix} \Sigma_{x_1} & \Sigma_a\, \mu_{x_0} + C_{ba} & C_{ab}\, \mu_{x_0} + \Sigma_b \\ \Sigma_a\, \mu_{x_0} + C_{ba} & \Sigma_a & C_{ab} \\ C_{ab}\, \mu_{x_0} + \Sigma_b & C_{ba} & \Sigma_b \end{bmatrix} \right) \tag{4.36}$$

If we now consider x_2,

$$\begin{aligned}\mathbb{E}[x_2] &= \mathbb{E}_{x_1,a,b}[a\,x_1 \;+\; b] \\ &= \mu_a\,\mu_{x_1} \;+\; \mathbb{C}[a,\,x_1] \;+\; \mu_b \\ &= \mu_a\,(\mu_a\,\mu_{x_0} \;+\; \mu_b) \;+\; \Sigma_a\,\mu_{x_0} + C_{ba} \;+\; \mu_b \end{aligned} \tag{4.37}$$

and

$$\begin{aligned}\mathbb{V}[x_2] &= \mathbb{V}_{x_1,a,b}[a\,x_1 \;+\; b] \\ &= \Sigma_a\,\Sigma_{x_0} \;+\; \mu_a^2\,\Sigma_{x_0} \;+\; \mu_b^2\,\Sigma_a \;+\; (C_{x_1,a} \;+\; \mu_a\,\mu_{x_1})\,C_{x_1,a} \;+\; 2\,\mu_{x_1}\,C_{ab} \;+\; 2\,\mu_a\,C_{x_1,b} \;+\; \Sigma_b \end{aligned} \tag{4.38}$$

where we have used the shorthand $C_{x_1,a} \;=\; \mathbb{C}[x_1,\,a]$ and similarly for $C_{x_1,b}$. Comparing equations 4.37 and 4.38 to the moments in equation 4.28 we can see that the covariance of the state with the model parameters has led to both a different mean and variance for x_2. We can illustrate the difference by simulating from the system described in equations 4.29 and 4.30 using the predictive equations with and without the covariance between the state and the model parameters. State trajectories for four such systems are shown in figure 4.14. Note how the state distribution after one time step is the same for both methods of propagating the state as the covariance between the state and the model is initially zero. However, the two methods then diverge and for all subsequent time steps the distributions are different. The top right hand plot shows an example where the covariance leads to a smaller variance in the state distribution, whereas the other three show systems where the variance is increased. The bottom right hand plot in particular shows a very large difference in state distribution between the two methods: neglecting the covariance between the state and the model in this case would lead to a vastly over-confident estimate of the state.

4.7.2 Dependent transitions in GP models

We now look at the same covariance between state and model for the GP dynamical model. There is a fundamental difference which means that the analysis we have just completed for the uncertain linear model cannot be solved for the GP. In the linear model the covariance between the state x_1 and the model is independent of the value of x_1. However, this is not the case for a GP. In a GP the covariance is determined by the covariance function, which depends on the state value. Recall that in 1D our GP model of the transition function is given by,

$$x_t \;=\; f(x_{t-1}) \;+\; \epsilon_t \;\triangleq\; f_t \;+\; \epsilon_t \tag{4.39}$$

$$f \;\sim\; \mathcal{GP}(m, k) \tag{4.40}$$

$$\epsilon_t \;\sim\; \mathcal{N}(0,\; \Sigma_\epsilon) \tag{4.41}$$

The graphical model is repeated from chapter 3 here in figure 4.15.

Suppose that we have some previously observed transitions, X, from which we have trained our GP model to give us a posterior on the transition function, as shown in green (mean and plus/minus two standard deviations) in the top plot of figure 4.16. In the plot we also show a number of different 'sampled functions' (to be precise, different collections of points) from this posterior at various values

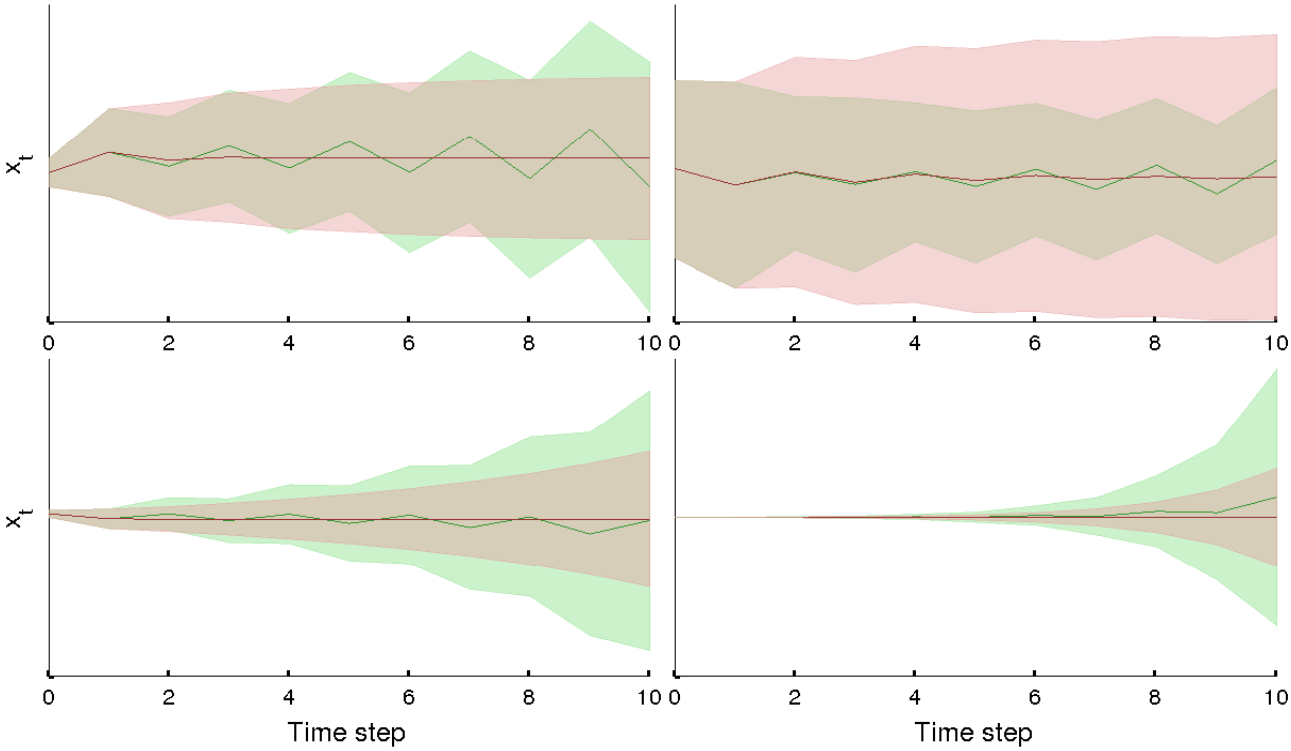


Figure 4.14: Comparison between simulations of a 1D uncertain linear system with (green) and without (red) the covariance between the state and the model parameters. The four plots show simulations using four different randomly generated 1D systems over a horizon of 10 steps. The solid line shows the mean of the state distribution and the shaded region ± 2 standard deviations.

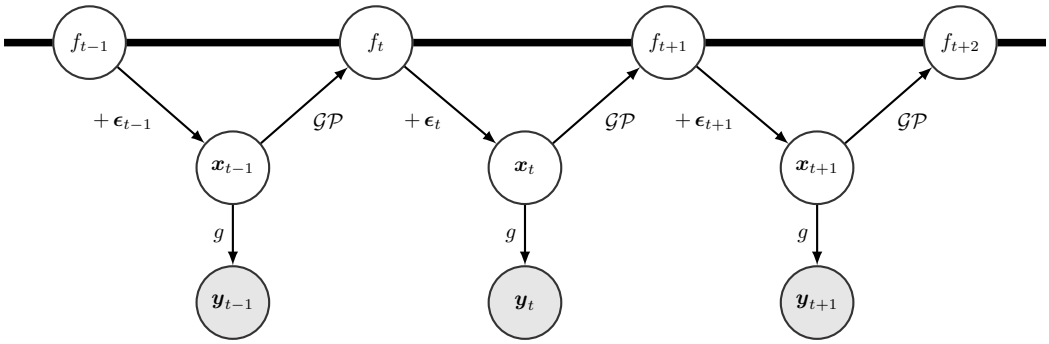


Figure 4.15: Graphical model of a Gaussian Process state space model. The random variables f represent the GP function values, which are fully connected to each other, as represented by the thick line. The transition of $\boldsymbol{x}_{t-1}$ to $\boldsymbol{x}_t$ is modelled by the GP followed by the addition of process noise $\boldsymbol{\epsilon}_t$

of x_{t-1}. We can see that there is strong covariance between values of x_t across the input space — functions above the mean remain above the mean for most of the input space. This suggests that previous transitions are likely to have a strong effect on future predictions. First we draw a sample x_0 from our initial distribution. Using our GP model we can make a prediction at x_0 and find that,

$$p(f_1, x_1 \mid x_0) = \mathcal{N}\left(\begin{bmatrix} f_1 \\ x_1 \end{bmatrix}; \begin{bmatrix} \mathfrak{m}(x_0) \\ \mathfrak{m}(x_0) \end{bmatrix}, \begin{bmatrix} \mathfrak{s}(x_0) & \mathfrak{s}(x_0) \\ \mathfrak{s}(x_0) & \mathfrak{s}(x_0) + \Sigma_\epsilon \end{bmatrix}\right) \tag{4.42}$$

The marginal distributions over f_1 and x_1 are shown as the blue and red Gaussians in figure 4.16 respectively. Imagine we now also draw a value of x_1, as shown by the red dot in figure 4.16. We can condition on this x_1 too, which is equivalent to adding the transition from x_0 to x_1 to the GP training

set and then recalculating the GP posterior moments,

$$p(f_1 \mid x_1,\, x_0) \;=\; \mathcal{N}\left(\mathfrak{m}(x_0) + \mathfrak{s}(x_0)\,(\mathfrak{s}(x_0) + \Sigma_\epsilon)^{-1}(x_1 - \mathfrak{m}(x_0)),\; \mathfrak{s}(x_0) - \mathfrak{s}(x_0)\,(\mathfrak{s}(x_0) + \Sigma_\epsilon)^{-1}\mathfrak{s}(x_0)\right) \quad (4.43)$$

Given that a GP specifies a joint Gaussian distribution on its latent variables with covariance given by k, the marginal distribution on f_2 given x_0 and x_1 has a similar form,

$$p(f_2 \mid x_1,\, x_0) \;=\; \mathcal{N}\Big(\mathfrak{m}(x_1) \;+\; C_{f_1,f_2}\,(\mathfrak{s}(x_0) + \Sigma_\epsilon)^{-1}(x_1 - \mathfrak{m}(x_0)),\; \mathfrak{s}(x_1) \;-\; C_{f_1,f_2}\,(\mathfrak{s}(x_0) + \Sigma_\epsilon)^{-1}C_{f_1,f_2}\Big) \quad (4.44)$$

with

$$C_{f_1,f_2} \;=\; k(x_0,\, x_1) - \boldsymbol{k}(x_0)^T\, K^{-1}\boldsymbol{k}(x_1) \quad (4.45)$$

where $\boldsymbol{k}(x_0)$ is the $N \times 1$ vector of covariances between x_0 and each of the training points, and K is the $N \times N$ matrix of covariances between each of the training points. This conditional distribution is shown as the red shaded area in figure 4.16 for different values of x_1. Note how different the green and red predictive distributions are. This is the effect of including the transition from x_0 to x_1 in the prediction of f_2. The difference depends on the ratio between the variance of the marginal $p(f_1 \mid x_0)$ and the process noise variance Σ_ϵ as well as on the the value of the covariance function, $k(x_0,\, x_1)$.

So far we have derived the moments of the distribution on f_2 for given samples of x_1 and x_0. Of course during an actual simulation state the states x_0 and x_1 are uncertain and so we need to integrate them out of equation 4.44. We can immediately foresee a problem however, x_0 appears inside a complex inverse in both the mean and variance of equation 4.44, which means we will be unable to integrate it out analytically. The dependence on x_0 is in fact even more complex as we must first integrate out x_1, because the distribution on x_1 is also a function of x_0 (equation 4.42). Doing so,

$$\begin{aligned} p(f_2 \mid x_0) \;&=\; \int p(f_2 \mid x_1,\, x_0)\, p(x_1 \mid x_0)\; dx_1 \\ &=\; \int p(f_2 \mid x_1,\, x_0)\; \mathcal{N}\left(x_1;\; \mathfrak{m}(x_0),\; \mathfrak{s}(x_0) \;+\; \Sigma_\epsilon\right)\; dx_1 \end{aligned} \quad (4.46)$$

Although this is the integral of a product of two Gaussians the resulting distribution is non-Gaussian as x_1 appears three times in both the mean and the variance of $p(f_2 \mid x_1,\, x_0)$. The histogram on the left of figure 4.17 shows a sample-based estimate of $p(f_2 \mid x_0)$, which has significant skewness. We can still compute the moments analytically however,

$$\begin{aligned} \mathbb{E}[f_2 \mid x_0] \;&=\; \mathbb{E}_{x_1}\left[\mathfrak{m}(x_0) \;+\; C_{f_1,f_2}\,(\mathfrak{s}(x_0) + \Sigma_\epsilon)^{-1}(x_1 - \mathfrak{m}(x_0))\right] \\ &=\; \mathbb{E}_{x_1}\left[\boldsymbol{k}(x_1)\right]^T\boldsymbol{\beta} \;+\; \mathbb{E}_{x_1}\left[\left(k(x_0, x_1) \;-\; \boldsymbol{k}(x_0)\,K^{-1}\,\boldsymbol{k}(x_1)\right)\left(x_1 \;-\; \boldsymbol{k}(x_0)^T\boldsymbol{\beta}\right)\right](\mathfrak{s}(x_0) + \Sigma_\epsilon)^{-1} \\ &=\; \mathbb{E}_{x_1}\left[\boldsymbol{k}(x_1)\right]^T\boldsymbol{\beta} \;+\; \Big[\left(\mathbb{E}_{x_1}\left[k(x_0, x_1)\right] \;-\; \boldsymbol{k}(x_0)\,K^{-1}\,\mathbb{E}_{x_1}\left[\boldsymbol{k}(x_1)\right]\right)\left(\mathbb{E}_{x_1}\left[x_1\right] \;-\; \boldsymbol{k}(x_0)^T\boldsymbol{\beta}\right) \\ &\qquad\qquad +\; \mathbb{C}_{x_1}\left[k(x_0, x_1),\, x_1\right] \;-\; \boldsymbol{k}(x_0)\,K^{-1}\,\mathbb{C}_{x_1}\left[\boldsymbol{k}(x_1),\, x_1\right]\Big](\mathfrak{s}(x_0) + \Sigma_\epsilon)^{-1} \end{aligned} \quad (4.47)$$

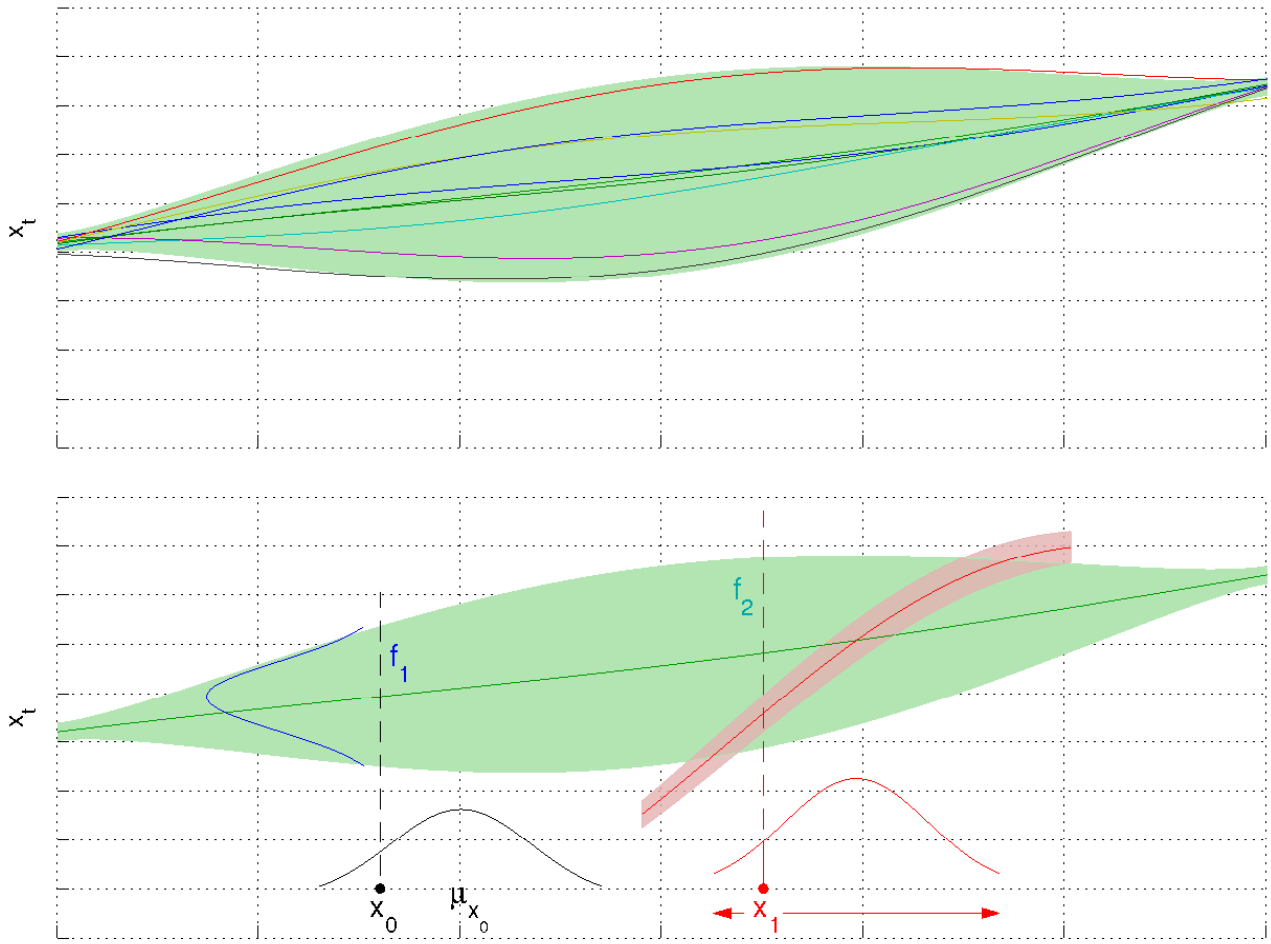


Figure 4.16: Illustration of how the state covaries with the GP transfer function. The top plot shows the GP posterior along with some sampled functions, which show long range covariance. The bottom plot shows the same posterior along with the one (in red) that results from also conditioning on the $\{x_0,\, f_1\}$ transition, as f_1 varies.

$$\begin{aligned}\mathbb{V}[f_2 \mid x_0] &= \mathbb{E}_{x_1}\left[\mathfrak{s}(x_1) - C_{f_1,f_2}\left(\mathfrak{s}(x_0)+\Sigma_\epsilon\right)^{-1}C_{f_1,f_2}\right] + \mathbb{V}_{x_1}\left[\mathfrak{m}(x_0) + C_{f_1,f_2}\left(\mathfrak{s}(x_0)+\Sigma_\epsilon\right)^{-1}(x_1-\mathfrak{m}(x_0))\right] \\ &= \mathbb{E}_{x_1}\left[\mathfrak{s}(x_1)\right] - \mathbb{E}_{x_1}\left[(k(x_0,\,x_1)-\boldsymbol{k}(x_0)^T K^{-1}\boldsymbol{k}(x_1))^2\right]\left(\mathfrak{s}(x_0)+\Sigma_\epsilon\right)^{-1} \\ &\quad + \mathbb{V}_{x_1}\left[\mathfrak{m}(x_0)\right] + \mathbb{V}_{x_1}\left[(k(x_0,\,x_1)-\boldsymbol{k}(x_0)^T K^{-1}\boldsymbol{k}(x_1))\,(x_1-\mathfrak{m}(x_0))\right]\left(\mathfrak{s}(x_0)+\Sigma_\epsilon\right)^{-2} \quad (4.48)\end{aligned}$$

Each of the moments in equations 4.47 and 4.48 involve integrals over the covariance function. These are the same moments as have been required in various other chapters. For the squared exponential kernel they can be found in Appendix A. It is important to note that every one of these moments will depend on x_0, often in a complicated way. Because of this it is very hard to see how one might solve the integral over x_0 even approximately. One option is to ignore the uncertainty in x_0 and just replace all references to x_0 with the mean μ_{x_0}. Although, this is a very crude approximation it can sometimes lead to reasonable results. The right hand histogram in figure 4.17 shows a sample approximation to $p(f_2)$, which is visually very similar to the left hand histogram which shows $p(f_2|x_0)$. However, in other situations this approximation can lead to worse performance than just treating the transitions as independent.

We hope that with further consideration a better approximate method for solving this problem can be found. It is interesting to note that, in their variational approximation to GP state space models (see

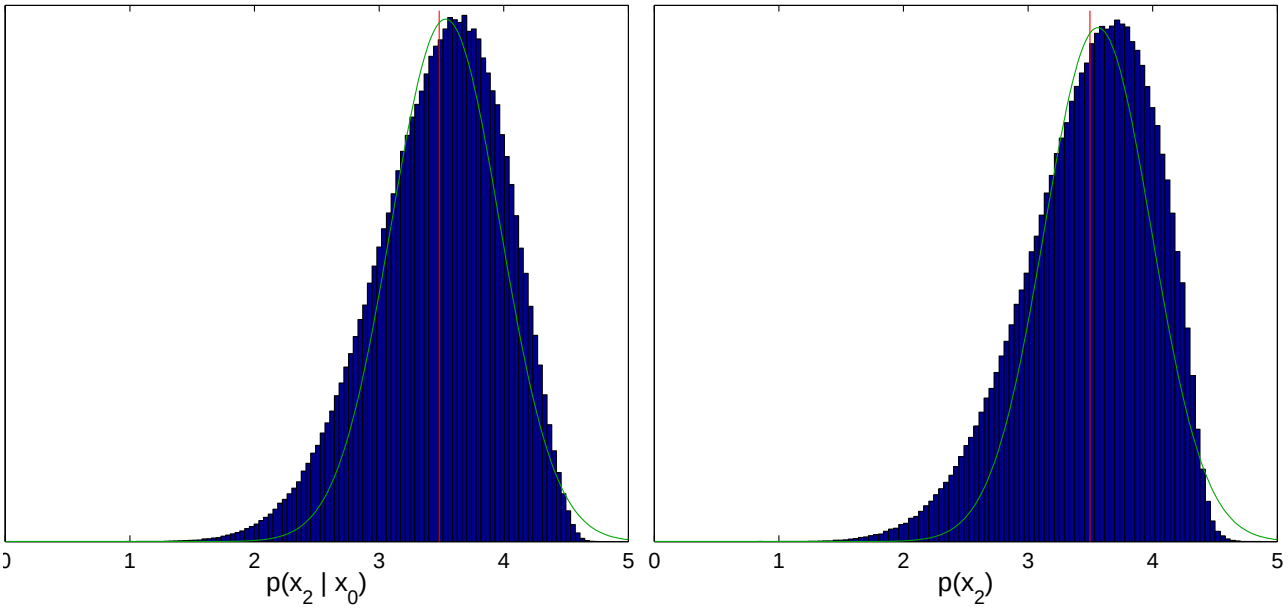


Figure 4.17: Histograms showing a sample-based approximation to the densities $p(x_2 \mid x_0)$, for the value of x_0 shown in figure 4.16, and $p(x_2)$. Neither distribution is Gaussian, as both of them have significant skewness. The red lines show the sample mean and the green Gaussians show the moment matched distributions based on independent transitions. The means are very similar between the independent transition model and for the model taking the previous transition into account. However, disregarding the covariance between transitions leads to a significant underestimate of the variance.

section 3.4.4) Frigola et al. (2014) find that their variational lower bound is the same for the model with dependent transitions as for the model with independent transitions. This implies that their approach is also not able to take the transition covariances into account and a more careful approach is required. At the beginning of this section we stated that if the state captures all the required information to determine a transition then the transitions are indeed independent. One different methodology to solving the problem therefore is to augment the state definition, for example by including previous states and actions (i.e. a higher order Markov system). Of course we still might then have to make independence assumptions but this may have a much smaller effect on the state trajectory than it would with a more parsimonious state representation.

As with the particle simulation methodology, considering dependent transitions allows a more accurate simulation process of the dynamical system, which is the heart of the control learning framework. If we cannot accurately simulate the system we are trying to control then the framework is seriously hindered in its ability to control the plant. However, once again, further research is required if we are to unlock any potential improvements dependent transitions may bring to the framework.

4.8 Conclusion

Automatic control learning algorithms promise large advances in the field of control engineering, both in tackling complex systems and in adapting to incoming data. The PILCO framework has previously demonstrated its ability to learn a successful controller on a range of control tasks. However, it had a significant shortcoming in how observation noise was handled. By introducing a proper GP state space model we have greatly improved the learning performance, enabling controllers to be learnt where it

was impossible previously. We witnessed that the controller still had some difficulty with stabilising all trials which we hypothesise is due to noise being injected through the control policy. Research is currently being carried out into including a filter in the control loop based on the GP dynamical model. We suspect that this will lead to another compelling increase in performance.

In this chapter, we have also investigated the use of particles to simulate trajectories (in a similar manner to the PAGASUS algorithm) and the dependence between successive predicted transitions from the GP. Both of these areas are a direct result of using a stochastic model for capturing the plant's dynamics and thus their effects are strongly related to the amount of uncertainty in the dynamical model. This means that they have their strongest effect at the beginning of training and this diminishes as more data is acquired. This suggests that the current training framework can avoid these problems by collecting more data than it might otherwise require if they were tackled properly. In both of these areas we highlighted the potential benefits that could be brought as well as the current barriers preventing their exploitation. We highlighted some potential avenues for finding solutions to the discussed problems which opens these up for future research.

Chapter 5

Conclusions and Further Work

Gaussian Processes are a very powerful method for modelling data. However, their classical derivation (Rasmussen and Williams, 2006) only considers the presence of output noise, which limits their use. This is particularly problematic if we wish to model dynamical systems, whereby the output at one time step becomes input at the subsequent time step. In these circumstances, it is clearly nonsensical to model the outputs as noisy but maintain that the inputs are clean. In this thesis we introduced a simple method for extending GPs to the case of input noise: NIGP (McHutchon and Rasmussen, 2011). This approach refers the input noise to the output and models it as heteroscedastic output noise. This transfer of noise to the output in undertaken by the use of a Taylor series approximation to the GP posterior. The intuition behind this can be seen by realising that uncertainty in the input location of a point affects the predicted output much more severely in areas where the gradient of the function is large, than in areas where the function is nearly flat. We demonstrated how NIGP can be easily applied to a number of datasets where input noise was a problem, and how predictive performance was improved over a classic GP. We also presented a variational approach to the problem but showed it had problems with under-fitting.

In chapter 3 we investigated modelling dynamical systems with GPs. In this application, as in NIGP, there is noise affecting the input measurements, however here we have additional structure in that the latent states form a chain over time: the system transitions from state-to-state. Whilst inference in this model is intractable there are a number of different approximate approaches that one could take. We discussed a Bayesian treatment based on sampling (Frigola et al., 2013), a variational approach, and then looked in detail at four different methods for learning in a 'parametric GP'. In this model we can integrate over the uncertainty in the latent states by introducing a pseudo-training set to the GP, which we optimise. First we introduced a novel analytic method which directly optimised the approximate marginal likelihood via moment-matching Gaussians to intractable densities. We then considered a family of approximate-analytic EM methods, which used different algorithms in the E step. We demonstrated the limitations of the expectation propagation E step method presented in Deisenroth and Mohamed (2012), and extended their approach to include a sampling step, which overcame these limitations. In this context, we investigated three different sampling schemes and showed how they led to an improvement in the performance of the EP-EM algorithm for the GP state space model problem. We then showed how to apply the sequential Monte Carlo method of Poyiadjis et al. (2011) to approximate the true derivatives of the marginal likelihood in the GP state space model, and designed a derivative-only optimisation scheme to use them. Finally we presented a

particle EM method, which used the particle Gibbs with ancestor sampling algorithm (Lindsten et al., 2012) in the E step.

Whilst optimising the pseudo-points is vulnerable to overfitting our experiments showed that this only affected the approximate-analytic EM method with any appreciable severity. Furthermore, this approach provides a powerful method for modelling dynamical systems as we demonstrated on three nonlinear test sets: a one dimensional 'kink' function, the four dimensional cart and pendulum system, and the ten dimensional robotic unicycle. Out of all the methods we considered in this chapter, the novel direct optimisation method proved to be the most successful.

In chapter 4 we discussed the PILCO control learning framework. This methodology uses a GP dynamical model to simulate a system of interest and then optimises a control policy to achieve a specified task based on this simulation. The framework has been previously tested on numerous problems, but always for the case of negligible observation noise. We show how the presence of noise causes the framework to fail at even simple tasks. Given the prevalence of noise in the real world this is a fatal flaw with the framework. However, we then demonstrated how including a proper GP state space model dramatically improves the control learning performance. We also discussed some other open areas for research in the framework, including simulation with particles rather than via moment-matching, and including the covariance between subsequent transitions within the predictions. Simulation with particles allows us to represent non-Gaussian densities over the state, which could be a big advantage. Unfortunately, when using particles the loss surface is no longer smooth as a function of the policy parameters, which makes it very difficult to optimise. We create a distribution over trajectories of several time steps by chaining together multiple predictions from the transition model. Because of this, the uncertainty in both the model and the state covaries, and this covariance should be taken into account when making predictions. We derive the necessary equations to do this in a stochastic linear system but show that it is intractable to do this for the GP transition model.

Although the comparisons in chapter 3 were extensive, there is still a lot of work which can be done. In particular the particle EM approach, in experiments, performed below the level that the theory suggested it might achieve. This could well be down to implementation problems, which also affected the approximate-analytic EM. We would like to investigate this further. In section 3.8.5, we discussed an extension to the particle EM approach based on Lindsten (2013b), however, we did not consider it in our experiments due to time restrictions. We would therefore like to pursue a more detailed evaluation of this method. We would also like to pursue the variational method further to see how it might perform on the more complex problems. Its weakness was in the number of variational parameters it had to optimise, however we could, instead of optimising the distribution on the latent states, use a smoothing method to estimate it. This would have its own intricacies and needs further research. In addition the state space models are currently painfully slow, although we expect there are a lot of savings that could be made.

There are a number of exciting avenues for further research in the automatic control learning framework. We have discussed a number of these, such as continuing the work on particle simulation, already. Others include introducing a filter to the controller to allow it reduce the amount of noise that is introduced via the control variables, considering systems with completely unobserved variables, and investigating other covariance functions. There are higher level considerations as well: how can this framework be combined with the successful linear control methods which have been applied in may applications so far? Can we design a hierarchical control approach which uses the strengths of

the automatic approach for more complex decisions and simpler controllers to implement low level strategies? Can we make any statements over stability or controllability, in the way that can be done with linear control? Given the potential that automatic control design has to open the door to control of vastly complex systems, these areas are worth investigating.

The field of modelling and control of nonlinear systems is undeniably important to our lives. In our opinion, Gaussian Processes are one of the most powerful approaches for application in this area, combining flexibility, and a principled quantification of uncertainty with tractability. In this thesis we have discussed how to apply GPs to real-world datasets which are corrupted, in inputs and outputs, with noise. We investigated GP state space models in some depth and introduced a powerful novel method. Finally, we demonstrated how these pieces can be put together to produce a potent control learning framework.

Appendices

Appendix A

Squared Exponential Covariance Function: Moments and Derivatives

We have used the squared exponential covariance function (a.k. the squared exponential kernel) throughout this thesis. At various points we required either its derivatives or its integral. We provide the derivation of both of these here. First we define the covariance function: the squared exponential kernel is given by,

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) \;=\; \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)\right) \tag{A.1}$$

where the hyperparameters are the signal variance σ_f^2 and a characteristic length-scale for each dimension, $\{l_i\}_{i=1}^D$. The squared length-scales are collected into a $D \times D$, diagonal matrix Λ,

$$\Lambda \;=\; \begin{bmatrix} l_1^2 & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & l_D^2 \end{bmatrix} \tag{A.2}$$

A.1 Differentiating the Squared Exponential covariance function

The derivative of the squared exponential with respect to the first argument, $\boldsymbol{x}_1$, is,

$$\begin{aligned} \frac{\partial k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_1} &= \frac{\partial}{\partial \boldsymbol{x}_1}\left\{\sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)\right)\right\} \\ &= \frac{\partial}{\partial \boldsymbol{x}_1}\left\{-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)\right\} k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &= -\frac{1}{2}\frac{\partial}{\partial \boldsymbol{x}_1}\left\{\boldsymbol{x}_1^T\Lambda^{-1}\boldsymbol{x}_1 - \boldsymbol{x}_2^T\Lambda^{-1}\boldsymbol{x}_1 - \boldsymbol{x}_1^T\Lambda^{-1}\boldsymbol{x}_2\right\} k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &= -\frac{1}{2}\left(2\Lambda^{-1}\boldsymbol{x}_1 - 2\Lambda^{-1}\boldsymbol{x}_2\right) k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &= -\Lambda^{-1}\,(\boldsymbol{x}_1 - \boldsymbol{x}_2)\; k(\boldsymbol{x}_1, \boldsymbol{x}_2) \end{aligned} \tag{A.3}$$

which is a $D \times 1$ vector. We can also take the derivative w.r.t. the second argument,

$$\begin{aligned}
\frac{\partial k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_2} &= \frac{\partial}{\partial \boldsymbol{x}_2}\left\{\sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1}(\boldsymbol{x}_1 - \boldsymbol{x}_2)\right)\right\} \\
&= \frac{\partial}{\partial \boldsymbol{x}_2}\left\{-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1}(\boldsymbol{x}_1 - \boldsymbol{x}_2)\right\} k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\
&= -\frac{1}{2}\frac{\partial}{\partial \boldsymbol{x}_2}\left\{-\boldsymbol{x}_2^T\Lambda^{-1}\boldsymbol{x}_1 - \boldsymbol{x}_1^T\Lambda^{-1}\boldsymbol{x}_2 + \boldsymbol{x}_2^T\Lambda^{-1}\boldsymbol{x}_2\right\} k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\
&= -\frac{1}{2}\left(-2\Lambda^{-1}\boldsymbol{x}_1 + 2\Lambda^{-1}\boldsymbol{x}_2\right) k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\
&= \Lambda^{-1}(\boldsymbol{x}_1 - \boldsymbol{x}_2)\, k(\boldsymbol{x}_1, \boldsymbol{x}_2) \qquad \text{(A.4)}
\end{aligned}$$

Note that the only difference between these two derivatives (equations A.3 and A.4) is the minus sign in equation A.3. This comes about because the distance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is calculated as $\boldsymbol{x}_1 - \boldsymbol{x}_2$ and hence increasing x_1 increases the separation and so decreases the covariance; the opposite is true for $\boldsymbol{x}_2$.

It is trivial to extend these results for the case when one of the inputs is a collection of points, such as a $N \times D$ training matrix X,

$$X = [\boldsymbol{x}_1, \ \ldots, \ \boldsymbol{x}_N]^T \qquad \text{(A.5)}$$

$$\frac{\partial \boldsymbol{k}(X, \boldsymbol{x}_*)}{\partial \boldsymbol{x}_*} = \boldsymbol{k}(X, \ \boldsymbol{x}_*) \ \odot \ \tilde{X}_* \ \Lambda^{-1} \qquad \text{(A.6)}$$

where we define $\tilde{X}_*$ to be the $N \times D$ matrix, $[\boldsymbol{x}_1 - \boldsymbol{x}_*, \ \ldots, \ \boldsymbol{x}_N - \boldsymbol{x}_*]^T$. Note that $\boldsymbol{k}(X, \boldsymbol{x}_*)$ is a $N \times 1$ column vector. We have abused notation slightly: the element-wise product between a $N \times 1$ vector and a $N \times D$ matrix should be carried out by replicating the vector D times into a matching $N \times D$ matrix.

Building on equations A.3 and A.4 we can now find the second derivative (cross term),

$$\begin{aligned}
\frac{\partial^2 k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_1 \partial \boldsymbol{x}_2} &= \frac{\partial}{\partial \boldsymbol{x}_1}\left\{\Lambda^{-1}(\boldsymbol{x}_1 - \boldsymbol{x}_2)\, k(\boldsymbol{x}_1, \boldsymbol{x}_2)\right\} \\
&= \Lambda^{-1}\left(I - (\boldsymbol{x}_1 - \boldsymbol{x}_2)(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \Lambda^{-1}\right) k(\boldsymbol{x}_1, \boldsymbol{x}_2) \qquad \text{(A.7)}
\end{aligned}$$

a $D \times D$ matrix. We can see the following relationship,

$$\frac{\partial^2 k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_1^2} = \frac{\partial^2 k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_2^2} = -\frac{\partial^2 k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_1 \partial \boldsymbol{x}_2} = -\frac{\partial^2 k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_2 \partial \boldsymbol{x}_1} \qquad \text{(A.8)}$$

We can summarise the derivatives as follows,

$$\frac{\partial k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_2} = \Lambda^{-1}(\boldsymbol{x}_1 - \boldsymbol{x}_2)\, k(\boldsymbol{x}_1, \boldsymbol{x}_2) \qquad \text{(A.9)}$$

$$\frac{\partial^2 k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_2^2} = -\Lambda^{-1} k(\boldsymbol{x}_1, \boldsymbol{x}_2) + \Lambda^{-1}(\boldsymbol{x}_1 - \boldsymbol{x}_2)\frac{\partial k(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\partial \boldsymbol{x}_2}^T \qquad \text{(A.10)}$$

To find higher derivatives we need to switch to writing down elements of the derivative. First we

rephrase the first two derivatives before proceeding to the third and fourth,

$$\frac{\partial k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(i)}} \;=\; \Lambda_{ii}^{-1}\;\left(x_1^{(i)} - x_2^{(i)}\right)\; k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) \tag{A.11}$$

$$\frac{\partial^2 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(j)} \partial x_2^{(i)}} \;=\; -\Lambda_{ij}^{-1}\, k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) \;+\; \Lambda_{ii}^{-1}\;\left(x_1^{(i)} - x_2^{(i)}\right)\; \frac{\partial k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(j)}} \tag{A.12}$$

$$\frac{\partial^3 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(k)} \partial x_2^{(j)} \partial x_2^{(i)}} \;=\; -\Lambda_{ij}^{-1}\, \frac{\partial k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(k)}} \;-\; \Lambda_{ik}^{-1}\, \frac{\partial k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(j)}} \;+\; \Lambda_{ii}^{-1}\;\left(x_1^{(i)} - x_2^{(i)}\right)\; \frac{\partial^2 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(k)} \partial x_2^{(j)}} \tag{A.13}$$

$$\frac{\partial^4 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(l)} \partial x_2^{(k)} \partial x_2^{(j)} \partial x_2^{(i)}} \;=\; -\Lambda_{ij}^{-1}\, \frac{\partial^2 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(l)} \partial x_2^{(k)}} - \Lambda_{ik}^{-1}\, \frac{\partial^2 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(l)} \partial x_2^{(j)}} - \Lambda_{il}^{-1}\, \frac{\partial^2 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(k)} \partial x_2^{(j)}} + \Lambda_{ii}^{-1}\;\left(x_1^{(i)} - x_2^{(i)}\right)\; \frac{\partial^3 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(l)} \partial x_2^{(k)} \partial x_2^{(j)}} \tag{A.14}$$

We sometimes need to evaluate these derivatives for $\boldsymbol{x}_1 \;=\; \boldsymbol{x}_2$,

$$k((\boldsymbol{x}_1, \boldsymbol{x}_2)|_{\boldsymbol{x}_1 \;=\; \boldsymbol{x}_2} \;=\; \sigma_f^2 \tag{A.15}$$

$$\left.\frac{\partial k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial \boldsymbol{x}_2^{(i)}}\right|_{\boldsymbol{x}_1 \;=\; \boldsymbol{x}_2} \;=\; 0 \tag{A.16}$$

$$\left.\frac{\partial^2 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(j)} \partial x_2^{(i)}}\right|_{\boldsymbol{x}_1 \;=\; \boldsymbol{x}_2} \;=\; -\Lambda_{ij}^{-1}\, \sigma_f^2 \tag{A.17}$$

$$\left.\frac{\partial^3 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(k)} \partial x_2^{(j)} \partial x_2^{(i)}}\right|_{\boldsymbol{x}_1 \;=\; \boldsymbol{x}_2} \;=\; 0 \tag{A.18}$$

$$\left.\frac{\partial^4 k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)}{\partial x_2^{(l)} \partial x_2^{(k)} \partial x_2^{(j)} \partial x_2^{(i)}}\right|_{\boldsymbol{x}_1 \;=\; \boldsymbol{x}_2} \;=\; \Lambda_{ij}^{-1}\, \Lambda_{kl}^{-1}\, \sigma_f^2 \;+\; \Lambda_{ik}^{-1}\, \Lambda_{jl}^{-1}\, \sigma_f^2 \;+\; \Lambda_{il}^{-1}\, \Lambda_{jk}^{-1}\, \sigma_f^2 \tag{A.19}$$

A.2 Squared Exponential Kernel Moments

Many times throughout this thesis we have needed to find the output moments of a GP prediction given a Gaussian input point,

$$\boldsymbol{x} \;\sim\; \mathcal{N}(\boldsymbol{\mu},\; \Sigma) \tag{A.20}$$

To find these moments we need to first find the moments of the covariance function, which we will do in this section. First we introduce some notation: where the covariance function is written with three arguments, e.g. $k(\boldsymbol{x}_1,\, \boldsymbol{x}_2,\, \Lambda)$, the third argument refers to the lengthscale matrix. Also, we will define some shorthand as we proceed using the capital letters E, V, and C for expectations, variances, and covariances. In these cases subscripts refer to the variables those moments are taken w.r.t. and not indicies, e.g. $E_{xk} \;=\; \mathbb{E}[\boldsymbol{x}\, k(\boldsymbol{x},\, \boldsymbol{x}_1)]$.

We first note that the SE kernel can be written in terms of a Gaussian density function,

$$k(\boldsymbol{x},\, \boldsymbol{x}_1) \;=\; \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_1)^T \Lambda^{-1} (\boldsymbol{x}-\boldsymbol{x}_1)\right) \tag{A.21}$$

$$\;=\; \sigma_f^2 \,(2\pi)^{D/2}\, |\Lambda|^{1/2}\, \mathcal{N}(\boldsymbol{x};\, \boldsymbol{x}_1,\, \Lambda) \tag{A.22}$$

and that the product of two Gaussian distributions is another Gaussian,

$$\mathcal{N}(\boldsymbol{x};\; \boldsymbol{\mu}_1,\; \Sigma_1)\mathcal{N}(\boldsymbol{x};\; \boldsymbol{\mu}_2,\; \Sigma_2) \;=\; Z\,\mathcal{N}(\boldsymbol{x};\; M,\; S) \tag{A.23}$$

$$M \;=\; (\Sigma_1^{-1}+\Sigma_2^{-1})^{-1}(\Sigma_1^{-1}\boldsymbol{\mu}_1+\Sigma_2^{-1}\boldsymbol{\mu}_2) \tag{A.24}$$

$$S \;=\; (\Sigma_1^{-1}+\Sigma_2^{-1})^{-1} \tag{A.25}$$

$$Z \;=\; (2\pi)^{-D/2}\, |\Sigma_1+\Sigma_2|^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^T(\Sigma_1+\Sigma_2)^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)\right) \tag{A.26}$$

$$\;=\; \sigma_f^{-2}\,(2\pi)^{-D/2}\, |\Sigma_1+\Sigma_2|^{-1/2}\, k(\boldsymbol{\mu}_1,\, \boldsymbol{\mu}_2,\, \Sigma_1+\Sigma_2) \tag{A.27}$$

where M, S, and Z are functions of $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, Σ_1, and Σ_2. Hence the product of two squared exponential kernels is,

$$k(\boldsymbol{x},\boldsymbol{x}_1)\, k(\boldsymbol{x},\boldsymbol{x}_2) \;=\; k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right) k\left(\boldsymbol{x},\frac{\boldsymbol{x}_1+\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right) \tag{A.28}$$

$$\;=\; \sigma_f^2\,\pi^{D/2}\,|\Lambda|^{1/2}\, k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right) \mathcal{N}(\boldsymbol{x};\, (\boldsymbol{x}_1+\boldsymbol{x}_2)/2,\, \Lambda/2) \tag{A.29}$$

From these starting points we can find the expected value of the kernel,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{x}}[k(\boldsymbol{x},\boldsymbol{x}_1)] \;&=\; \int k(\boldsymbol{x},\, \boldsymbol{x}_1)\, p(\boldsymbol{x})\; d\boldsymbol{x} \\
&=\; \sigma_f^2\,(2\pi)^{D/2}\,|\Lambda|^{1/2} \int \mathcal{N}(\boldsymbol{x};\; \boldsymbol{x}_1,\, \Lambda)\, \mathcal{N}(\boldsymbol{x},\mu,\Sigma)\; d\boldsymbol{x} \\
&=\; \sigma_f^2\,(2\pi)^{D/2}\,|\Lambda|^{1/2}\, Z \\
&=\; |\Lambda|^{1/2}\,|\Lambda+\Sigma|^{-1/2}\, k(\boldsymbol{\mu},\, \boldsymbol{x}_1,\, \Lambda+\Sigma) \\
&=\; |\Sigma\,\Lambda^{-1}+I|^{-1/2}\, k(\boldsymbol{\mu},\, \boldsymbol{x}_1,\, \Lambda+\Sigma) \qquad \text{(A.30)} \\
&\triangleq\; E_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda) \qquad (1\times 1) \qquad \text{(A.31)}
\end{aligned}$$

which is the same form as the SE covariance function except with $\boldsymbol{\mu}$ replacing $\boldsymbol{x}$, the lengthscales expanded by Σ, and an additional scaling factor. We now find the mean of a product of kernels,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{x}}[k(\boldsymbol{x}_1,\boldsymbol{x})\, k(\boldsymbol{x},\boldsymbol{x}_2)] \;&=\; k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right) \mathbb{E}_x\left[k\left(\boldsymbol{x},\frac{\boldsymbol{x}_1+\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)\right] \\
&=\; k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right) E_k\left(\frac{(\boldsymbol{x}_1+\boldsymbol{x}_2)}{2},\, \boldsymbol{\mu}_1,\, \Sigma,\; \frac{\Lambda}{2}\right) \qquad \text{(A.32)} \\
&\triangleq\; E_{kk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma) \qquad (1\times 1) \qquad \text{(A.33)}
\end{aligned}$$

Therefore, the variance of a kernel and the covariance of two kernels,

$$\begin{aligned}
\mathbb{V}_{\boldsymbol{x}}[k(\boldsymbol{x},\boldsymbol{x}_1)] &= E_{kk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_1,\Sigma) - E_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda)\,E_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda) && \text{(A.34)}\\
&\triangleq V_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma) \qquad (1\times 1) && \text{(A.35)}
\end{aligned}$$

$$\begin{aligned}
\mathbb{C}_{\boldsymbol{x}}[k(\boldsymbol{x},\boldsymbol{x}_1),\,k(\boldsymbol{x},\boldsymbol{x}_2)] &= E_{kk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma) - E_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda)\,E_k(\boldsymbol{\mu},\boldsymbol{x}_2,\Sigma,\Lambda) && \text{(A.36)}\\
&\triangleq C_{kk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma) \qquad (1\times 1) && \text{(A.37)}
\end{aligned}$$

The mean of $\boldsymbol{x}$ times a kernel,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{x}}\left[\boldsymbol{x}\,k(\boldsymbol{x},\boldsymbol{x}_1)\right] &= \int \boldsymbol{x}\,k(\boldsymbol{x},\boldsymbol{x}_1)\,p(\boldsymbol{x})\,d\boldsymbol{x}\\
&= \sigma_f^2\,(2\pi)^{D/2}\,|\Lambda|^{1/2}\int \boldsymbol{x}\,\mathcal{N}\left(\boldsymbol{x};\,\boldsymbol{x}_1,\,\Lambda\right)\,\mathcal{N}\left(\boldsymbol{x};\,\boldsymbol{\mu},\,\Sigma\right)\,d\boldsymbol{x}\\
&= \sigma_f^2\,(2\pi)^{D/2}\,|\Lambda|^{1/2}\,Z\,M\\
&= E_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda)\,M\\
&= E_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda)\,\Lambda\,(\Lambda+\Sigma)^{-1}(\Sigma\,\Lambda^{-1}\boldsymbol{x}_1+\boldsymbol{\mu}) && \text{(A.38)}\\
&\triangleq E_{xk}(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda) \qquad (D\times 1) && \text{(A.39)}
\end{aligned}$$

and $\boldsymbol{x}$ times a product of kernels,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{x}}\left[\boldsymbol{x}\,k(\boldsymbol{x},\boldsymbol{x}_1)\,k(\boldsymbol{x},\boldsymbol{x}_2)\right] &= k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)\mathbb{E}_x\left[\boldsymbol{x}\,k\left(\boldsymbol{x},\frac{\boldsymbol{x}_1+\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)\right]\\
&= k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)E_{xk}\left(\boldsymbol{\mu},\,\frac{\boldsymbol{x}_1+\boldsymbol{x}_2}{2},\,\Sigma,\,\frac{\Lambda}{2}\right)\\
&\triangleq E_{xkk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma,\Lambda) \qquad (D\times 1) && \text{(A.40)}
\end{aligned}$$

which leads us to the covariance between x times a covariance function and another covariance function,

$$\begin{aligned}
\mathbb{C}_{\boldsymbol{x}}\left[\boldsymbol{x}\,k(\boldsymbol{x},\,\boldsymbol{x}_1),\;k(\boldsymbol{x},\,\boldsymbol{x}_2)\right] &= E_{xkk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma,\Lambda) - E_{xk}(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda)\,E_k(\boldsymbol{\mu},\boldsymbol{x}_1,\Sigma,\Lambda)\\
&\triangleq C_{xkk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma) \qquad (D\times 1) && \text{(A.41)}
\end{aligned}$$

The mean of a quadratic product in $\boldsymbol{x}$ and the kernel,

$$
\begin{aligned}
&\mathbb{E}\left[\boldsymbol{x}\,\boldsymbol{x}^T k(\boldsymbol{x},\boldsymbol{x}_1)\,k(\boldsymbol{x},\boldsymbol{x}_2)\right] \\
&= k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)\int \boldsymbol{x}\,\boldsymbol{x}^T\,k\left(\boldsymbol{x},\frac{\boldsymbol{x}_1+\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)p(\boldsymbol{x})\;d\boldsymbol{x} \\
&= \sigma_f^2\,(2\pi)^{D/2}\,|\Lambda|^{1/2}\,k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)\int \boldsymbol{x}\,\boldsymbol{x}^T\,\mathcal{N}\left(\boldsymbol{x};\,\frac{\boldsymbol{x}_1+\boldsymbol{x}_2}{2},\,\frac{\Lambda}{2}\right)\mathcal{N}(\boldsymbol{x};\;\boldsymbol{\mu},\;\Sigma)\;d\boldsymbol{x} \\
&= \sigma_f^2\,(2\pi)^{D/2}\,|\Lambda|^{1/2}\,k\left(\frac{\boldsymbol{x}_1}{2},\frac{\boldsymbol{x}_2}{2},\frac{\Lambda}{2}\right)Z\,\left(S\;+\;M\,M^T\right) \\
&= E_{kk}\,(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma)\,\frac{\Lambda}{2}\left(\Sigma+\frac{\Lambda}{2}\right)^{-1}\left(\left(2\,\Lambda^{-1}\Sigma+I\right)+\;\left(2\,\Sigma\,\Lambda^{-1}\boldsymbol{x}_1+\boldsymbol{\mu}\right)\left(2\,\Sigma\,\Lambda^{-1}\boldsymbol{x}_1+\boldsymbol{\mu}\right)^T\right)\left(\frac{\Lambda}{2}+\Sigma\right)^{-1}\frac{\Lambda}{2} \qquad \text{(A.42)} \\
&\triangleq E_{xxkk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma,\Lambda) \qquad (D\times D) \qquad \text{(A.43)}
\end{aligned}
$$

Finally the variance and covariance of x times a covariance function,

$$
\mathbb{V}_{\boldsymbol{x}}[\boldsymbol{x}\,k(\boldsymbol{x},\boldsymbol{x}_1)] \;=\; E_{xxkk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_1,\Sigma)\;-\;E_{xk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_1,\Sigma)\,E_{xk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_1,\Sigma,\Lambda)^T \qquad \text{(A.44)}
$$

$$
\begin{aligned}
\mathbb{C}_{\boldsymbol{x}}\left[\boldsymbol{x}\,k(\boldsymbol{x},\,\boldsymbol{x}_1),\;\boldsymbol{x}\,k(\boldsymbol{x},\,\boldsymbol{x}_2)\right] \;&=\; E_{xxkk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma)\;-\;E_{xk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma)\,E_{xk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma,\Lambda)^T \qquad \text{(A.45)} \\
&\triangleq C_{xkxk}(\boldsymbol{\mu},\boldsymbol{x}_1,\boldsymbol{x}_2,\Sigma) \qquad (D\times D) \qquad \text{(A.46)}
\end{aligned}
$$

A.3 Derivatives with Uncertain Inputs

Although we did not use this in this chapter, we can also ask what the distribution over the derivatives of a GP posterior is when the input location is Gaussian distributed, i.e.,

$$
\boldsymbol{x}_*\;\sim\;\mathcal{N}\left(\boldsymbol{\mu},\;\Sigma\right) \qquad \text{(A.47)}
$$

We can use the rule of iterated expectations to find the mean derivative,

$$
\begin{aligned}
\mathbb{E}\left[\frac{\partial f_*}{\partial \boldsymbol{x}_*}\right] \;&=\; \mathbb{E}_{\boldsymbol{x}_*}\left[\mathbb{E}_{f_*}\left[\frac{\partial f_*}{\partial \boldsymbol{x}_*}\right]\right] \\
&=\; \mathbb{E}_{\boldsymbol{x}_*}\left[\frac{\partial \bar{f}_*}{\partial \boldsymbol{x}_*}\right] \\
&=\; \mathbb{E}_{\boldsymbol{x}_*}\left[-\Lambda^{-1}\,\tilde{X}_*\,\left(\boldsymbol{k}(\boldsymbol{x}_*,X)^T\odot\boldsymbol{\beta}\right)\right] \\
&=\; -\Lambda^{-1}\,\mathbb{E}_{\boldsymbol{x}_*}\left[\sum_{i=1}^{N}\beta_i\,(\boldsymbol{x}_*\;-\boldsymbol{x}_i)\,k(\boldsymbol{x}_*,\boldsymbol{x}_i)\right] \\
&=\; -\Lambda^{-1}\sum_{i=1}^{N}\;\beta_i\,\mathbb{E}_{\boldsymbol{x}_*}\left[(\boldsymbol{x}_*\;-\boldsymbol{x}_i)\,k(\boldsymbol{x}_*,\boldsymbol{x}_i)\right] \\
&=\; -\Lambda^{-1}\sum_{i=1}^{N}\;\beta_i\,\mathbb{E}_{\boldsymbol{x}_*}\left[\boldsymbol{x}_*\,k(\boldsymbol{x}_*,\boldsymbol{x}_i)\right]\;-\;\beta_i\,\boldsymbol{x}_i\mathbb{E}_{\boldsymbol{x}_*}\left[k(\boldsymbol{x}_*,\boldsymbol{x}_i)\right] \\
&=\; -\Lambda^{-1}\sum_{i=1}^{N}\;\beta_i\,E_{xk}(\boldsymbol{\mu},\boldsymbol{x}_i,\Sigma)\;-\;\beta_i\,\boldsymbol{x}_i\,E_k(\boldsymbol{\mu},\boldsymbol{x}_i,\Sigma) \qquad \text{(A.48)}
\end{aligned}
$$

We can use the rule of total variance to find the variance of the derivative,

$$\begin{aligned}
\mathbb{V}\left[\frac{\partial f_*}{\partial \boldsymbol{x}_*}\right] &= \mathbb{E}_{\boldsymbol{x}_*}\left[\mathbb{V}_{f_*}\left[\frac{\partial f_*}{\partial \boldsymbol{x}_*}\right]\right] + \mathbb{V}_{\boldsymbol{x}_*}\left[\mathbb{E}_{f_*}\left[\frac{\partial f_*}{\partial \boldsymbol{x}_*}\right]\right] \\
&= \mathbb{E}_{\boldsymbol{x}_*}\left[\frac{\partial^2 k(\boldsymbol{x}_1^*,\, \boldsymbol{x}_2^*)}{\partial \boldsymbol{x}_1^*\, \partial \boldsymbol{x}_2^*} - \frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X)}{\partial \boldsymbol{x}_*}\, K^{-1}\, \frac{\partial \boldsymbol{k}(X,\, \boldsymbol{x}_*)}{\partial \boldsymbol{x}_*}\right] + \mathbb{V}_{\boldsymbol{x}_*}\left[\frac{\partial \bar{f}_*}{\partial \boldsymbol{x}_*}\right] \\
&= \mathbb{E}_{\boldsymbol{x}_*}\left[\frac{\partial^2 k(\boldsymbol{x}_1^*,\, \boldsymbol{x}_2^*)}{\partial \boldsymbol{x}_1^*\, \partial \boldsymbol{x}_2^*} - \frac{\partial \boldsymbol{k}(\boldsymbol{x}_*,\, X)}{\partial \boldsymbol{x}_*}\, K^{-1}\, \frac{\partial \boldsymbol{k}(X,\, \boldsymbol{x}_*)}{\partial \boldsymbol{x}_*}\right] + \mathbb{V}_{\boldsymbol{x}_*}\left[-\Lambda^{-1}\, \tilde{X}_*^T\, \left(\boldsymbol{k}(\boldsymbol{x}_*, X)^T \odot \boldsymbol{\beta}\right)\right]
\end{aligned} \tag{A.49}$$

We will calculate these expectations separately. Firstly the expectation of the second derivative of the kernel (see section A.1 for derivation of the second derivative),

$$\begin{aligned}
\mathbb{E}\left[\frac{\partial^2 k(\boldsymbol{x}_1^*,\, \boldsymbol{x}_2^*)}{\partial \boldsymbol{x}_1^*\, \partial \boldsymbol{x}_2^*}\right] &= \mathbb{E}\left[\Lambda^{-1}\; k\left(\boldsymbol{x}_*, \boldsymbol{x}_*\right)\right] \\
&= \Lambda^{-1}\, \sigma_f^2
\end{aligned} \tag{A.50}$$

$$\begin{aligned}
\mathbb{E}\left[\frac{\partial \boldsymbol{k}(\boldsymbol{x}_*, X)}{\partial x_i^*}\, K^{-1}\, \frac{\partial \boldsymbol{k}(X,\, \boldsymbol{x}_*)}{\partial x_j^*}\right] &= \Lambda_i^{-1}\; \mathbb{E}\left[[\tilde{X}_*^{(i)T} \;\odot\; \boldsymbol{k}\left(\boldsymbol{x}_*\,, X\right)]\; K^{-1}\; [\boldsymbol{k}\left(X, \boldsymbol{x}_*\right) \;\odot\; \tilde{X}_*^{(j)}]\right]\; \Lambda_j^{-1} \\
&= \Lambda_i^{-1}\; \Big(\mathbb{E}[\tilde{X}_*^{(i)T} \odot \boldsymbol{k}\left(\boldsymbol{x}_*\,, X\right)]\; K^{-1}\; \mathbb{E}[\boldsymbol{k}\left(X, \boldsymbol{x}_*\right) \odot \tilde{X}_*^{(j)}] \\
&\qquad + \;\operatorname{tr}\Big(K^{-1}\, \mathbb{C}[\tilde{X}_*^{(i)T} \odot \boldsymbol{k}\left(\boldsymbol{x}_*\,, X\right),\; \tilde{X}_*^{(j)T} \odot \boldsymbol{k}\left(\boldsymbol{x}_*\,, X\right)]\Big)\Big)\; \Lambda_j^{-1}
\end{aligned} \tag{A.51}$$

where,

$$\tilde{X}_*^{(i)} = [x_1^{(i)} - x_*^{(i)},\; \ldots,\; x_N^{(i)} - x_*^{(i)}]^T \tag{A.52}$$

which is a $N \times 1$ vector.

$$\begin{aligned}
\mathbb{E}_{x_*}[(x_1^{(i)} - x_*^{(i)})\, k\left(\boldsymbol{x}_1,\; \boldsymbol{x}_*\right)] &= x_1^{(i)}\, \mathbb{E}_k(\boldsymbol{\mu},\, \boldsymbol{x}_1,\, \Sigma) \;-\; \mathbb{E}_{xk}(\boldsymbol{\mu},\, \boldsymbol{x}_1,\, \Sigma) \\
&= E_k(\boldsymbol{\mu},\, \boldsymbol{x}_1,\, \Sigma)\, (x_1^{(i)} - \mu_i)\, (\Sigma_{ii}\, \Lambda_i^{-1} + 1)^{-1}
\end{aligned} \tag{A.53}$$

Defining $\boldsymbol{U}$ to be a $N \times 1$ vector with elements, $U_k = x_k^{(i)} - \mu_i$,

$$\mathbb{E}_{x_*}[\tilde{X}_*^{(i)} \;\odot\; k\left(X,\; \boldsymbol{x}_*\right)] = E_k(X,\, \boldsymbol{\mu},\, \Sigma) \odot \boldsymbol{U}\, (\Sigma_{ii}\, \Lambda_i^{-1} + 1)^{-1} \tag{A.54}$$

which is a $N \times 1$ vector.

$$\begin{aligned}
&\mathbb{C}_{x_*}[(x_1^{(i)} - x_*^{(i)})\, k\left(\boldsymbol{x}_1,\; \boldsymbol{x}_*\right),\; (x_2^{(j)} - x_*^{(j)})\, k\left(\boldsymbol{x}_2,\; \boldsymbol{x}_*\right)] \\
&\quad= x_1^{(i)}\, x_2^{(j)}\, \mathbb{C}_{x_*}[k\left(\boldsymbol{x}_1,\; \boldsymbol{x}_*\right),\; k\left(\boldsymbol{x}_2,\; \boldsymbol{x}_*\right)] \;-\; x_1^{(i)} \mathbb{C}_{x_*}[\, k\left(\boldsymbol{x}_1,\; \boldsymbol{x}_*\right),\; x_*^{(j)}\, k\left(\boldsymbol{x}_2,\; \boldsymbol{x}_*\right)] \\
&\qquad\quad- \;x_2^{(j)}\, \mathbb{C}_{x_*}[x_*^{(i)}\, k\left(\boldsymbol{x}_1,\; \boldsymbol{x}_*\right),\; k\left(\boldsymbol{x}_2,\; \boldsymbol{x}_*\right)] \;+\; \mathbb{C}_{x_*}[x_*^{(i)}\, k\left(\boldsymbol{x}_1,\; \boldsymbol{x}_*\right),\; x_*^{(j)}\, k\left(\boldsymbol{x}_2,\; \boldsymbol{x}_*\right)] \\
&\quad= x_1^{(i)}\, x_2^{(j)}\, C_{kk}(\boldsymbol{\mu}, \boldsymbol{x}_1, \boldsymbol{x}_2, \Sigma) - x_1^{(i)}\, C_{xkk}^{(j)}(\boldsymbol{\mu}, \boldsymbol{x}_2, \boldsymbol{x}_1, \Sigma) - x_2^{(j)}\, C_{xkk}^{(i)}(\boldsymbol{\mu}, \boldsymbol{x}_1, \boldsymbol{x}_2, \Sigma) + C_{xkxk}^{(ij)}(\boldsymbol{\mu}, \boldsymbol{x}_1, \boldsymbol{x}_2, \Sigma)
\end{aligned} \tag{A.55}$$

The C terms are derived and defined in section A.2. Therefore,

$$\begin{aligned}
&\mathbb{C}[\tilde{X}_*^{(i)T} \odot \boldsymbol{k}\left(\boldsymbol{x}_* \,, X\right),\ \tilde{X}_*^{(j)T} \odot \boldsymbol{k}\left(\boldsymbol{x}_* \,, X\right)] \\
&\quad = \ (X^{(i)}\, X^{(j)T}) \odot C_{kk}(\boldsymbol{\mu}, X, X, \Sigma) - X^{(i)} \odot C^{(j)}_{xkk}(\boldsymbol{\mu}, X, X, \Sigma) - X^{(j)T} \odot C^{(i)}_{xkk}(\boldsymbol{\mu}, X, X, \Sigma) + C^{(ij)}_{xkxk}(\boldsymbol{\mu}, X, X, \Sigma)
\end{aligned} \tag{A.56}$$

which is a $N \times N$ matrix.

Finally we need, $\mathbb{V}_{\boldsymbol{x}_*}\left[\Lambda^{-1}\, \tilde{X}_*^T\ (\boldsymbol{k}(X, \boldsymbol{x}_*) \odot \boldsymbol{\beta})\right]$, which we will break up and compute as,

$$\begin{aligned}
&\mathbb{C}_{\boldsymbol{x}_*}\left[\Lambda_i^{-1}\, \tilde{X}_*^{(i)T}\ (\boldsymbol{k}(X, \boldsymbol{x}_*) \odot \boldsymbol{\beta}),\ \Lambda_j^{-1}\, \tilde{X}_*^{(j)T}\ (\boldsymbol{k}(X, \boldsymbol{x}_*) \odot \boldsymbol{\beta})\right] \\
&\quad = \ \Lambda_i^{-1}\, \Lambda_j^{-1}\, \mathbb{C}_{\boldsymbol{x}_*}\left[\sum_k (x_k^{(i)} - x_*^{(i)})\, k(\boldsymbol{x}_k, \boldsymbol{x}_*)\, \beta_k,\ \sum_l (x_l^{(j)} - l_*^{(j)})\, k(\boldsymbol{x}_l, \boldsymbol{x}_*)\, \beta_l\right] \\
&\quad = \ \Lambda_i^{-1}\, \Lambda_j^{-1} \sum_k \sum_l \beta_k\, \beta_l \left(x_k^{(i)}\, x_l^{(j)}\, C_{kk} - x_k^{(i)}\, C^{(j)}_{xkk} - x_l^{(j)}\, C^{(i)}_{xkk} + C^{(ij)}_{xkxk}\right) \\
&\quad = \ \Lambda_i^{-1}\, \Lambda_j^{-1}\, \boldsymbol{\beta}_k^T \left(X^{(i)}\, X^{(j)T} \odot C_{kk} - X^{(i)}\, C^{(j)}_{xkk} - X^{(j)T}\, C^{(i)}_{xkk} + C^{(ij)}_{xkxk}\right) \boldsymbol{\beta}_l
\end{aligned} \tag{A.57}$$

Bibliography

Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle Markov chain Monte Carlo methods. *JRSS: Series B*, 72(3):269–342.

Åström, K. J. and Furuta, K. (2000). Swinging up a pendulum by energy control. *Automatica*, 36(2):287–295.

Åström, K. J. and Wittenmark, B. (1973). On self tuning regulators. *Automatica*, 9(2):185–199.

Atkeson, C. G. and Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *International Conference on Robotics and Automation*, pages 3557–3564. IEEE Press.

Attias, H. (1999). Inferring parameters and structure of latent variable models by variational bayes. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 21–30. Morgan Kaufmann Publishers Inc.

Bai, E.-W. (1998). An optimal two-stage identification algorithm for Hammerstein–Wiener nonlinear systems. *Automatica*, 34(3):333–338.

Barber, D. and Schottky, B. (1998). Radial basis functions: a bayesian treatment. *Advances in Neural Information Processing Systems*, pages 402–408.

Bischoff, B., Nguyen-Tuong, D., van Hoof, H., McHutchon, A., Rasmussen, C. E., Knoll, A., Peters, J., and Deisenroth, M. P. (2014). Policy search for learning robot control using sparse data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Booth, J. G. and Hobert, J. P. (1999). Maximizing generalized linear mixed model likelihoods with an automated monte carlo em algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(1):265–285.

Cappé, O., Moulines, E., and Rydén, T. (2005). *Inference in hidden Markov models*. Springer.

Chan, K. and Ledolter, J. (1995). Monte carlo em estimation for time series models involving counts. *Journal of the American Statistical Association*, 90(429):242–252.

Dallaire, P., Besse, C., and Chaib-draa, B. (2008). Learning Gaussian Process Models from Uncertain Data. *16th International Conference on Neural Information Processing*.

Damianou, A. C., Titsias, M. K., and Lawrence, N. D. (2011). Variational gaussian process dynamical systems. In *NIPS*, pages 2510–2518.

Deisenroth, M. P. (2009). *Efficient reinforcement learning using Gaussian Processes*. KIT Scientific Publishing.

Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2014). Gaussian Processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 36(5).

Deisenroth, M. P., Huber, M. F., and Hanebeck, U. D. (2009). Analytic moment-based Gaussian Process filtering. In *ICML 26*, pages 225–232.

Deisenroth, M. P. and Mohamed, S. (2012). Expectation propagation in Gaussian Process dynamical systems. *Advances in Neural Information Processing Systems 25*.

Deisenroth, M. P. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *ICML 28*.

Delyon, B., Lavielle, M., and Moulines, E. (1999). Convergence of a stochastic approximation version of the em algorithm. *Annals of Statistics*, pages 94–128.

Douc, R. and Cappé, O. (2005). Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69. IEEE.

Douc, R., Moulines, E., Olsson, J., et al. (2009). Optimality of the auxiliary particle filter. *Probability and Mathematical Statistics*, 29(1):1–28.

Doucet, A. (2001). *Sequential monte carlo methods*. Wiley Online Library.

Doucet, A. and Johansen, A. M. (2009). A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704.

Feldbaum, A. (1960). Dual control theory. *Automation and Remote Control*, 21(9):874–1039.

Ferris, B., Fox, D., and Lawrence, N. (2007). WiFi-SLAM using Gaussian Process latent variable models. In *IJCAI 20*, pages 2480–2485.

Frigola, R., Chen, Y., and Rasmussen, C. E. (2014). Variational Gaussian Process state space models.

Frigola, R., Lindsten, F., Schön, T., and Rasmussen, C. (2013). Bayesian inference and learning in GP state-space models with PMCMC. *Advances in Neural Information Processing Systems 26*.

Furuta, K., Yamakita, M., and Kobayashi, S. (1991). Swing up control of inverted pendulum. In *Industrial Electronics, Control and Instrumentation, 1991. Proceedings. IECON'91., 1991 International Conference on*, pages 2193–2198. IEEE.

Gal, Y., van der Wilk, M., and Rasmussen, C. E. (2014). Distributed variational inference in sparse gaussian process regression and latent variable models. *Workshop on New Learning Models and Frameworks for Big Data, ICML*.

Ghahramani, Z. and Hinton, G. (1996). Parameter estimation for linear dynamical systems. Technical report, Technical Report CRG-TR-96-2, University of Toronto.

Ghahramani, Z. and Roweis, S. (1999). Learning nonlinear dynamical systems using an EM algorithm. In *Advances in Neural Information Processing Systems 11*, pages 431–437.

Girard, A. and Murray-Smith, R. (2003). Learning a gaussian process model with uncertain inputs.

Girard, A., Rasmussen, C. E., Quinonero-Candela, J., and Murray-Smith, R. (2003). Gaussian Process priors with uncertain inputs—application to multiple-step ahead time series forecasting. *Advances in Neural Information Processing Systems 16*.

Goldberg, P. W., Williams, C. K. I., and Bishop, C. M. (1998). Regression with input-dependent noise: A Gaussian Process treatment. *Advances in Neural Information Processing Systems-98*.

Gordon, N. J., Salmond, D. J., and Smith, A. F. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET.

Hall, J. (2013). *Machine learning for control: incorporating prior knowledge*.

Hunter, I. and Korenberg, M. (1986). The identification of nonlinear biological systems: Wiener and hammerstein cascade models. *Biological Cybernetics*, 55(2-3):135–144.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233.

Julier, S. and Uhlmann, J. (1997). New extension of the Kalman filter to nonlinear systems. In *AeroSense*, pages 182–193. SPIE.

Kalman, R. (1958). Design of a self-optimising control system. *Trans. ASME*, 80:468–478.

Kalman, R. E. (1963). Mathematical description of linear dynamical systems. *Journal of the Society for Industrial & Applied Mathematics, Series A: Control*, 1(2):152–192.

Kantas, N., Doucet, A., Singh, S., and Maciejowski, J. (2009). An overview of sequential Monte Carlo methods for parameter estimation in general state-space models. In *Sysid 15*, volume 15, pages 774–785.

Kaufman, H., Barkana, I., and Sobel, K. (1998). *Direct adaptive control algorithms: theory and applications*. Springer.

Kersting, K., Plagemann, C., Pfaff, P., and Burgard, W. (2007). Most likely heteroscedastic Gaussian Process regression. *ICML-07*.

Kim, H., Jordan, M. I., Sastry, S., and Ng, A. Y. (2003). Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*.

Ko, J. and Fox, D. (2009). GP-BayesFilters: Bayesian filtering using Gaussian Process prediction and observation models. *Autonomous Robots*, 27:75–90.

Ko, J. and Fox, D. (2011). Learning GP-BayesFilters via Gaussian Process latent variable models. *Autonomous Robots*, 30:3–23.

Ko, J., Klein, D., Fox, D., and Haehnel, D. (2007). GP-UKF: Unscented Kalman filters with GP and observation models. pages 1901–1907.

Landau, I. D., Lozano, R., M'Saad, M., and Karimi, A. (2011). *Adaptive control: algorithms, analysis and applications*. Springer.

Lawrence, N. D. (2004). Gaussian process latent variable models for visualisation of high dimensional data. *Advances in Neural Information Processing Systems 16*, pages 329–336.

Lawrence, N. D. and Moore, A. J. (2007). Hierarchical Gaussian process latent variable models. In *ICML 24*, pages 481–488. ACM.

Le, Q. V., Smola, A. J., and Canu, S. (2005). Heteroscedastic Gaussian Process regression. *Procedings of ICML-05*, pages 489–496.

Levine, R. A. and Casella, G. (2001). Implementations of the monte carlo em algorithm. *Journal of Computational and Graphical Statistics*, 10(3):422–439.

Lindsten, F. (2013a). An efficient stochastic approximation em algorithm using conditional particle filters. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6274–6278.

Lindsten, F. (2013b). An efficient stochastic approximation EM algorithm using conditional particle filters. In *ICASSP 38*, pages 6274–6278.

Lindsten, F. (2013c). *Particle filters and Markov chains for learning of dynamical systems*.

Lindsten, F., Jordan, M. I., and Schön, T. B. (2012). Ancestor sampling for particle Gibbs. *Advances in Neural Information Processing Systems 25*, pages 2600–2608.

Lindsten, F. and Schön, T. B. (2013). Backward simulation methods for Monte Carlo statistical inference. *Found. Tr. Mach. Learn.*, 6(1).

Ljung, L. (1998). *System identification*. Springer.

Maciejowski, J. M. (2002). *Predictive control: with constraints*. Pearson education.

MacKay, D. J. (1998). Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166.

Maxwell, J. C. (1867). On governors. *Proceedings of the Royal Society of London*, 16:270–283.

McHutchon, A. and Rasmussen, C. E. (2010). Machine learning for control. In *Cambridge University Engineering Department Masters Projects*.

McHutchon, A. and Rasmussen, C. E. (2011). Gaussian process training with input noise. *Advances in Neural Information Processing Systems*.

Melkumyan, A. and Ramos, F. (2011). Multi-kernel Gaussian Processes. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 1408–1413. AAAI Press.

Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. In *UAI 17*, pages 362–369.

Narendra, K. S. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *Neural Networks, IEEE Transactions on*, 1(1):4–27.

Nelson, A. T. (2000). *Nonlinear estimation and modeling of noisy time series by dual Kalman filtering methods*. Oregon Graduate Institute of Science and Technology.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. Springer.

Ng, A. Y. and Jordan, M. (2000). PEGASUS: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc.

O'Hagan, A. and Kingman, J. (1978). Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–42.

Olsson, H., Åström, K. J., Canudas de Wit, C., Gäfvert, M., and Lischinsky, P. (1998). Friction models and friction compensation. *European journal of control*, 4(3):176–195.

Olsson, J., Cappé, O., Douc, R., Moulines, E., et al. (2008). Sequential monte carlo smoothing with application to parameter estimation in nonlinear state space models. *Bernoulli*, 14(1):155–179.

Parks, P. (1966). Liapunov redesign of model reference adaptive control systems. *Automatic Control, IEEE Transactions on*, 11(3):362–367.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Pitt, M. and Shephard, N. (1999). Filtering via simulation: Auxiliary particle filters. *JASA*, 94(446):590–599.

Poyiadjis, G., Doucet, A., and Singh, S. (2011). Particle approximations of the score and observed information matrix in state space models with application to parameter estimation. *Biometrika*, 98:65–80.

Queiro, R., Douglass, A., and Rasmussen, C. E. (2011). Machine learning for control. In *Cambridge University Engineering Department Masters Projects*.

Rakitsch, B., Lippert, C., Borgwardt, K., and Stegle, O. (2013). It is all in the noise: Efficient multi-task Gaussian process inference with structured residuals. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 26*, pages 1466–1474.

Rasmussen, C. E. and Deisenroth, M. P. (2008). Probabilistic inference for fast learning in control. In *Recent Advances in Reinforcement Learning*, pages 229–242. Springer.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Roweis, S. and Ghahramani, Z. Learning nonlinear dynamical systems using the expectation-maximization algorithm. In Haykin, S., editor, *Kalman filtering and neural networks*.

Rubin, D. B. (1987). The calculation of posterior distributions by data augmentation: Comment: A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest: The SIR algorithm. *Journal of the American Statistical Association*, pages 543–546.

Schön, T. B., Wills, A., and Ninness, B. (2011). System identification of nonlinear state-space models. *Automatica*, 47(1):39 – 49.

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., and

Juditsky, A. (1995). Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724.

Snelson, E. and Ghahramani, Z. (2006a). Sparse Gaussian Processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*.

Snelson, E. and Ghahramani, Z. (2006b). Variable noise and dimensionality reduction for sparse Gaussian Processes. *Procedings of UAI-06*.

Solak, E., Murray-Smith, R., Leithead, W., Leith, D., and Rasmussen, C. (2003). Derivative observations in Gaussian Process models of dynamic systems. *Advances in Neural Information Processing Systems-03*, pages 1033–1040.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*. MIT Press.

Sutton, R. S., Barto, A. G., and Williams, R. J. (1992). Reinforcement learning is direct adaptive optimal control. *Control Systems, IEEE*, 12(2):19–22.

Tesauro, G. (1995). Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–68.

Titsias, M. and Lawrence, N. (2010). Bayesian gaussian process latent variable model. *AISTATS 13*.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian Processes. In *AISTATS*, pages 567–574.

Turner, R., Deisenroth, M. P., and Rasmussen, C. E. (2010). State-space inference and learning with Gaussian Processes. In *AISTATS 13*.

Wan, E. and Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In *AS-SPCC*, pages 153–158.

Wan, E. A., Van Der Merwe, R., and Nelson, A. T. (1999). Dual estimation and the unscented transformation. In *NIPS*, pages 666–672.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2008). Gaussian Process dynamical models for human motion. *TPAMI*, 30:283–298.

Wei, G. C. and Tanner, M. A. (1990). A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704.

Whitaker, H. P., Yamron, J., and Kezer, A. (1958). *Design of model-reference adaptive control systems for aircraft*. Massachusetts Institute of Technology, Instrumentation Laboratory.

Wilson, A. and Ghahramani, Z. (2010). Copula processes. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2460–2468.

Wilson, A. and Ghahramani, Z. (2011). Generalised Wishart Processes. In *Proceedings of the Twenty-*

Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11), pages 736–744, Corvallis, Oregon. AUAI Press.

Ypma, A. and Heskes, T. (2005). Novel approximations for inference in nonlinear dynamical systems using expectation propagation. *Neurocomputing*, 69(1):85–99.

Yu, B. M., Shenoy, K. V., and Sahani, M. (2006). Expectation propagation for inference in nonlinear dynamical models with Poisson observations. In *NSSPW*, pages 83–86. IEEE.

Yuan, M. and Wahba, G. (2004). Doubly penalized likelihood estimator in heteroscedastic regression. *Statistics and Probability Letter*, 69:11–20.

Zhou, K., Doyle, J. C., Glover, K., et al. (1996). *Robust and optimal control*, volume 40. Prentice Hall New Jersey.