Gaussian Processes for Regression and Optimisation

Phillip Boyle

Submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2007

Abstract

Gaussian processes have proved to be useful and powerful constructs for the purposes of regression. The classical method proceeds by parameterising a covariance function, and then infers the parameters given the training data. In this thesis, the classical approach is augmented by interpreting Gaussian processes as the outputs of linear filters excited by white noise. This enables a straightforward definition of dependent Gaussian processes as the outputs of a multiple output linear filter excited by multiple noise sources. We show how dependent Gaussian processes defined in this way can also be used for the purposes of system identification.

One well known problem with Gaussian process regression is that the computational complexity scales poorly with the amount of training data. We review one approximate solution that alleviates this problem, namely reduced rank Gaussian processes. We then show how the reduced rank approximation can be applied to allow for the efficient computation of dependent Gaussian processes.

We then examine the application of Gaussian processes to the solution of other machine learning problems. To do so, we review methods for the parameterisation of full covariance matrices. Furthermore, we discuss how improvements can be made by marginalising over alternative models, and introduce methods to perform these computations efficiently. In particular, we introduce sequential annealed importance sampling as a method for calculating model evidence in an on-line fashion as new data arrives.

Gaussian process regression can also be applied to optimisation. An algorithm is described that uses model comparison between multiple models to find the optimum of a function while taking as few samples as possible. This algorithm shows impressive performance on the standard control problem of double pole balancing. Finally, we describe how Gaussian processes can be used to efficiently estimate gradients of noisy functions, and numerically estimate integrals.

Acknowledgments

Firstly, many thanks to my supervisor, Marcus Frean, who introduced me to Gaussian processes and lead me down the path of using them for optimisation. Marcus has supplied endless enthusiasm, invaluable leads, steered me from *ad hocery*, and humoured far too many wacky ideas. Furthermore, Marcus is the chief conspirator to the Festival of Doubt, a forum that germinated many of the ideas in this thesis.

Thank you to Sarah, my wife, who has endured this work, and provided support and sacrifice to keep me fed and on track. Without you I would not have started this work, nor would I have finished.

To my boys, Jack and Harry, you have no idea what I've been doing all this time, up in that office, tapping on a keyboard with strange scribblings littering the floor. Perhaps I can explain one day.

To Ainsley, Audrey, Gaelene, Roger and Jo - thanks for everything.

Thank you to Richard Mansfield for illuminating discussions over burgers, life-saving LaTeX advice, and for enhancing my world view.

Thank you to the participants in the Festival of Doubt, in particular Pondy who provided helpful criticism on many occasions, and Meng, who helped me stay on track.

Finally, thanks to my current employer and Glen Singleton for giving me the time and space required to *knock the bugger off*.

Contents

List of Figures

Chapter 1

Introduction

One broad definition of machine learning is the study of algorithms that improve automatically through experience [43]. Within this broad topic area, this thesis is mainly concerned with the method and application of *supervised learning*, a form of inductive learning that learns a functional mapping from training inputs to observed outputs. Given a set of training input vectors paired with observed outputs, a supervised learning machine attempts to build a function that summarises the input-output relationship. This summary can then be used for curve-fitting (e.g. interpolation), smoothing, or generalisation.

This thesis examines the use of Gaussian processes for supervised learning, specifically regression, and uses the results for the purposes of continuous optimisation and active learning.

1.1 Regression

A regression problem is a supervised learning problem in which we wish to learn a mapping from inputs to continuously valued outputs, given a training set of input-output pairs. We observe n training inputs $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_n]$ which reside in an input space $\mathcal{X}$, which may be continuous or discrete. The i^{th} training input $\mathbf{x}_i$ is associated with a training output, or target y_i, which in the simplest case is a real scalar value. The targets are assumed to have arisen from some (unknown) function of the inputs, but may also have been

corrupted by (unknown) noise. For notational convenience, we combine the targets into a vector $\mathbf{y} = [y_1 \dots y_n]^{\mathrm{T}}$.

Firstly, consider a form of regression known as *parametric* regression, where it is assumed that the training data has been generated by an underlying function $f(\mathbf{x}; \mathbf{w})$ defined in terms of some parameters $\mathbf{w}$. The functional mapping $f(\mathbf{x}; \cdot)$ along with a particular parameter set $\mathbf{w}$ defines a parametric model. Obviously, some parameter sets are better than others at explaining the observed outputs. Informally, parametric regression corresponds to finding the set of parameters that provide the "best" explanation of the data. We now have the problem of clarifying what we mean when we say that one model is the "best", or one model is "better" than another.

One way of finding the "best" model is to perform regression by finding the parameters that minimise some cost function $\mathcal{L}(\mathbf{w})$. We then say that models are better if they have lower costs. A common cost function is the sum of squared errors:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 \tag{1.1}$$

which favours models that fit the outputs more closely. We find the best model by minimising $\mathcal{L}(\mathbf{w})$ with respect to $\mathbf{w}$, and call the solution the *least squares* model. One famous example of this is back-propagation [62] where the parameters are the weights of a feedforward neural network, and the gradient of the cost is used to optimise the weights and fit the model. Another example is least squares polynomial regression, where the functional mapping is a polynomial and the parameters are the polynomial coefficients [17].

One problem with least squares regression is the lack of error bars on predictions. That is, the regression model supplies a scalar prediction at any point, without any measure of the confidence we should place in that prediction. More useful models would supply error bars with each prediction, or better, supply a full predictive distribution.

Another problem is that of *overfitting*. That is, least squares models (and in general, least cost models) are only concerned with reducing the model error at the training input points. What the model does at points between inputs is inconsequential to the modelling process. If we have a powerful enough model then we can come close to a zero-error model that interpolates the data almost exactly. Typically, however, we find such overly complex models

have poor *generalisation performance* - the models make poor predictions at test points not sufficiently similar to any training input.

Overfitting can be avoided by using a simpler model. A simple model tends to smooth out or ignore complicated features and noise. However, if the model is too simple, its predictive performance in the training data will be poor. Overall we have a trade-off situation where a model that is too simple interpolates poorly and has large predictive error at the training points. On the other hand, a model that is too complex fits the observations (and noise) well, but may make wildly inaccurate predictions at novel test points. The model we would like lies somewhere in between - a model that interpolates the observations sufficiently, and has a good generalisation performance.

An alternative to specifying a cost function is to assume a noise model on the outputs as described by the following generative model:

$$y_i = f(\mathbf{x}_i; \mathbf{w}) + \epsilon_i \tag{1.2}$$

where ϵ_i is independently and identically distributed (*i.i.d*) noise. In this thesis, it is generally assumed that this noise is Gaussian with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

Now we can make use of the *likelihood function*, or the probability density of the data given the parameters

$$p(\mathbf{y}|\,\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} p(y_i|\,\mathbf{x}_i, \mathbf{w}, \sigma^2) \tag{1.3}$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(\mathbf{x}_i; \mathbf{w}))^2}{2\sigma^2}\right) \tag{1.4}$$

where the likelihood function is factored because of the assumption that separate noise samples are independent [61]. The regression model is then built by finding the set of parameters $\mathbf{w}$ that maximise the likelihood function. The log of the likelihood function (1.3) is proportional to the negative of the sum of squared errors (1.1), so this *maximum likelihood* regression model is essentially the same as the least squares model.

1.2 Bayesian Regression

Bayesian parametric regression is an alternative regression method that counters the problems of overfitting. We make use of Bayes' rule to find the *posterior* distribution over the parameters, characterised by the probability density of the parameters conditioned on the observations:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \sigma^2) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2)\, p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X}, \sigma^2)} \tag{1.5}$$

where $p(\mathbf{w})$ is the *prior* probability density function (or prior density for short), and is set according to our prior belief about the distribution of the parameters. The numerator on the right consists of the likelihood function multiplied by the prior density. The denominator is the *marginal likelihood* and is found by integrating over the likelihood-prior product

$$p(\mathbf{y}|\mathbf{X}, \sigma^2) = \int p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2) p(\mathbf{w})\, d\mathbf{w} \tag{1.6}$$

To make a prediction y_* at a test point $\mathbf{x}_*$, we find the predictive distribution

$$p(y_*|\mathbf{x}_*, \mathbf{y}, \mathbf{X}, \sigma^2) = \int p(y_*|\mathbf{x}_*, \mathbf{w}, \sigma^2) p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \sigma^2) d\mathbf{w} \tag{1.7}$$

So rather than using a single set of parameters to make predictions, we integrate over the entire posterior density. This means that it is not just a single set of parameters that contributes to predictions, but *all* parameters contribute to a prediction, where the predictive contribution from a particular set of parameters is weighted by its posterior probability. The consequence of doing so is a predictive model powerful enough to model the problem's features, but less prone to overfitting.

Another nice feature of Bayesian prediction is that we have access to the full predictive distribution, rather than just a scalar prediction at each test point. This is very useful as a measure of the model's confidence in its prediction. If the predictive distribution is tightly packed around a single value, then we can be confident of the model's predictions, assuming that the parametric form of $f(\mathbf{x}; \mathbf{w})$ is appropriate for the data. On the other hand, if the predictive distribution is spread widely over a range of values, then the model is telling us that it has high uncertainty in what it expects to observe given this particular test input.

1.3 Gaussian Processes for Regression

1.3.1 Gaussian Processes

Consider a probability density function $p(f)$ defined over a function space F. If we were to sample functions f from F according to $p(f)$ then we would be generating sample paths from a *stochastic process*. The samples can be considered *sample paths* or *random functions* drawn from the distribution with density function $p(f)$.

We restrict our attention here to function spaces where each function in the space has a domain $\mathcal{X}$ and range $\mathbb{R}$. That is, for each $f \in F$ we have $f : \mathcal{X} \to \mathbb{R}$. If we generate samples from F, and for each sample f find the value at some fixed point $\mathbf{x} \in \mathcal{X}$, we will find that $f(\mathbf{x})$ is a random variable with some fixed distribution. As a simple example, consider the stochastic process defined by $f(x) = \exp(w)\sin(\alpha x)$ with $w \sim \mathcal{N}(0, \frac{1}{4})$ and $\alpha \sim \mathcal{N}(1, 1)$. We generate functions simply by sampling from $p(\alpha, w)$, with some examples shown in figure 1.1. The probability density of $f(1)$ is shown by the panel on the right of the figure. We can observe all the sample functions at n different fixed test points to generate a random vector, $\mathbf{f} = [f(\mathbf{x}_1) \ldots f(\mathbf{x}_n)]^{\mathrm{T}}$. The joint probability density $p(\mathbf{f})$ could then be found (at least empirically), which in this case has a non-trivial form.

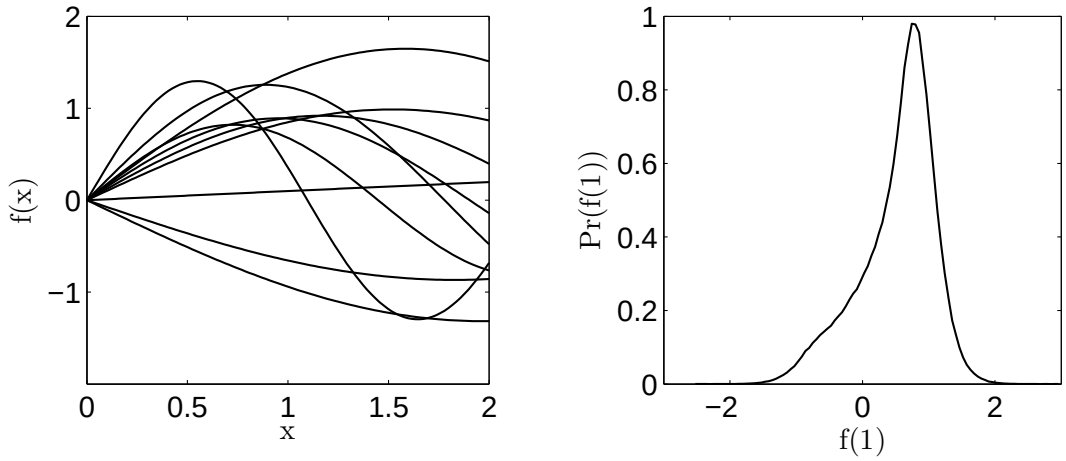

Figure 1.1: Example of a stochastic process. The panel on the left shows 10 independent sample paths. The panel on the right shows the probability density of the sample functions evaluated at $x = 1$. This was produced by normalising the histogram of 1000 sample paths evaluated at $f(1)$.

In this thesis, we consider a subset of all stochastic processes that have the property that the joint distribution over any finite set of fixed test points is a multivariate Gaussian. That is, the distribution of $\mathbf{f} \in \mathbb{R}^n$ is a multivariate Gaussian for all finite n and all $\mathbf{x}_i \in \mathcal{X}$. Such a stochastic process is known as a *Gaussian process*.

An example of a Gaussian process is shown in figure 1.2. The left panel shows 5 independent sample paths drawn from the Gaussian process. The samples are independent but all have similar characteristics such as expected rate of change and expected magnitude. The right panel shows the (theoretical) joint probability density of the sample paths evaluated at two fixed points $f(0.3)$ and $f(0.5)$. This is a bivariate Gaussian, consistent with the definition of a Gaussian process.

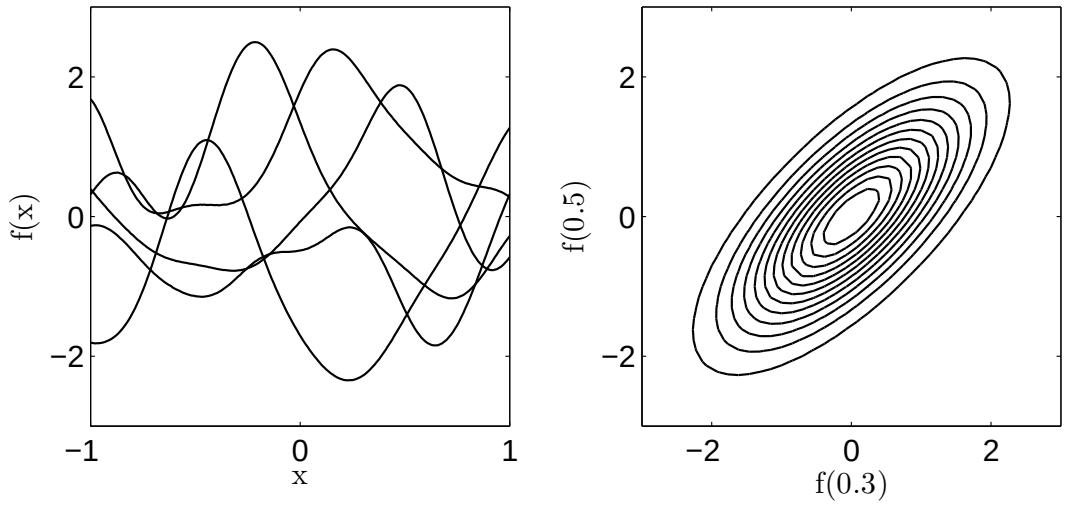

Figure 1.2: Example of a Gaussian process. The left panel shows 5 independent sample paths from the Gaussian process. The panel on the right shows the contours of the joint probability density function for the bivariate Gaussian distribution for the variables $f(0.3)$ and $f(0.5)$.

1.3.2 Gaussian Process Models

Earlier, we saw how we could assume a particular parametric generative model, and then use Bayes' rule to infer the parameters. In this section we consider an alternative, where we assume that each observation y_i is depen-

dent on a latent variable f_i as follows

$$y_i = f_i + \epsilon_i \tag{1.8}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is *i.i.d* noise.

We are thus considering n latent variables which we collect into a vector $\mathbf{f} = [f_1 \dots f_n]^{\mathrm{T}}$. In the Gaussian process for regression methodology, we place a zero-mean multivariate Gaussian prior distribution over $\mathbf{f}$. That is

$$\mathbf{f} \big| \mathbf{X}, \boldsymbol{\theta} \;\; \sim \;\; \mathcal{N}(\mathbf{0}, \mathbf{K}) \tag{1.9}$$

where $\mathbf{K}$ is an $n \times n$ covariance matrix dependent on $\mathbf{X}$ and some hyperparameters $\boldsymbol{\theta}$. In particular, the $(i,j)^{th}$ element of $\mathbf{K}$ is equal to $k(\mathbf{x}_i, \mathbf{x}_j)$ where $k(\cdot, \cdot)$ is a positive definite function parameterised by $\boldsymbol{\theta}$. In this context $k(\cdot, \cdot)$ is known as a *covariance function*[1].

Given some observations and a covariance function, we wish to make a prediction using the Gaussian process model. To do so we consider a test point $\mathbf{x}_*$ and the associated latent variable f_*. Under the Gaussian process framework, the joint distribution of $\mathbf{f}$ and f_* is also a zero-mean multivariate Gaussian, and is found by augmenting (1.9) with the new latent variable f_*:

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \Bigg| \, \mathbf{X}, \boldsymbol{\theta} \;\; \sim \;\; \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & \kappa \end{bmatrix} \right) \tag{1.10}$$

where $\mathbf{k} = [k(\mathbf{x}_*, \mathbf{x}_1) \dots k(\mathbf{x}_*, \mathbf{x}_n)]^{\mathrm{T}}$ is the $n \times 1$ vector formed from the covariance between $\mathbf{x}_*$ and the training inputs. The scalar $\kappa = k(\mathbf{x}_*, \mathbf{x}_*)$.

Given the Gaussian noise assumption in equation (1.8), we can express the joint distribution over the observed targets $\mathbf{y}$ and unobserved (test) target y_*:

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \Bigg| \, \mathbf{X}, \boldsymbol{\theta} \;\; \sim \;\; \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma^2 \mathbf{I} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & \kappa + \sigma^2 \end{bmatrix} \right) \tag{1.11}$$

Given that the joint distribution is Gaussian, we can condition on $\mathbf{y}$ using standard formulae [61] to find

$$y_* \big| \mathbf{y}, \mathbf{X}, \boldsymbol{\theta}, \sigma^2 \;\; \sim \;\; \mathcal{N}\big(m(\mathbf{x}_*), v(\mathbf{x}_*)\big) \tag{1.12}$$

[1] Positive definite covariances imply a positive definite covariance matrix $\mathbf{K}$ which is required to make equation (1.9) normalisable

where the predictive mean and variance are

$$m(\mathbf{x}_*) = \mathbf{k}^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{1.13}$$

$$v(\mathbf{x}_*) = \kappa + \sigma^2 - \mathbf{k}^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k} \tag{1.14}$$

Hence, given a covariance function defined by hyperparameters $\boldsymbol{\theta}$, we can calculate a Gaussian predictive distribution for *any* test point $\mathbf{x}_*$. More generally, we can calculate the multivariate Gaussian predictive distribution for any *set* of m test points $\mathbf{X}_* = [\mathbf{x}_{1*} \dots \mathbf{x}_{m*}]$ as follows:

$$m(\mathbf{X}_*) = \mathbf{K}_*^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{1.15}$$

$$v(\mathbf{X}_*) = \mathbf{K}_{**} + \sigma^2\mathbf{I} - \mathbf{K}_*^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_* \tag{1.16}$$

where $\mathbf{K}_*$ is an $n \times m$ matrix of covariances between the training inputs and test points. The $m \times m$ matrix $\mathbf{K}_{**}$ consists of the covariances between the test points.

Gaussian process regression is like Bayesian parametric regression where the latent variables replace the parameters. Implicitly, we find a posterior density over the latent variables and then integrate over that posterior density to make predictions. We can perform the integral analytically because the distributions making up the integrand are Gaussian.

For a Gaussian process model the marginal likelihood is equal to the integral over the product of the likelihood function and the prior density[1], which are both Gaussian in form. The product of two Gaussians is another Gaussian, so the marginal likelihood is available in analytical form:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, \sigma^2) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \boldsymbol{\theta}, \sigma^2)p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta})d\mathbf{f} \tag{1.17}$$

$$= \int \mathcal{N}(\mathbf{f}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{0}, \mathbf{K})d\mathbf{f} \tag{1.18}$$

$$= \frac{1}{(2\pi)^{\frac{n}{2}}|\mathbf{K} + \sigma^2\mathbf{I}|^{\frac{1}{2}}} \exp\left(-\tfrac{1}{2}\mathbf{y}^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}\right) \tag{1.19}$$

For numerical reasons we usually work with the *log marginal likelihood*

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, \sigma^2) = -\tfrac{n}{2}\log 2\pi - \tfrac{1}{2}\log|\mathbf{K} + \sigma^2\mathbf{I}| - \tfrac{1}{2}\mathbf{y}^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{1.20}$$

which can be considered as the log-evidence for *this* particular Gaussian process model, defined by $\boldsymbol{\theta}$ and σ^2.

[1] at this level of inference, the likelihood function is the likelihood of the latent variables $\mathbf{f}$ and the prior density is over $\mathbf{f}$. The marginal likelihood comes about by *marginalising* over $\mathbf{f}$.

1.3.3 Learning the Hyperparameters

Usually, when we use Gaussian processes for regression, we do not know *a priori* the most appropriate hyperparameters and noise variance. For the methodology to be useful, we need a principled method for inferring these unknowns from the data. In other words, we desire a method to learn the hyperparameters and noise variance given the observations.

The marginal likelihood (equation (1.17)) can be thought of as the likelihood of the hyperparameters and noise variance. One way to build a Gaussian process model is to find the values of $\boldsymbol{\theta}$ and σ^2 that maximise this likelihood. In doing so, we find the maximum likelihood hyperparameters $\boldsymbol{\theta}_{ML}$ and maximum likelihood noise variance σ^2_{ML}. Once found, we make predictions by feeding $\boldsymbol{\theta}_{ML}$ and σ^2_{ML} into equation (1.12).

In many cases we may have a prior belief about the form of the data. If we can translate this into a prior belief about the hyperparameters and noise variance, then it is better to incorporate this information into the learning of these values. To do so, we find the posterior density over the hyperparameters and noise variance as follows:

$$p(\boldsymbol{\theta}, \sigma^2|\mathbf{y}, \mathbf{X}) \quad \propto \quad p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, \sigma^2)\, p(\boldsymbol{\theta}, \sigma^2) \tag{1.21}$$

which is just the likelihood function times the prior density just discussed. Now, instead of maximising the likelihood function, we find the hyperparameters and noise variance to maximise the posterior density. This gives us the so called *maximum a posteriori*, or MAP values $\boldsymbol{\theta}_{MAP}$ and σ^2_{MAP}, which we feed into (1.12) to make predictions. Using a prior distribution to find the MAP values can often produce better results than simply using the maximum likelihood values [17].

Both of the above methods make predictions using a single set of hyperparameters and a single value for the noise variance. Although these methods work well in practise, they are in fact only approximations to the Bayesian solution, which makes predictions by marginalising over the uncertainty in the hyperparameters and noise variance as follows [86, 34]:

$$p(y_*|\mathbf{x}_*, \mathbf{y}, \mathbf{X}) = \iint p(y_*|\mathbf{x}_*, \mathbf{y}, \mathbf{X}, \boldsymbol{\theta}, \sigma^2) p(\boldsymbol{\theta}, \sigma^2|\mathbf{y}, \mathbf{X}) d\boldsymbol{\theta} d\sigma^2 \tag{1.22}$$

Normally, however, this integral is not analytically tractable and we are forced to make some sort of approximation. When we use the MAP method, we

are effectively approximating the posterior density $p(\boldsymbol{\theta}, \sigma^2|\mathbf{y}, \mathbf{X})$ with a delta function centred on $(\boldsymbol{\theta}_{MAP}, \sigma^2_{MAP})$ so the predictive distribution simplifies to $p(y_*|\mathbf{y}, \mathbf{X}, \boldsymbol{\theta}_{MAP}, \sigma^2_{MAP})$. Other methods attempt to use more of the information offered by the posterior distribution. For example, one can approximate the posterior distribution with a Gaussian centred on the MAP solution, and then perform the (approximated) integral analytically (similar to the Bayesian treatment of Neural Networks by MacKay [38, 36]). However, this may produce poor results if the posterior distribution is multimodal, or if the model has a large number of parameters (meaning the posterior distribution is defined over a space with a large number of dimensions). Another solution is to use *Markov Chain Monte Carlo* (MCMC) methods to simulate the posterior distribution by numerically generating a set of samples [86, 57, 47]. The predictive distribution is then approximated as follows:

$$p(y_*|\mathbf{x}_*, \mathbf{y}, \mathbf{X}) \approx \frac{1}{M} \sum_{m=1}^{M} p(y_*|\mathbf{y}, \mathbf{X}, \boldsymbol{\theta}_m, \sigma^2_m) \tag{1.23}$$

where we have M samples with $(\boldsymbol{\theta}_m, \sigma^2_m) \sim p(\boldsymbol{\theta}, \sigma^2|\mathbf{y}, \mathbf{X})$. The advantage is that this prediction becomes increasingly correct as the number of samples increases. Unfortunately, drawing samples from the posterior distribution by simulating Markov chains is not always straightforward. For instance, it can be difficult to determine beforehand how long the simulation must run to ensure that the generated samples are representative of the posterior distribution. Typically, the solution is to run long simulations with the consequence that MCMC methods can be computationally demanding. For good coverage of MCMC and these issues, refer to the work of Neal [45] and MacKay [41, 34].

1.4 History of Gaussian Processes

The study of Gaussian processes and their use for prediction is far from new [34]. Indeed, the underlying theory dates back to Weiner-Kolmogorov prediction theory and time series analysis in the 1940's [34, 61, 33, 40]. More recent is the introduction of *kriging* [42], and its subsequent development as a method for the interpolation of geostatistical data [12]. Kriging, named after the mining engineer D.G.Krige, is identical to Gaussian process regression,

but is derived and interpreted somewhat differently to that above (e.g. see [27]). Furthermore, as a geostatistical method, it is mainly concerned with low-dimensional problems and tends to ignore any probabilistic interpretations [34]. In the wider statistical community, the use of Gaussian processes to define prior distributions over functions dates back to 1978, where O'Hagan [50] applied the theory to one-dimensional curve fitting.

In the machine learning community, the use of Gaussian processes for supervised learning is a more recent development which traces back to introduction of back-propagation for learning in neural networks [62]. This original non-probabilistic treatment was subsequently enhanced by Buntine [9], MacKay [35], and Neal [48] who introduced a Bayesian interpretation that provided a consistent method for handling network complexity (see [38, 4, 28] for reviews). Soon after, Neal [46] showed that under certain conditions these *Bayesian Neural Networks* converge to Gaussian processes in the limit of an infinite number of units. This resulted in the introduction of Gaussian processes for regression in a machine learning context [86, 57, 47]. Briefly, this work included a description of how to

(1) specify and parameterise a covariance function.

(2) build a covariance matrix and hence express the prior distribution over function values.

(3) find the posterior distribution over parameters using Bayes' Theorem.

(4) either optimise to find the most likely (ML) or maximum *a posteriori* (MAP) parameters, or integrate over the posterior density using Hamiltonian Monte Carlo.

(5) calculate the predictive distribution at any test point.

For good introductions to Gaussian processes for regression refer to the 1997 thesis of Gibbs [17], the Gaussian processes chapter in MacKay's book [34], and the recent book by Williams and Rasmussen [61]. Additionally, Seeger provides recent reviews [63, 64, 65] and relates Gaussian processes for machine learning to other kernel machine methods.

Since the original introduction of Gaussian processes for regression, there have been numerous enhancements and applications. One of the main areas

of interest has been on developing methods to reduce the computational cost of Gaussian process regression, both in the training and prediction phases. The fundamental problem is that for a training set of size n, exact calculation of the marginal-likelihood (1.17) has complexity $\mathcal{O}(n^3)$. This cost is a direct result of inverting an $n \times n$ matrix, so some of the methods aim to approximate this calculation. For example, [17, 18] describe and analyse an iterative method to approximate the inverse with complexity $\mathcal{O}(n^2)$. Another interesting approach is presented by Williams et al. [88, 87], who make use of the Nyström method to form a rank $m < n$ matrix approximation to the covariance matrix, which can then be inverted with a cost $\mathcal{O}(m^2 n)$. There have been many more recent developments (e.g. [71, 80, 13, 14, 15, 66]), including the reduced rank approximation of Quiñonero-Candela and Rasmussen [55] to be considered in chapter 5. For a good review and summary of these methods see [56, 61].

Other recent work has been extensive and varied. For example, Gibbs [17] and Paciorek [52, 53] developed methods for creating non-stationary covariance functions, and hence, models of non-stationary data. We have seen methods to deal with input-dependent noise [20] and non-Gaussian noise [72]. Mixtures of Gaussian processes were introduced by [81] followed by an extension to a tractable infinite mixture of Gaussian processes experts [59]. Interesting machine learning applications include Gaussian processes for reinforcement learning [60], the incorporation of derivative observations into Gaussian process models [73], Gaussian processes to speed up the evaluation of Bayesian integrals [58], and Gaussian process models of dynamical systems [83].

Gaussian processes have also proved useful for classification problems. However, in this case the likelihood function and evidence and hence the posterior distribution are not Gaussian, so exact inference is not possible. As a result, much work has gone into developing approximations. Many of the resultant classifiers make use of the Laplace approximation [3], Markov Chain Monte Carlo [47], and variational methods [17, 19]. Although Gaussian process classifiers are powerful and promising, this thesis is concerned only with Gaussian processes for regression.

1.5 Overview of the Thesis

Chapter 2 reviews the theory of *Linear Time Invariant filters* and describes how they can be used to generate Gaussian processes. In particular, it is established that one can specify and statistically characterise a Gaussian process by defining a linear filter, instead of using the classical method where a Gaussian process is characterised directly via a covariance function. Overall, this chapter presents a new way to construct a parameterised Gaussian processes. The advantage of doing so is that it is usually easier to define a stable, linear filter than it is to define a valid, positive definite covariance function.

Chapter 3 extends the framework developed in chapter 2, by introducing the notion of multiple output linear filters. Doing so naturally defines a set of Gaussian processes that are dependent on one another, which are named in this thesis as *Dependent Gaussian Processes*. Dependent Gaussian processes can be used to produce *multiple output* models, something that until now has been problematic. That is, a current open problem concerns the difficulty of directly specify valid covariance and cross-covariance functions that result in a set of dependent Gaussian processes. The problem is alleviated if instead we construct the set of dependent Gaussian processes using multiple output filters.

Chapter 4 shows how one can apply the dependent Gaussian processes framework to the problem of *system identification*. That is, it is shown how to treat a system identification problem as a dependent Gaussian processes modelling problem. Although the system identification problem is far from new, this chapter shows how one can approach and solve the problem using the contributions from the previous chapter.

One of the problems with Gaussian processes for regression is the computational complexity of implementation. The cost of training and prediction scales poorly with the amount of data - specifically, the complexity is $\mathcal{O}(n^3)$ for n training examples. There are a number of approximations that have been developed to overcome this complexity. Chapter 5 reviews one such method, *Reduced Rank Gaussian Processes*. A new method is then presented showing how the reduced rank Gaussian processes methodology can be used to construct non-stationary Gaussian processes.

Chapter 6 extends the contents of chapters 3 and 5 and introduces a new ap-

proximation to implement dependent Gaussian processes in a computationally efficient manner. These models are named *Reduced Rank Dependent Gaussian Processes*. The framework is then extended by introducing *non-stationary* reduced rank Gaussian processes.

Chapter 7 reviews some methods to increase the modelling power of Gaussian processes that use squared-exponential covariance functions. In particular, this includes an examination of parameterisations capable of encoding full covariance matrices, thereby enabling the specification of covariance functions that do not necessarily align with the input space axes. In other words, this chapter discuss methods to allow covariance functions to rotate in high dimensional space.

Chapter 8 is concerned with improving Gaussian processes for regression by using Bayes' Theorem to marginalise over different covariance functions. In its simplest form, this amounts to weighting alternate models based on the *evidence* for each model as determined from Bayes' Theorem. In this way, models supported by strong evidence will have more influence on the regression model's predictions. The method of *annealed importance sampling* is reviewed, and a new heuristic is described that automatically constructs an annealing schedule. Finally, this chapter presents a novel method, termed *sequential annealed importance sampling*, which can be used calculate the evidence for a Gaussian process model.

A current open research problem is that of *continuous optimisation* in a sample efficient manner - that is, optimising a set of continuous decision variables with a minimal number of objective function evaluations. Chapter 9, introduces an approach to this problem that uses *response surfaces* and the *expected improvement* to guide search. Following is a review of how Gaussian processes can be used to build response surfaces and hence solve continuous optimisation problems - a method known as *Gaussian Process Optimisation*. This chapter identifies some problems with this method. In particular, it is shown that this method can perform poorly if the main features of the objective function are rotated relative to the axes of the covariance function.

Chapter 10 presents novel expansions of the Gaussian Process Optimisation algorithm presented in the previous chapter. This includes new enhancements that use Gaussian process models with rotated covariance functions, and the use multiple models with model comparison to improve performance.

Furthermore, it is described how reduced rank Gaussian processes as presented in chapter 5 can be used to improve optimisation performance on problems requiring many iterations. Finally, it is shown how the Gaussian processes for optimisation algorithm can be used to solve the double pole balancing problem in an efficient manner.

Chapter 11 continues with the application of Gaussian processes to other machine learning problems. In particular, this chapter presents new algorithms that use Gaussian processes to address the currently open problem of efficiently estimating the gradient and definite integral of a noisy function.

Chapter 2

Gaussian Processes from Linear Filters

The previous chapter described how Gaussian processes could be used for regression. To do so required the specification of a covariance function, which must be a positive definite function. Positive definiteness forces the covariance matrix $\mathbf{K}$ in equation (1.9) to have positive eigenvalues, thus ensuring that the prior density over function values $p(\mathbf{f})$ is normalisable[1].

This chapter examines how Gaussian processes can be constructed by stimulating linear filters with noise. We find the Gaussian process so constructed is completely characterised by the properties of the filter. Furthermore, for regression, instead of learning a positive definite covariance function, we can learn a filter.

2.1 Linear Time Invariant Filters

Consider a device that operates on a continuous, real valued input signal over time $x(t)$ and emits a continuous real valued output $y(t)$. This device is a

[1]The $\log$ of a Gaussian is a negative quadratic. If this quadratic has positive eigenvalues then it will tend to $-\infty$ at extrema, and the Gaussian will tend to zero at extrema. However, if *any* eigenvalues are negative, then the negative quadratic will tend to ∞ in at least one dimension, and hence cannot be exponentiated to form a valid, normalised probability density function.

linear time invariant (LTI) filter [23, 2] if it has the following properties:

(1) *Linearity*. The output is linearly related to the input in that if we multiply the input by a constant, then the output is multiplied by the same amount. Furthermore, if the input consists of the superposition of two signals $x(t) = x_1(t) + x_2(t)$, then the output $y(t) = y_1(t) + y_2(t)$, is the sum of the two output signals that result from independent application of the two input signals.

(2) *Time Invariance*. Shifting the input signal in time results in exactly the same shift in time for the output. So, if the output is $y(t)$ in response to an input $x(t)$, then the output in response to a shifted input $x(t + \tau)$ is $y(t + \tau)$.

An LTI filter is *completely* characterised by its impulse response, $h(t)$, which is equivalent to the output when the filter is stimulated by a unit impulse $\delta(t)$. Given the impulse response, we can find the output of the filter in response to any finite input via convolution:

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(t-\tau)x(\tau)d\tau = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau \tag{2.1}$$

with the input-output relationship shown diagrammatically in figure 2.1.

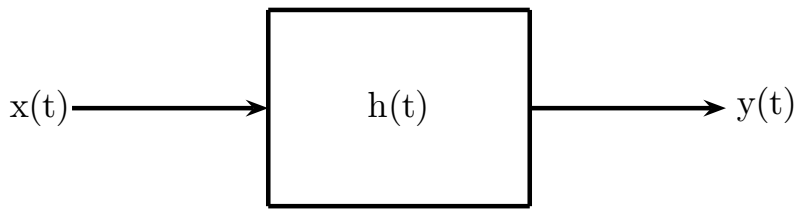

Figure 2.1: Single-input single-output linear time invariant filter, characterised by an impulse response $h(t)$. The output $y(t)$ is related to the input $x(t)$ by equation (2.1).

So, if we apply a unit impulse to an LTI filter with impulse response $h(t)$, the output we observe $y(t) = \int_{-\infty}^{\infty} h(\tau)\delta(t-\tau)d\tau = h(t)$, is the *impulse response*, as defined.

As an example, consider an idealised tuning fork that emits a decaying, pure tone when struck. We idealise the input strike by defining it as a unit impulse, and define the impulse response as $h(t) = \exp(-\alpha t)\sin(\omega t)$. When this tuning

fork is struck, the output is a pure, decaying sinusoid equal to the impulse response. By the linearity property, if we strike the fork twice separated by time τ, the response will be equal to the sum of two decaying sinusoids, separated by time τ.

A filter is said to be *bounded input bounded output* (BIBO) stable if the output is bounded for all inputs that are bounded [23]. The input is considered bounded if there exists a positive, real finite number M such that $|x(t)| \leq M$ for all t. Similarly, the output is bounded if there exists a positive, real finite number N such that $|y(t)| \leq N$ for all t. A necessary and sufficient condition for BIBO stability is that the impulse response is absolutely integrable: $\int_{-\infty}^{\infty} |h(t)|dt < \infty$. In this thesis, we will only consider BIBO stable filters, hence will only consider impulse responses that are absolutely integrable.

Stable linear filters have the property that if a Gaussian process is applied as input, then the output is necessarily a Gaussian process [23]. *Gaussian white noise*, is a particular Gaussian process in which the covariance between two points t_i and t_j is $\delta_{ij}\sigma^2$, where δ_{ij} is the Kronecker delta function, and σ^2 is the noise variance. So, if we input Gaussian white noise into an LTI filter, we will generate a Gaussian process at the output. The covariance function of this process is completely characterised by the input noise variance, and the impulse response. Normally, a Gaussian process model is built by parameterising the covariance function, but by viewing Gaussian processes as the outputs of LTI filters we have an alternative method. That is, we can specify a Gaussian process model by parameterising the impulse response.

When a linear filter is excited with Gaussian white noise $w(t)$, the covariance function of the zero-mean output process is found as follows:

$$\begin{aligned}
\operatorname{cov}(y(t), y(t')) &= E\{y(t)y(t')\} && (2.2)\\
&= E\left\{\int_{-\infty}^{\infty} h(\tau)w(t-\tau)\,d\tau \int_{-\infty}^{\infty} h(\lambda)w(t'-\lambda)\,d\lambda\right\} && (2.3)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\tau)h(\lambda)E\{w(t-\tau)w(t'-\lambda)\}\,d\tau d\lambda && (2.4)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\tau)h(\lambda)\delta(\lambda-(t'-t+\tau))\,d\tau d\lambda && (2.5)\\
&= \int_{-\infty}^{\infty} h(\tau)h(t'-t+\tau)\,d\tau && (2.6)
\end{aligned}$$

where we can interchange the order of the expectation and integration in (2.4) because the impulse response is absolutely integrable. The covariance function is thus found to be equivalent to the autocorrelation function of the impulse response.

So now, instead of directly parameterising a positive definite covariance function, we parameterise the impulse response for a stable LTI filter. In doing so, the covariance function is automatically implied by (2.6). The only restriction is that the filter is stable, which we must enforce via a suitable parameterisation.

2.1.1 Filters in Higher Dimensions

We have seen how we can construct Gaussian processes over time by stimulating linear filters with a Gaussian white noise process. In this section, we generalise to Gaussian processes over continuous D dimensional input spaces $\mathcal{X} = \mathbb{R}^D$. To do so, we need to consider multidimensional linear filters.

Just as a $1D$ filter is completely characterised by its impulse response, a multidimensional filter over $\mathbb{R}^D$ is completely characterised by it's D dimensional impulse response $h(\mathbf{x}), \mathbf{x} \in \mathbb{R}^D$. Furthermore, the output $y(\mathbf{x})$ in response to an input $u(\mathbf{x})$ is found by the multidimensional convolution

$$y(\mathbf{x}) = \int_{\mathbb{R}^D} u(\mathbf{s}) h(\mathbf{x} - \mathbf{s}) \, d^D\mathbf{s} \tag{2.7}$$

From appendix A.1, the covariance function between two outputs $y(\mathbf{x}_i)$ and $y(\mathbf{x}_j)$ is

$$\operatorname{cov}(y(\mathbf{x}_i), y(\mathbf{x}_j)) = \int_{\mathbb{R}^D} h(\mathbf{s}) h(\mathbf{s} + \mathbf{x}_i - \mathbf{x}_j) \, d^D\mathbf{s} \tag{2.8}$$

which in the case of *time invariant* filters simplifies to the stationary form

$$c(\boldsymbol{\tau}) = \operatorname{cov}(y(\mathbf{x}_i), y(\mathbf{x}_j)) \tag{2.9}$$

$$= \int_{\mathbb{R}^D} h(\mathbf{s}) h(\mathbf{s} + \boldsymbol{\tau}) \, d^D\mathbf{s} \tag{2.10}$$

where $\boldsymbol{\tau}$ is the difference between the two input points.

In order to find the covariance function in closed form, we must be able to perform the correlation integral (2.8). In general, this is intractable, but in

certain cases we can find analytic solutions. In the following sections we will examine two such cases.

2.1.2 Gaussian Filters

A Gaussian filter over $\mathbb{R}^D$ is an LTI filter with a Gaussian, or squared-exponential impulse response

$$h(\mathbf{x}) = v \exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\mathbf{A}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{2.11}$$

parameterised by a scale $v \in \mathbb{R}$, an offset $\boldsymbol{\mu} \in \mathbb{R}^D$ and a positive definite matrix $\mathbf{A}$ controlling the shape of the Gaussian.

The covariance function of the Gaussian process generated by exciting this filter with Gaussian white noise is found as in appendix A.2:

$$c(\boldsymbol{\tau}) = \frac{v^2(2\pi)^{\frac{D}{2}}}{\sqrt{|2\mathbf{A}^{-1}|}} \exp\left(-\frac{1}{2}\boldsymbol{\tau}^{\mathrm{T}}\left(\frac{\mathbf{A}^{-1}}{2}\right)\boldsymbol{\tau}\right) \tag{2.12}$$

Therefore, the covariance function for the output of a Gaussian filter excited by white noise is Gaussian. Furthermore, the covariance function is independent of the offset $\boldsymbol{\mu}$, meaning that translating the impulse response has no effect on the statistics of the output Gaussian process. The offset $\boldsymbol{\mu}$ can therefore be set to zero in this case, simplifying the parameter set to just v and $\mathbf{A}$. In chapter 3, we see that $\boldsymbol{\mu}$ becomes significant when constructing sets of dependent Gaussian processes.

2.1.3 Ideal Low-Pass Filters

An *ideal* low-pass filter has an impulse response defined in terms of the sine cardinal, or sinc function. In time, the sinc function is:

$$\mathrm{sinc}(t) = \begin{cases} 1 & t = 0 \\ \frac{\sin(t)}{t} & \text{otherwise} \end{cases} \tag{2.13}$$

The impulse response for an ideal low-pass filter with cutoff frequency of f_c and unit gain is [23]:

$$h(t) = 2f_c\,\mathrm{sinc}(2f_c\pi t) \tag{2.14}$$

Note that the frequency response for this filter is given by the Fourier transform of the impulse response and is equal to:

$$H(f) = \text{rect}\left(\frac{f}{2f_c}\right) = \begin{cases} 1 & f \leq f_c \\ 0 & f > f_c \end{cases} \tag{2.15}$$

This gives unit gain for frequencies less than or equal to the cutoff, but completely attenuates higher frequencies. This filter is ideal in that it has a perfectly rectangular frequency response.

Over $\mathbb{R}^D$, the unit gain impulse response is defined by a product of sinc functions:

$$h(\mathbf{s}) = 2f_c \prod_{d=1}^{D} \text{sinc}(2f_c \pi s_d) \tag{2.16}$$

where spatial frequencies in all directions $s_1 \dots s_D$ are completely attenuated above f_c, meaning this filter is isotropic.

We can generate an isotropic, stationary Gaussian process by applying Gaussian white noise to an ideal isotropic low-pass filter with impulse response $h(\mathbf{s}) = \prod_{d=1}^{D} \text{sinc}(\pi s_d)$. The covariance function of this process is:

$$c(\boldsymbol{\tau}) = \int_{\mathbb{R}^D} h(\mathbf{s}) h(\mathbf{s} + \boldsymbol{\tau}) d^D\mathbf{s} \tag{2.17}$$

$$= \prod_{d=1}^{D} \int_{-\infty}^{\infty} \text{sinc}(\pi s_d) \,\text{sinc}(\pi(s_d + \tau_d)) ds_d \tag{2.18}$$

$$= \prod_{d=1}^{D} \text{sinc}(\pi \tau_i) \tag{2.19}$$

where $\boldsymbol{\tau} = [\tau_1 \dots \tau_D]^{\mathrm{T}}$ is the difference between the two input points. In deriving 2.19 we have made use of the fact that $\int \text{sinc}(\pi y) \,\text{sinc}(\pi(x - y)) dy = \text{sinc}(\pi x)$.

So the covariance function of ideal low-pass filtered Gaussian white noise is a sinc function. From figure 2.2, we see that the sinc function is not everywhere positive, so it differs from the stationary kernels that are most often used in Gaussian process regression, such as the squared exponential, Matérn, γ-exponential and rational quadratic, which are always positive. Furthermore, the sinc function exhibits *ringing*, or an oscillation about zero that decays with input distance. Ringing effects are common in situations where ideal low-pass

filters are employed. It seems reasonable to build models based on ideal low-pass filtered noise, but a ringing covariance function is not so palatable. There do not seem to be many situations in which we expect correlations across space to fall away, and then become negative correlations, and then again become positive correlations in a decaying periodic fashion.

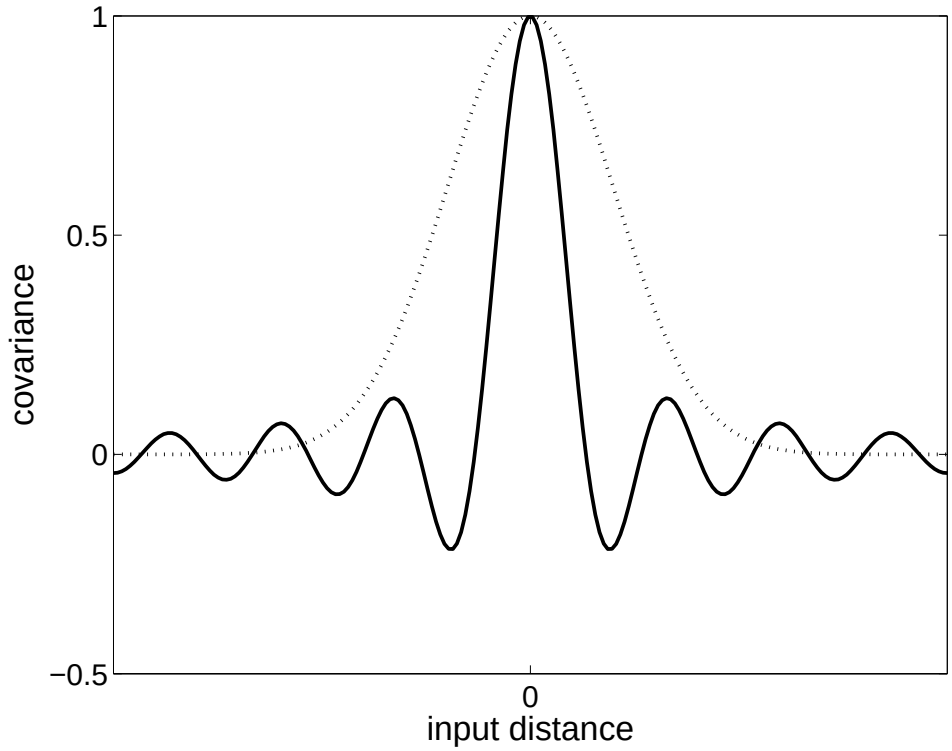

Figure 2.2: Sinc covariance function (solid) compared with a squared exponential covariance function (dotted).

2.2 Digital Filters

The previous section introduced linear filters defined over continuous time. Such filters are known as *analog* filters. In this section, we consider devices defined over discrete time, known as *digital* filters [25, 2]. A linear digital filter takes an input time sequence $x(n)$, and emits an output sequence $y(n)$. The input-output relationship is completely determined by the filter's impulse response sequence $h(n)$.

This section considers two types of digital filters: the finite impulse response (FIR) filter, and the infinite impulse response (IIR) filter. The following sections show how discrete Gaussian processes are constructed from each. Before we begin, we define the discrete time unit impulse

$$\delta(n) = \begin{cases} 1 & n = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.20}$$

2.2.1 FIR Filters

A FIR filter exhibits an impulse response that is non-zero for a finite number of time steps. Over discrete time, indexed by n, the impulse response, $h(n)$, of the filter is the output in response to $\delta(n)$

$$h(n) = \sum_{i=0}^{M-1} b_i \delta(n-i) \tag{2.21}$$

which is fully characterised by the M filter coefficients, collected into a vector $\mathbf{b} = [b_0 \dots b_{M-1}]^{\mathrm{T}}$.

The output of an analog filter is a continuous convolution of the input with the impulse response. Similarly, the output $y(n)$ of a digital filter is a *discrete convolution* of the input $x(n)$ and the impulse response $h(n)$

$$\begin{aligned} y(n) = x(n) * h(n) &= \sum_{k=-\infty}^{\infty} x(n-k)h(k) \\ &= \sum_{k=-\infty}^{\infty} \left(x(n-k) \sum_{i=0}^{M-1} b_i \delta(k-i) \right) \\ &= \sum_{k=-\infty}^{\infty} x(n-k) b_k \\ &= \sum_{k=0}^{M-1} x(n-k) b_k \end{aligned} \tag{2.22}$$

where the final step makes use of the fact that there are only M potentially non-zero coefficients, $b_0 \dots b_{M-1}$.

The FIR filter is shown diagrammatically in figure 2.3.

If the filter weights are finite, then the FIR filter is BIBO stable. Consequently, *any* vector of weights $\mathbf{b} \in \mathbb{R}^M$ defines a stable FIR filter.

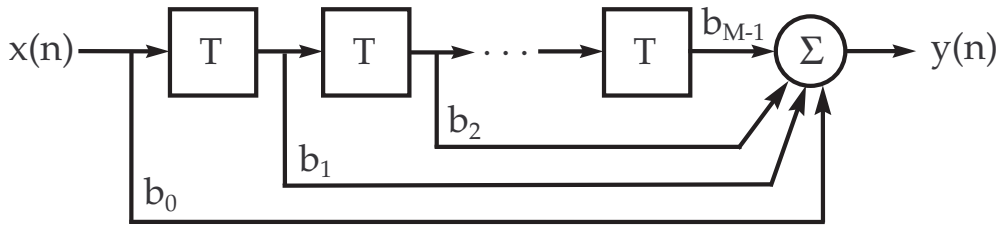

Figure 2.3: FIR filter with M coefficients. The input feeds into a buffer of length M. The output is a weighted sum of all of the levels in the buffer, with the i^{th} level weighted by coefficient b_i. Each level of the buffer results in a delay of time T, corresponding to the sampling period.

If an FIR filter is stimulated with discrete-time Gaussian white noise $x(n) \sim \mathcal{N}(0,1)$, then the output at any time step is a weighted sum of Gaussian random variables, and is therefore Gaussian itself. Overall, the output forms a Gaussian process. The covariance function of the output, between times m and n is

$$\begin{aligned}\operatorname{cov}(y(m), y(n)) &= E\left\{\sum_{i=0}^{M-1} b_i x(m-i) \sum_{j=0}^{M-1} b_j x(n-j)\right\} \\ &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} b_i b_j E\left\{x(m-i)x(n-j)\right\} \\ &= \sum_{j=0}^{M-1} b_j b_{j+m-n} \end{aligned} \tag{2.23}$$

Note that for an FIR filter,

$$b_j = \begin{cases} 0 & j < 0 \\ 0 & j \geq M \\ b_j & \text{otherwise} \end{cases} \tag{2.24}$$

so its covariance function has compact support, with $\operatorname{cov}(y(m), y(n)) = 0$ if $|m-n| \geq M$.

Note that the covariance function (2.23) is equivalent to a discrete correlation. That is, the covariance function is found by discretely correlating $\mathbf{b}$, the vector of filter weights. This is analogous to the continuous case, where the covariance function is equal to the correlation of impulse responses.

The FIR filter with q weights and $b_0 = 1$ is equivalent to a moving average model [5] of order $q - 1$, denoted MA($q - 1$). This means that the FIR filter simply calculates a weighted moving average of the q most recent input values. The advantage of this simplicity is *unconditional* stability. That is, if the input is bounded then the output must also be bounded.

Given that we have the covariance function, we can compute the covariance matrix and find the likelihood function of the filter weights given some data. The vector of weights $\mathbf{b}$ takes on the role of the hyperparameters $\boldsymbol{\theta}$ in equation (1.17) (page 8). This enables us to model a time series by finding either the maximum likelihood (ML) or maximum *a posteriori* (MAP) vector of filter weights. We can then make predictions by using the standard Gaussian process predictive distribution equation (1.12) (page 7). Alternatively, we can make Bayesian predictions by drawing a set of weight samples from the posterior distribution over weights. This approach is equivalent to a Bayesian moving average model, and the ML approach is equivalent to fitting the moving average parameters by least squares.

The FIR filters considered thus far have all been *causal* filters. That is, the impulse response is exactly zero for all time preceding the impulse onset. Although physically unrealisable, we can consider the notion of *acausal* FIR filters, defined by a set of $2M + 1$ weights, $\{b_{-M} \dots b_0 \dots b_M\}$. The impulse response is $h(n) = \sum_{i=-M}^{M} b_i \delta(n - i)$, and is potentially non-zero for $-M \leq n \leq M$. In effect, the acausal filter can respond to an impulse before it is applied. Nevertheless, we can still pretend that a Gaussian process was generated by an acausal filter. This is so because we can transform an acausal filter into a causal filter by delaying the impulse response by a large enough amount of time. When we time shift the impulse response as such, we find that the resulting covariance function remains invariant. Overall, the covariance function for a Gaussian process produced by an acausal filter is exactly the same as a that of a causal filter that has had its output delayed by a large enough amount of time.

2.2.2 IIR Filters

An IIR filter exhibits an impulse response that is potentially non-zero for an infinite time

$$h(n) = \sum_{i=0}^{M-1} b_i \delta(n-i) + \sum_{j=1}^{N} a_j h(n-j) \tag{2.25}$$

which is equivalent to the FIR impulse response (equation 2.20) plus a recursive component which gives rise to the response's infinite nature. The filter is completely characterised by the $M+N$ filter weights $b_0 \ldots b_{M-1}, a_1 \ldots a_N$.

The output $y(n)$ in response to an input $x(n)$ is again found by discrete convolution

$$y(n) = h(n) * x(n) \tag{2.26}$$

$$= \sum_{i=0}^{M-1} b_i x(n-i) + \sum_{j=1}^{N} a_j y(n-j) \tag{2.27}$$

The IIR filter is shown diagrammatically in figure 2.4.

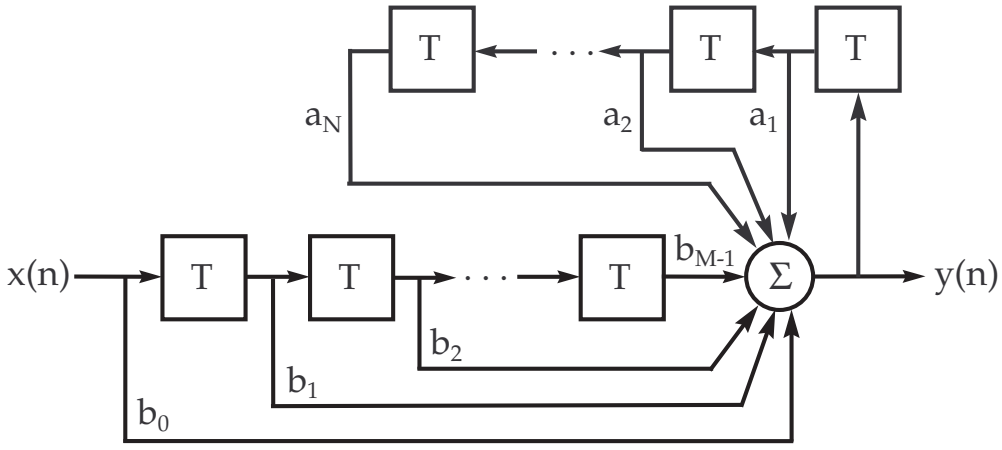

Figure 2.4: IIR filter with M feed-forward coefficients and N recursive coefficients. The input feeds into a FIR filter defined by $b_0 \ldots b_{M-1}$. The output is the sum of the FIR output and a recursive component defined by coefficients $a_1 \ldots a_N$.

If the input $x(n)$ is discrete time Gaussian white noise, then the output is a Gaussian process. This is so because we can think of the IIR filter as an FIR

filter with an infinite number of weights. The stationary covariance function for the output Gaussian process is

$$\begin{aligned}\mathrm{cov}(y(m), y(n)) &= E\left\{\sum_{i=-\infty}^{\infty} x(m-i)h(i) \sum_{j=-\infty}^{\infty} x(n-j)h(j)\right\} && (2.28)\\ &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} h(i)h(j)E\left\{x(m-i)x(n-j)\right\} && (2.29)\\ &= \sum_{j=-\infty}^{\infty} h(j)h(j+m-n) && (2.30)\end{aligned}$$

which is equal to the discrete time correlation of the impulse responses.

In general, we can not calculate equation (2.30) directly because the impulse responses are infinite. To find a closed form for the covariance function, we make use of the z-transform, $\mathcal{Z}[\cdot]$, which transforms a discrete time signal into the complex frequency domain. The correlation then becomes a multiplication:

$$\begin{aligned}\mathcal{Z}[\mathrm{cov}(y(m), y(n))] &= \mathcal{Z}\left[\sum_{j=-\infty}^{\infty} h(j)h(j+m-n)\right] && (2.31)\\ &= \mathcal{Z}[h(n) * h(-n)] && (2.32)\\ &= H(z)H(z^{-1}) && (2.33)\end{aligned}$$

where $H(z) = \mathcal{Z}[h(n)]$ and z is the complex frequency variable. The covariance function can then be recovered using the two-sided[1] inverse z-transform, $\mathcal{Z}^{-1}[\cdot]$.

As a simple example, consider the IIR filter with weights $b_0 = 1$ and $a_1 = \alpha$. From equation (2.25), we find the impulse response $h(n) = \delta(n) + \alpha h(n-1)$ for $n \geq 0$. We then find the frequency response via the z-transform:

$$\begin{aligned}\mathcal{H}(z) = Z[h(n)] &= 1 + \alpha H(z)z^{-1}\\ &= \frac{1}{1-\alpha z^{-1}} && (2.34)\end{aligned}$$

[1]The inverse z-transform is only unique when a region of convergence (ROC) is specified. The inverse is found by performing a contour integral. A contour within one ROC may result in a time signal defined for $n > 0$. We require the ROC that results in a time signal defined for $-\infty < n < \infty$, so that our auto-covariance functions are even functions. That is, we use the ROC that results in a two-sided time signal.

Now, we z-transform the covariance function $c(m) = \mathrm{cov}(y(n+m)y(n)))$

$$\mathcal{Z}[c(m)] = H(z)H(z^{-1}) \tag{2.35}$$
$$= \frac{z}{(z-\alpha)(1-\alpha z)} \tag{2.36}$$

Note that

$$\mathcal{Z}\left[\frac{\alpha^{|m|}}{1-\alpha^2}\right] = \frac{1}{1-\alpha^2}\sum_{m=-\infty}^{\infty}\alpha^{|m|}z^{-m} \tag{2.37}$$
$$= \frac{1}{1-\alpha^2}\left(\frac{\alpha z}{1-\alpha z} + \frac{z}{z-\alpha}\right) \tag{2.38}$$
$$= \frac{z}{(z-\alpha)(1-\alpha z)} \tag{2.39}$$
$$= Z[c(m)] \tag{2.40}$$

So the covariance function is given by

$$c(m) = \frac{\alpha^{|m|}}{1-\alpha^2} \tag{2.41}$$

where m is the time difference between inputs. Note that this process is a first order stationary Gaussian Markov process, otherwise known as the Ornstein-Uhlenbeck process [61].

For an IIR filter to be BIBO stable, the magnitudes of the system poles must all be less than 1. The system poles are the roots of the denominator of the transfer function $H(z) = \mathcal{Z}[h(n)]$. The poles are a function of the recursive weights $\mathbf{a}$, so not all possible filter weights will result in a stable filter. To ensure stability, we must either use known stable forms, or directly set each pole to have a magnitude less than 1, and ensure that each complex pole is accompanied by its conjugate to ensure a real impulse response. In the simple example above, the system has a real pole at $z = \alpha$ and is therefore stable for $-1 < \alpha < 1$.

The previous section described how the moving average model is equivalent to a special case of the FIR filter. Similarly, a special case of the IIR filter is equivalent to a autoregressive moving average (ARMA) model. That is, an ARMA model [5] with M moving average terms and N autoregressive terms is equivalent to the IIR filter in equation (2.25) with $b_0 = 0$

2.3 Summary

A Gaussian process can be constructed by exciting a linear filter with Gaussian white noise. If the filter is analog then we produce a Gaussian process that is continuous in time or space. For digital filters, we generate discrete time or discrete space Gaussian processes. In any case, a filter is *completely* characterised by its impulse response. We have seen in this chapter that we can directly parameterise this impulse response, rather than directly parameterising the Gaussian process's covariance function. The next chapter will make use of this result to construct sets of dependent Gaussian processes *without* having to directly specify and parameterise valid cross-covariance functions.

Chapter 3

Dependent Gaussian Processes

The previous chapter discussed the use of linear filters as generators of Gaussian processes. In this chapter, this notion is extended by using linear filters to generate sets of *dependent* Gaussian processes. This is an extension of the work of Boyle and Frean [6, 7], where dependent Gaussian processes were first defined.

3.1 Introduction

The Gaussian processes for regression implementations considered thus far model only a single output variable. Attempts to handle multiple outputs generally involve using an independent model for each output - a method known as multi-kriging [86, 40] - but such models cannot capture covariance between outputs. As an example, consider the two tightly coupled outputs shown at the top of Figure 3.2 (page 39), in which one output is simply a shifted version of the other. Here we have detailed knowledge of output 1, but sampling of output 2 is sparse. A model that treats the two outputs as independent cannot exploit their obvious similarity. Intuitively, we should make predictions about output 2 using what we learn from both outputs 1 *and* 2.

Joint predictions are possible (e.g. co-kriging [12]) but are problematic in that it is not clear how covariance functions should be defined [18, 40]. Although there are many known positive definite auto-covariance functions (e.g. Gaus-

sians and many others [1, 34]), it is difficult to define cross-covariance functions that result in positive definite covariance matrices. For example, consider the covariance matrix between two Gaussian processes $f_1(x)$ and $f_2(x)$:

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{3.1}$$

It is straightforward to specify positive definite auto-covariance functions to build the blocks $\mathbf{C}_{11}$ and $\mathbf{C}_{22}$, but it is not clear how to specify cross-covariance functions to build the cross-covariance blocks $\mathbf{C}_{12}$ and $\mathbf{C}_{21}$ such that the overall matrix $\mathbf{C}$ remains positive definite. To elaborate, $\mathbf{C}$ is a positive definite matrix if and only if $\mathbf{z}^{\mathrm{T}}\mathbf{C}\mathbf{z} > 0$ for any non-zero vector $\mathbf{z}^{\mathrm{T}} = [\mathbf{z}_1^{\mathrm{T}} \ \mathbf{z}_2^{\mathrm{T}}]$. So, if the blocks $\mathbf{C}_{11}$ and $\mathbf{C}_{22}$ are positive definite, then for $\mathbf{C}$ to be positive definite the following must hold:

$$\begin{aligned} \mathbf{z}^{\mathrm{T}}\mathbf{C}\mathbf{z} &> 0 \\ \mathbf{z}_1^{\mathrm{T}}\mathbf{C}_{11}\mathbf{z}_1 + \mathbf{z}_1^{\mathrm{T}}\mathbf{C}_{12}\mathbf{z}_2 + \mathbf{z}_2^{\mathrm{T}}\mathbf{C}_{21}\mathbf{z}_1 + \mathbf{z}_2^{\mathrm{T}}\mathbf{C}_{22}\mathbf{z}_2 &> 0 \\ \mathbf{z}_1^{\mathrm{T}}\mathbf{C}_{12}\mathbf{z}_2 &> -\tfrac{1}{2}\left\{\mathbf{z}_1^{\mathrm{T}}\mathbf{C}_{11}\mathbf{z}_1 + \mathbf{z}_2^{\mathrm{T}}\mathbf{C}_{22}\mathbf{z}_2\right\} \end{aligned} \tag{3.2}$$

The cross covariance matrix $\mathbf{C}_{12}$ is built from a cross covariance function $\mathrm{cov}_{12}(\cdot,\cdot)$ by setting the $(i,j)^{th}$ matrix element equal to $\mathrm{cov}_{12}(\mathbf{x}_{1,i}, \mathbf{x}_{2,j})$, where $\mathbf{x}_{1,i}$ is the i^{th} training input for output 1 and $\mathbf{x}_{2,j}$ is the j^{th} training input for output 2. It is not clear how to specify a non-zero $\mathrm{cov}_{12}(\cdot,\cdot)$ such that (3.2) is true for any non-zero $\mathbf{z}$. Simply setting $\mathrm{cov}_{12}(\cdot,\cdot)$ to some positive definite function will not always satisfy the requirement in (3.2).

Contrast this situation to neural network modelling, where the handling of multiple outputs is routine; it is simple to parameterise and train a hidden layer network with multiple outputs. Furthermore, due to the existence of common hidden nodes, such a network can quite simply capture the dependency between outputs that covary. If the outputs are independent, then the simplest solution is to use a separate network to model each output.

3.2 Multiple Input Multiple Output Filters

In chapter 2, we saw how Gaussian processes could be constructed by stimulating linear filters with Gaussian white noise. It is also possible to charac-

terise stable linear filters, with M-inputs and N-outputs, by a set of $M \times N$ impulse responses. We denote the response on the n^{th} output in response to an impulse on the m^{th} input as $h_{mn}(t)$. Such a filter is known as a multiple input multiple output (MIMO) filter. We stimulate the filter with M independent Gaussian white noise processes, and the resulting N outputs are by definition dependent Gaussian processes. Dependencies exist between the output processes because they are derived from a common set of input noise sources. In fact, the n^{th} output $y_n(t)$ is related to the set of M inputs $u_1(t) \dots u_M(t)$ as follows:

$$y_n(t) = \sum_{m=1}^{M} \int_{-\infty}^{\infty} h_{mn}(\tau) u_m(t-\tau) d\tau \tag{3.3}$$

Now we can model multiple dependent outputs by parameterising the set of impulse responses for a multiple output linear filter, and inferring the parameter values from data that we observe. Instead of the difficult task of specifying and parameterising auto and cross-covariance functions that imply a positive definite covariance matrix, we specify and parameterise a set of impulse responses corresponding to a MIMO filter. The only restriction is that the filter be linear and stable, and this is achieved by requiring all impulse responses to be absolutely integrable.

In chapter 2, we defined single output linear filters over $\mathbb{R}^D$ to generate Gaussian processes over $\mathbb{R}^D$. In a similar way, we can define MIMO linear filters over $\mathbb{R}^D$, and stimulate them to produce multiple dependent Gaussian processes, each defined over $\mathbb{R}^D$. We do this simply by replacing the convolution in (3.3) with a multidimensional convolution, as we did with (2.7) (page 20).

Constructing GPs by stimulating linear filters with Gaussian noise is equivalent to constructing GPs through kernel convolutions, as described below. Recall from equation (2.1) (page 18), that a linear filter's output is found by convolving the input process with the filter's impulse response. Similarly, a Gaussian process $v(\mathbf{x})$ can be constructed over an input space $\mathcal{X}$ by convolving a continuous white noise process $w(\mathbf{x})$ with a smoothing kernel $h(\mathbf{x})$, $v(\mathbf{x}) = h(\mathbf{x}) * w(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$, [24]. To this can be added a second white noise source $\eta(\mathbf{x})$, representing measurement uncertainty or system noise, and together this gives a model $y(\mathbf{x})$ for observations $\mathbf{y}$. This view of GPs is shown in graphical form in figure 3.1(a) (page 35).

Higdon [24] extended this kernel convolution framework to multiple depen-

dent output processes by assuming a single common latent process. For example, two dependent processes $v_1(\mathbf{x})$ and $v_2(\mathbf{x})$ are constructed from a shared dependence on $u(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}_0$, as follows

$$v_1(\mathbf{x}) = \int_{\mathcal{X}_0 \cup \mathcal{X}_1} h_1(\mathbf{x} - \boldsymbol{\lambda}) u(\boldsymbol{\lambda}) d\boldsymbol{\lambda} \quad \text{and} \quad v_2(\mathbf{x}) = \int_{\mathcal{X}_0 \cup \mathcal{X}_2} h_2(\mathbf{x} - \boldsymbol{\lambda}) u(\boldsymbol{\lambda}) d\boldsymbol{\lambda} \quad (3.4)$$

where $\mathcal{X} = \mathcal{X}_0 \cup \mathcal{X}_1 \cup \mathcal{X}_2$ is a union of disjoint subspaces. $v_1(\mathbf{x})$ is dependent on $u(\mathbf{x})$ when $\mathbf{x} \in \mathcal{X}_1$ but not when $\mathbf{x} \in \mathcal{X}_2$. Similarly, $v_2(\mathbf{x})$ is dependent on $u(\mathbf{x})$ when $\mathbf{x} \in \mathcal{X}_2$ but not when $\mathbf{x} \in \mathcal{X}_1$. This means $v_1(\mathbf{x})$ and $v_2(\mathbf{x})$ might possess independent components.

In what follows, we assume that Gaussian processes are the outputs of linear filters, so multiple outputs are modelled somewhat differently. Instead of assuming a single latent process defined over a union of subspaces, we assume multiple latent processes each feeding to an input of a MIMO filter. Some outputs may be dependent through a shared reliance on common latent processes, and some outputs may possess unique, independent features through a connection to a latent process that affects no other output.

We now examine some simple, toy examples to demonstrate how dependent Gaussian process models can be built for outputs that are not-independent.

3.3 Two Dependent Outputs

In the first instance, we consider the two output case, where we model data with two dependent Gaussian processes. Consider two outputs $y_1(\mathbf{x})$ and $y_2(\mathbf{x})$ over a region $\mathbb{R}^D$. We have n_1 observations of output 1 and n_2 observations of output 2, giving us data $\mathcal{D}_1 = \{\mathbf{x}_{1i}\,,\, y_{1i}\}_{i=1}^{n_1}$ and $\mathcal{D}_2 = \{\mathbf{x}_{2i}\,,\, y_{2i}\}_{i=1}^{n_2}$. We wish to learn a model from the combined data $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2\}$ in order to predict $y_1(\mathbf{x}_*)$ or $y_2(\mathbf{x}_*)$, for $\mathbf{x}_* \in \mathbb{R}^D$. For notational convenience, we collect the n_i inputs from the i^{th} data set into a matrix $\mathbf{X}_i = [\mathbf{x}_{i1} \dots \mathbf{x}_{i\,n_i}]$, and the outputs into a vector $\mathbf{y}_i = [y_{i1} \dots y_{i\,n_i}]^{\mathrm{T}}$.

As shown in figure 3.1(b), we can model each output as the linear sum of three stationary Gaussian processes. One of these (v) arises from a noise source unique to that output, under convolution with a kernel h. The second (u) is similar, but arises from a separate noise source w_0 that influences *both* outputs

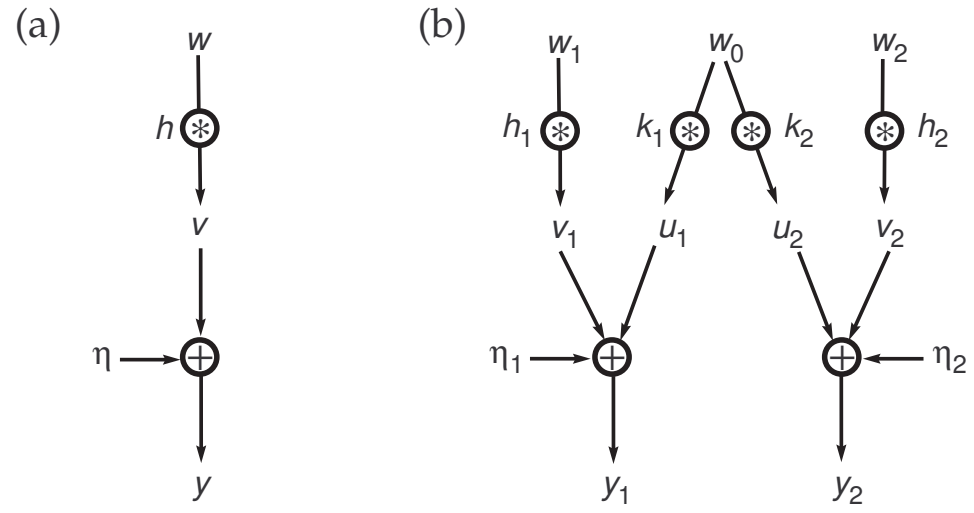

Figure 3.1: (*a*) Gaussian process prior distribution for a single output. The output y is the sum of two Gaussian white noise processes, one of which has been convolved ($*$) with a kernel (h).
(*b*) The model for two dependent outputs y_1 and y_2. The processes $w_0, w_1, w_2, \eta_1, \eta_2$ are independent Gaussian white noise sources. Notice that if w_0 is forced to zero y_1 and y_2 become independent processes as in (*a*) - we use this as a control model.

(although via different kernels, k). The third, which we'll label η is additive noise as before.

Thus we have $y_i(\mathbf{x}) = u_i(\mathbf{x}) + v_i(\mathbf{x}) + \eta_i(\mathbf{x})$, where $\eta_i(\mathbf{x})$ is a stationary Gaussian white noise process with variance, σ_i^2. The input sources $w_0(\mathbf{x}), w_1(\mathbf{x})$ and $w_2(\mathbf{x})$ are independent stationary Gaussian white noise processes. The intermediate processes $u_1(\mathbf{x}), u_2(\mathbf{x}), v_1(\mathbf{x})$ and $v_2(\mathbf{x})$ are defined as $u_i(\mathbf{x}) = k_i(\mathbf{x}) * w_0(\mathbf{x})$ and $v_i(\mathbf{x}) = h_i(\mathbf{x}) * w_i(\mathbf{x})$.

In this example, k_1, k_2, h_1 and h_2 are parameterised squared-exponential kernels

$$k_1(\mathbf{x}) = v_1 \exp\left(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}_1\mathbf{x}\right) \tag{3.5}$$

$$k_2(\mathbf{x}) = v_2 \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\mathbf{A}_2(\mathbf{x}-\boldsymbol{\mu})\right) \tag{3.6}$$

$$h_i(\mathbf{x}) = w_i \exp\left(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{B}_i\mathbf{x}\right) \tag{3.7}$$

Note that $k_2(\mathbf{x})$ is offset from zero by $\boldsymbol{\mu}$ to allow modelling of outputs that are coupled and translated relative to one another. The positive definite matrices

$\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1$ and $\mathbf{B}_2$ parameterise the kernels.

We now wish to derive the set of functions $\mathrm{cov}_{ij}^{y}(\mathbf{d}) = \mathrm{cov}_{ij}(y_i(\mathbf{x}_a), y_j(\mathbf{x}_b))$ that define the auto-covariance ($i = j$) and cross-covariance ($i \neq j$) between the outputs i and j, between arbitrary inputs $\mathbf{x}_a$ and $\mathbf{x}_b$ separated by a vector $\mathbf{d} = \mathbf{x}_a - \mathbf{x}_b$. By performing a convolution integral, (appendix A.2), $\mathrm{cov}_{ij}^{y}(\mathbf{d})$ can be expressed in a closed form , and is fully determined by the parameters of the squared-exponential kernels and the noise variances σ_1^2 and σ_2^2 as follows:

$$\mathrm{cov}_{11}^{y}(\mathbf{d}) = \mathrm{cov}_{11}^{u}(\mathbf{d}) + \mathrm{cov}_{11}^{v}(\mathbf{d}) + \delta_{ab}\sigma_1^2 \tag{3.8}$$

$$\mathrm{cov}_{12}^{y}(\mathbf{d}) = \mathrm{cov}_{12}^{u}(\mathbf{d}) \tag{3.9}$$

$$\mathrm{cov}_{21}^{y}(\mathbf{d}) = \mathrm{cov}_{21}^{u}(\mathbf{d}) \tag{3.10}$$

$$\mathrm{cov}_{22}^{y}(\mathbf{d}) = \mathrm{cov}_{22}^{u}(\mathbf{d}) + \mathrm{cov}_{22}^{v}(\mathbf{d}) + \delta_{ab}\sigma_2^2 \tag{3.11}$$

where

$$\mathrm{cov}_{ii}^{u}(\mathbf{d}) = \frac{\pi^{\frac{D}{2}} v_i^2}{\sqrt{|\mathbf{A}_i|}} \exp\left(-\frac{1}{4}\mathbf{d}^{\mathrm{T}}\mathbf{A}_i\mathbf{d}\right) \tag{3.12}$$

$$\mathrm{cov}_{12}^{u}(\mathbf{d}) = \frac{(2\pi)^{\frac{D}{2}} v_1 v_2}{\sqrt{|\mathbf{A}_1 + \mathbf{A}_2|}} \exp\left(-\frac{1}{2}(\mathbf{d} - \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}(\mathbf{d} - \boldsymbol{\mu})\right) \tag{3.13}$$

$$\mathrm{cov}_{21}^{u}(\mathbf{d}) = \frac{(2\pi)^{\frac{D}{2}} v_1 v_2}{\sqrt{|\mathbf{A}_1 + \mathbf{A}_2|}} \exp\left(-\frac{1}{2}(\mathbf{d} + \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}(\mathbf{d} + \boldsymbol{\mu})\right) = \mathrm{cov}_{12}^{u}(-\mathbf{d}) \tag{3.14}$$

$$\mathrm{cov}_{ii}^{v}(\mathbf{d}) = \frac{\pi^{\frac{D}{2}} w_i^2}{\sqrt{|\mathbf{B}_i|}} \exp\left(-\frac{1}{4}\mathbf{d}^{\mathrm{T}}\mathbf{B}_i\mathbf{d}\right) \tag{3.15}$$

where $\boldsymbol{\Sigma} = \mathbf{A}_1(\mathbf{A}_1 + \mathbf{A}_2)^{-1}\mathbf{A}_2 = \mathbf{A}_2(\mathbf{A}_1 + \mathbf{A}_2)^{-1}\mathbf{A}_1$, and D is the problem dimensionality.

Given $\mathrm{cov}_{ij}^{y}(\mathbf{d})$, we can construct the covariance matrices $\mathbf{C}_{11}, \mathbf{C}_{12}, \mathbf{C}_{21}$, and $\mathbf{C}_{22}$ as follows

$$\mathbf{C}_{ij} = \begin{bmatrix} \mathrm{cov}_{ij}^{y}(\mathbf{x}_{i1} - \mathbf{x}_{j1}) & \cdots & \mathrm{cov}_{ij}^{y}(\mathbf{x}_{i1} - \mathbf{x}_{j\,n_j}) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}_{ij}^{y}(\mathbf{x}_{i\,n_i} - \mathbf{x}_{j1}) & \cdots & \mathrm{cov}_{ij}^{y}(\mathbf{x}_{i\,n_i} - \mathbf{x}_{j\,n_j}) \end{bmatrix} \tag{3.16}$$

Together these define the positive definite symmetric covariance matrix $\mathbf{C}$ for the *combined* output data $\mathcal{D}$:

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{3.17}$$

We define a set of hyperparameters $\boldsymbol{\theta}$ that parameterise $\{v_1, v_2, w_1, w_2, \mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \boldsymbol{\mu}, \sigma_1, \sigma_2\}$. Now, we can calculate the log-likelihood

$$\mathcal{L} = -\tfrac{1}{2}\log\left|\mathbf{C}\right| - \tfrac{1}{2}\,\mathbf{y}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{y} - \frac{n_1 + n_2}{2}\log 2\pi \tag{3.18}$$

where $\mathbf{y} = [y_{11} \cdots y_{1\,n_1} \mid y_{21} \cdots y_{2\,n_2}]^{\mathrm{T}}$ and $\mathbf{C}$ is a function of $\boldsymbol{\theta}$ and $\mathcal{D}$. This is similar in form to equation (1.20) (page 8), which is the log-likelihood for a *single* Gaussian process.

Learning a model now corresponds to either maximising the log-likelihood $\mathcal{L}$, or maximising the posterior probability $p(\boldsymbol{\theta} \mid \mathcal{D})$. Alternatively, we could simulate the predictive distribution for y by taking samples from the joint distribution $p(y, \boldsymbol{\theta} \mid \mathcal{D})$, using Markov Chain Monte Carlo methods [57, 86, 47].

For a single output Gaussian process model, we can make predictions using equations (1.13) and (1.14) (page 8). Similarly, for the two output case, the predictive distribution at a point $\mathbf{x}_*$ on the i^{th} output is Gaussian with mean $m(\mathbf{x}_*)$ and variance $v(\mathbf{x}_*)$ given by

$$m(\mathbf{x}_*) = \mathbf{k}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{y} \tag{3.19}$$

$$\text{and} \quad v(\mathbf{x}_*) = \kappa - \mathbf{k}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{k} \tag{3.20}$$

$$\text{where} \quad \kappa = \mathrm{cov}^{y}_{ii}(\mathbf{0}) = v_i^2 + w_i^2 + \sigma_i^2 \tag{3.21}$$

and

$$\mathbf{k} = \begin{bmatrix} \mathrm{cov}^{y}_{i1}(\mathbf{x}_* - \mathbf{x}_{11}) \\ \vdots \\ \mathrm{cov}^{y}_{i1}(\mathbf{x}_* - \mathbf{x}_{1\,n_1}) \\ \mathrm{cov}^{y}_{i2}(\mathbf{x}_* - \mathbf{x}_{21}) \\ \vdots \\ \mathrm{cov}^{y}_{i2}(\mathbf{x}_* - \mathbf{x}_{2\,n_2}) \end{bmatrix} \tag{3.22}$$

3.3.1 Example 1 - Strongly dependent outputs over $\mathbb{R}$

Consider two outputs, observed over a $1D$ input space. Let $\mathbf{A}_i = \exp(f_i)$, $\mathbf{B}_i = \exp(g_i)$, and $\sigma_i = \exp(\beta_i)$. Our hyperparameters become $\boldsymbol{\theta} = \{v_1, v_2, w_1, w_2, f_1, f_2, g_1, g_2, \mu, \beta_1, \beta_2\}$ where each element of $\boldsymbol{\theta}$ is a scalar.

Gaussian prior distributions over hyperparameters $\boldsymbol{\theta}$ were set as follows:

$$\begin{aligned} v_i &\sim \mathcal{N}(0, 0.5^2) \\ w_i &\sim \mathcal{N}(0, 0.5^2) \\ f_i &\sim \mathcal{N}(3.5, 1^2) \\ g_i &\sim \mathcal{N}(3.5, 1^2) \\ \mu &\sim \mathcal{N}(0, 0.5^2) \\ \beta_i &\sim \mathcal{N}(-4, 0.75^2) \end{aligned}$$

Training data was generated by evaluating the two functions shown by the dotted lines in figure 3.2 at a number of points, and then adding Gaussian noise with $\sigma = 0.025$. Note that output 2 is simply a translated version of output 1 corrupted with independent noise. $n = 48$ data points were generated by taking $n_1 = 32$ samples from output 1 and $n_2 = 16$ samples from output 2. The samples were taken uniformly in the interval $[-1, 1]$ for both outputs, except that those from -0.15 to 0.65 were missing for output 2. The model was built by maximising $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})$ using a multi-start conjugate gradient algorithm [54], with 5 starts, sampling from $p(\boldsymbol{\theta})$ for initial conditions. In other words, the model was built by finding the MAP hyperparameters.

The resulting dependent model is shown in figure 3.2 along with an independent (control) model with no coupling (as in figure 3.1(b) with $w_0 = 0$). Observe that the dependent model has learnt the coupling and translation between the outputs, and has filled in output 2 where samples are missing. The control model cannot achieve such "in-filling" as it is consists of two *independent* Gaussian processes.

3.3.2 Example 2 - Strongly dependent outputs over $\mathbb{R}^2$

Consider two outputs, observed over a $2D$ input space. Let

$$\mathbf{A}_i = \frac{1}{\alpha_i^2}\mathbf{I} \qquad \mathbf{B}_i = \frac{1}{\tau_i^2}\mathbf{I} \qquad \text{where } \mathbf{I} \text{ is the identity matrix.}$$

Furthermore, let $\sigma_i = \exp(\beta_i)$. In this toy example, we set $\boldsymbol{\mu} = \mathbf{0}$, so our hyperparameters become $\boldsymbol{\theta} = \{v_1, v_2, w_1, w_2, \alpha_1, \alpha_2, \tau_1, \tau_2\beta_1, \beta_2\}$ where each element of $\boldsymbol{\theta}$ is a scalar.

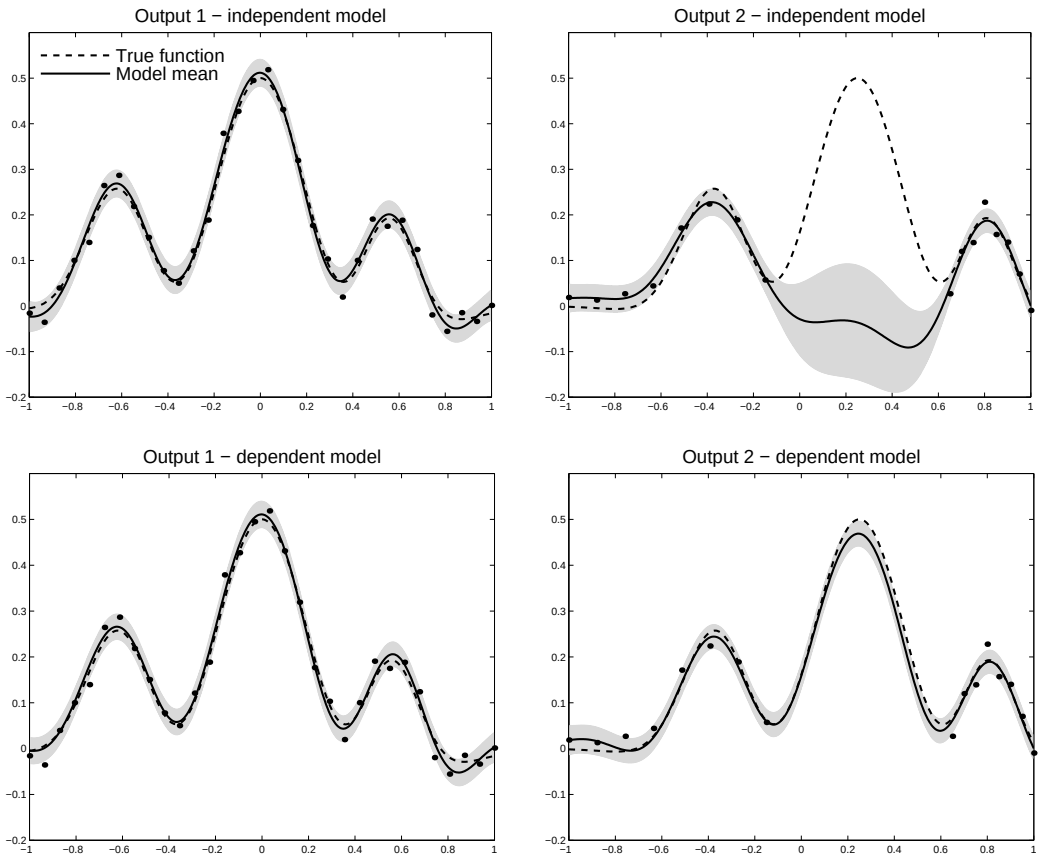

Figure 3.2: Strongly dependent outputs where output 2 is simply a translated version of output 1, with independent Gaussian noise, $\sigma = 0.025$. The solid lines represent the model, the dotted lines are the true function, and the dots are samples. The shaded regions represent 1σ error bars for the model prediction. *(top)* Independent model of the two outputs. *(bottom)* Dependent model.

Gaussian prior distributions were set over $\boldsymbol{\theta}$, as follows:

$$
\begin{aligned}
v_i &\sim \mathcal{N}(1, 0.5^2) \\
w_i &\sim \mathcal{N}(0, 0.5^2) \\
\alpha_i &\sim \mathcal{N}(0.3, 0.1^2) \\
\tau_i &\sim \mathcal{N}(0.3, 0.1^2) \\
\beta_i &\sim \mathcal{N}(-3.7, 0.5^2)
\end{aligned}
$$

Training data was generated by evaluating the function shown on the left of figure 3.3 at $n = 117$ points, and adding Gaussian noise with $\sigma = 0.025$. $n_1 = 81$ samples were placed in the training set $\mathcal{D}_1$ and the remaining $n_2 = 36$

samples formed training set $\mathcal{D}_2$. The training inputs from these sets formed uniform lattices over the region $[-0.9, 0.9] \otimes [-0.9, 0.9]$. Overall, the two sets of training data were generated from the same underlying function, and differ only due to noise contributions and the placement of the inputs.

The model was built by maximising $p(\boldsymbol{\theta}|\mathcal{D})$ to find the MAP hyperparameters as before.

The dependent model is shown in Figure 3.3 along with an independent control model. The dependent model has filled in output 2 where there a no corresponding samples. Again, the control model cannot achieve such "infilling" as it is consists of two *independent* Gaussian processes.

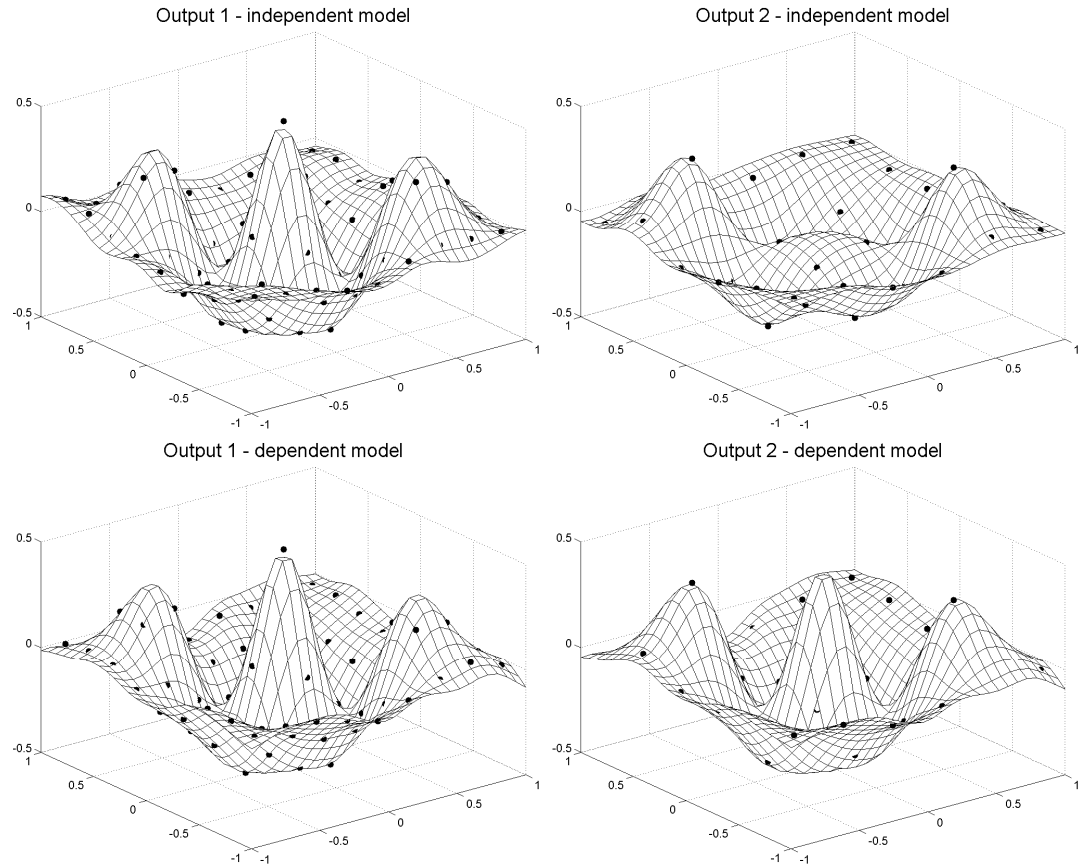

Figure 3.3: Strongly dependent outputs where output 2 is simply a copy of output 1, with independent Gaussian noise. *(top)* Independent model of the two outputs. *(bottom)* Dependent model. Output 1 is modelled well by both models. Output 2 is modelled well only by the dependent model.

3.3.3 Example 3 - Partially Coupled Outputs

Example 1 in section 3.3.1 and example 2 in section 3.3.2 discuss Gaussian process models of two outputs that are fully dependent on one another, apart from a noise contribution. In example 1, output 2 is simply a translated version of output 1 corrupted with independent noise. In example 2, the two outputs are identical, apart from noise. This section presents a $1D$ example with two outputs that are not simply related by translation. The outputs are only *partially* dependent as shown in the top panel of Figure 3.4 (page 42). Observe that the two outputs are significantly dependent, but not fully dependent.

For the model, the hyperparameters $\boldsymbol{\theta}$ are the same as those from section 3.3.1 with Gaussian prior distributions. 48 data points were generated by taking 32 samples from output 1 and 16 samples from output 2. The samples were uniformly spaced in the interval $[-1, 1]$ for output 1 but only in $[-1, 0]$ for output 2. All samples were taken with additive Gaussian noise, $\sigma = 0.025$.

As before, $p(\boldsymbol{\theta}|\mathcal{D})$ was maximised and predictions were compared with those from an independent model. As Figure 3.4 shows, the dependent model has learnt the coupling between the outputs, and attempts to fill in output 2 where samples are missing. The "in-filling" is not as striking as the previous examples because output 2 possesses an independent component, but is much better than the default GP model.

3.4 More than two Outputs

The MIMO framework described here for constructing GPs is capable of modelling N-outputs, each defined over a D-dimensional input space. In general, for $\mathbf{x} \in \mathbb{R}^D$, we can define a model where we assume M-independent Gaussian white noise processes $w_1(\mathbf{x}) \dots w_M(\mathbf{x})$, N-outputs $y_1(\mathbf{x}) \dots y_N(\mathbf{x})$, and $M \times N$ kernels. The kernel $k_{mn}(\mathbf{x})$ defines the connection from input m to output n. The auto-covariance ($i = j$) and cross-covariance ($i \neq j$) functions between output processes i and j become (appendix A.1)

$$\mathrm{cov}_{ij}^{y}(\mathbf{d}) = \sum_{m=1}^{M} \int_{\mathbb{R}^D} k_{mi}(\mathbf{x}) k_{mj}(\mathbf{x} + \mathbf{d}) d^D\mathbf{x} \tag{3.23}$$

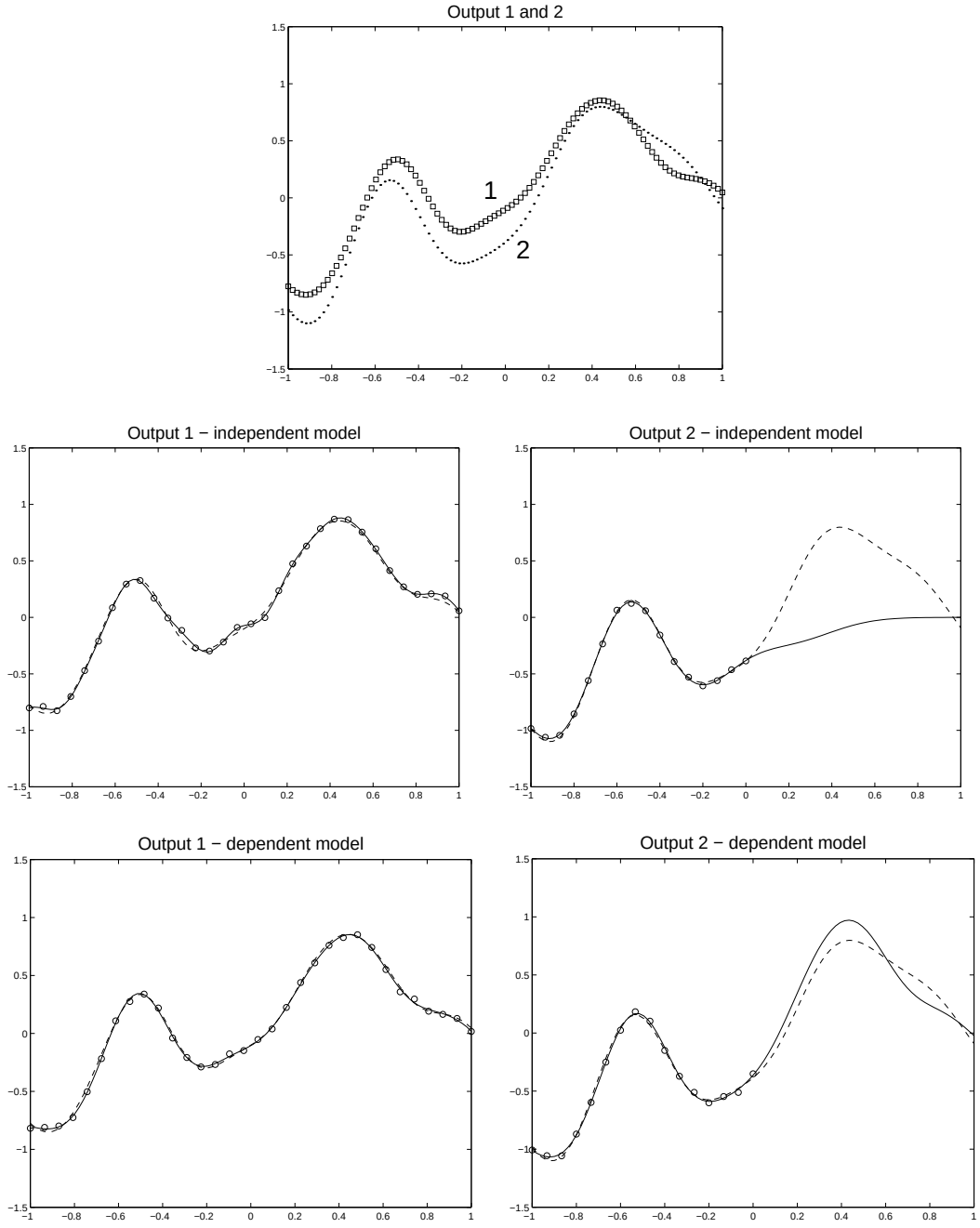

Figure 3.4: Dependent but unique outputs. *(top)* Outputs 1 and 2 are overlaid to illustrate the (partial) coupling between them. Note that these outputs are not entirely dependent. *(middle)* Independent model of the two outputs. *(bottom)* Dependent model.

and the matrix defined by equation (3.17) is extended to

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \dots & \mathbf{C}_{1N} \\ \vdots & \ddots & \vdots \\ \mathbf{C}_{N1} & \dots & \mathbf{C}_{NN} \end{bmatrix} \tag{3.24}$$

If we have n_i observations of output i, then we have $n = \sum_{i=1}^{N} n_i$ observations in total and $\mathbf{C}$ is a $n \times n$ matrix. Our combined data set becomes $\mathcal{D} = \{\mathcal{D}_1 \cdots \mathcal{D}_N\}$, where $\mathcal{D}_i = \{(\mathbf{x}_{i1}\,,\, y_{i1}) \cdots (\mathbf{x}_{i\,n_i}\,,\, y_{i\,n_i})\}$.

The log-likelihood becomes

$$\mathcal{L} = -\frac{1}{2}\log|\mathbf{C}| - \frac{1}{2}\mathbf{y}^T\mathbf{C}^{-1}\mathbf{y} - \frac{n}{2}\log 2\pi \tag{3.25}$$

$$\text{where} \quad \mathbf{y}^T = \begin{bmatrix} (y_{11} \cdots y_{1\,n_1}) & \cdots & (y_{i1} \cdots y_{i\,n_i}) & \cdots & (y_{N1} \cdots y_{N\,n_N}) \end{bmatrix}$$

and the predictive distribution at a point $\mathbf{x}_*$ on the i^{th} output is Gaussian with mean $m(\mathbf{x}_*)$ and variance $v(\mathbf{x}_*)$ given by

$$m(\mathbf{x}_*) = \mathbf{k}^T\mathbf{C}^{-1}\mathbf{y} \tag{3.26}$$

$$\text{and} \quad v(\mathbf{x}_*) = \kappa - \mathbf{k}^T\mathbf{C}^{-1}\mathbf{k} \tag{3.27}$$

$$\text{where} \quad \kappa = \mathrm{cov}_{ii}^{y}(\mathbf{0}) = v_i^2 + w_i^2 + \sigma_i^2$$

$$\text{and} \quad \mathbf{k}^T = \begin{bmatrix} \mathbf{k}_1^T \cdots \mathbf{k}_j^T \cdots \mathbf{k}_N^T \end{bmatrix}$$

$$\text{and} \quad \mathbf{k}_j^T = \begin{bmatrix} \mathrm{cov}_{ij}^{y}(\mathbf{x}_* - \mathbf{x}_{j1}) \dots \mathrm{cov}_{ij}^{y}(\mathbf{x}_* - \mathbf{x}_{j\,n_j}) \end{bmatrix}$$

3.5 Time Series Forecasting

Consider the observation of multiple time series, where some of the series lead or predict the others. For example, consider the set of three time series simulated for 100 steps each shown in figure 3.5 where series 3 is positively coupled to a lagged version of series 1 ($lag = 0.5$) and negatively coupled to a lagged version of series 2 ($lag = 0.6$).

The series were generated by sampling from a dependent Gaussian process

with the following Gaussian covariance functions (see appendix A.2):

$$\mathrm{cov}_{11}(t_a, t_b) = w^2 \exp\left(-\tfrac{1}{4}\tfrac{(t_a - t_b)^2}{r^2}\right) + \delta_{ab}\sigma^2 \tag{3.28}$$

$$\mathrm{cov}_{12}(t_a, t_b) = 0 \tag{3.29}$$

$$\mathrm{cov}_{13}(t_a, t_b) = wv_1 \exp\left(-\tfrac{1}{4}\tfrac{(t_a - t_b - \mu_1)^2}{r^2}\right) \tag{3.30}$$

$$\mathrm{cov}_{21}(t_a, t_b) = 0 \tag{3.31}$$

$$\mathrm{cov}_{22}(t_a, t_b) = w^2 \exp\left(-\tfrac{1}{4}\tfrac{(t_a - t_b)^2}{r^2}\right) + \delta_{ab}\sigma^2 \tag{3.32}$$

$$\mathrm{cov}_{23}(t_a, t_b) = wv_2 \exp\left(-\tfrac{1}{4}\tfrac{(t_a - t_b - \mu_2)^2}{r^2}\right) \tag{3.33}$$

$$\mathrm{cov}_{31}(t_a, t_b) = wv_1 \exp\left(-\tfrac{1}{4}\tfrac{(t_a - t_b + \mu_1)^2}{r^2}\right) \tag{3.34}$$

$$\mathrm{cov}_{32}(t_a, t_b) = wv_2 \exp\left(-\tfrac{1}{4}\tfrac{(t_a - t_b + \mu_2)^2}{r^2}\right) \tag{3.35}$$

$$\mathrm{cov}_{33}(t_a, t_b) = (v_1^2 + v_2^2) \exp\left(-\tfrac{1}{4}\tfrac{(t_a - t_b)^2}{r^2}\right) + \delta_{ab}\sigma^2 \tag{3.36}$$

where $w = 0.9, v_1 = 0.5, v_2 = -0.6, r = 0.15, \mu_1 = -0.5, \mu_2 = -0.6$, and $\sigma = 0.025$. The samples were evaluated at 100 input points $t_1 \ldots t_{100}$, which were uniformly spread over the interval $[0, 7.8]$. The evaluation at each input produced 3 outputs to give a total of 300 time series data points.

Given the 300 observations, a dependent GP model of the three time series was built and compared with three independent GP models. The dependent model used the covariance functions above with maximum likelihood values for $w, v_1, v_2, r, \mu_1, \mu_2$ and σ. The independent models used the same set of covariance functions but with $\mathrm{cov}_{ij} = 0$ for $i \neq j$, and maximum likelihood values for w, v_1, v_2, r and σ.

The dependent GP model incorporated a prior belief that series 3 was coupled to series 1 and 2, but with the lags unknown. The independent GP model assumed no coupling between its outputs, and consisted of three independent GP models. The models were non-recursively queried for forecasts of 10 future values of series 3. It is clear from figure 3.5 that the dependent GP model does a far better job at forecasting the dependent series 3. The independent model becomes inaccurate after just a few time steps into the future. This inaccuracy is expected, as knowledge of series 1 and 2 is required to accurately predict series 3. The dependent GP model performs well as it has learnt that series 3 is positively coupled to a lagged version of series 1 and negatively coupled to a lagged version of series 2.

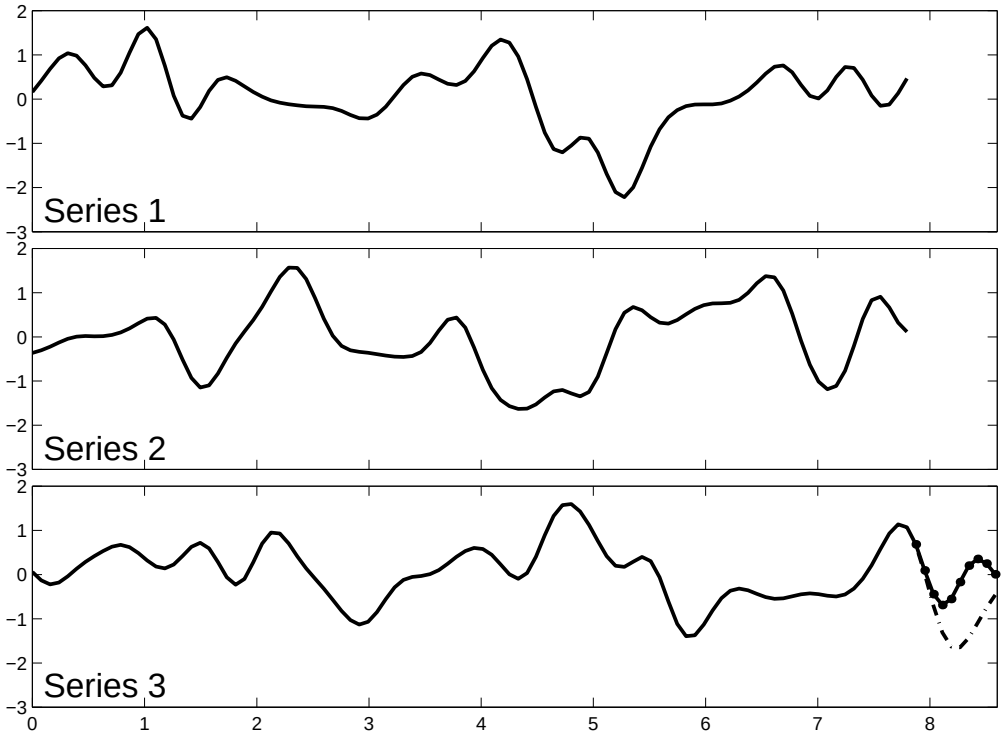

Figure 3.5: Three coupled time series, where series 1 and series 2 predict series 3. Forecasting for series 3 begins after 100 time steps where $t = 7.8$. The dependent model forecast is shown with a solid line, and the independent (control) forecast is shown with a broken line. The dependent model does a far better job at forecasting the next 10 steps of series 3 (black dots).

3.6 Non-stationary kernels

The kernels used in (3.23) need not be Gaussian, and need not be spatially invariant, or stationary. To reiterate, we require kernels that are absolutely integrable, $\int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} |k(\mathbf{x})| d^D\mathbf{x} < \infty$ which correspond to BIBO stable linear filters. This provides a large degree of flexibility, as it is straightforward to specify a kernel that is absolutely integrable. Initially, one might think that a Gaussian process model could be built by specifying *any* absolutely integrable kernel. However, if the resultant Gaussian process is to be used for modelling, then a closed form of the covariance function $\mathrm{cov}_{ij}^{y}(\mathbf{d})$ is required (so that the covariance matrix can be constructed and differentiated with respect to the hyperparameters). Unfortunately, for some non-Gaussian or non-

stationary kernels, the convolution integral (2.8) (page 20), may not be analytically tractable, hence a covariance function will not be available.

3.7 MIMO Digital Filters

Chapter 2, showed how to construct discrete time or space Gaussian processes by exciting digital filters with Gaussian white noise. In a similar fashion, discrete time or space *dependent* Gaussian processes can be constructed by stimulating MIMO digital filters with Gaussian white noise. Two cases are considered below: MIMO FIR filters, and MIMO IIR filters.

3.7.1 MIMO FIR Filters

Consider two FIR filters of the same order, defined by impulse responses

$$h_1(n) = \sum_{i=0}^{M-1} a_i \delta(n-i) \tag{3.37}$$

$$h_2(n) = \sum_{i=0}^{M-1} b_i \delta(n-i) \tag{3.38}$$

If both filters are stimulated with Gaussian white noise, then two dependent Gaussian processes are generated with a auto/cross covariance function $c_{ij}(m) = \mathrm{cov}(y_i(n+m), y_j(n))$.

$$c_{11}(m) = \sum_{j=0}^{M-1} a_j a_{j+m} \tag{3.39}$$

$$c_{22}(m) = \sum_{j=0}^{M-1} b_j b_{j+m} \tag{3.40}$$

$$c_{12}(m) = \sum_{j=0}^{M-1} a_{j+m} b_j = \sum_{j=0}^{M-1} a_j b_{j-m} = c_{21}(-m) \tag{3.41}$$

$$c_{21}(m) = \sum_{j=0}^{M-1} a_j b_{j+m} = \sum_{j=0}^{M-1} a_{j-m} b_j = c_{12}(-m) \tag{3.42}$$

It is easy to generalise to an M-input N-output FIR filter by finding the N^2 covariance functions $\{c_{11}(m) \dots c_{1N}(m)\} \dots \{c_{N1}(m) \dots c_{NN}(m)\}$.

As an example, consider the impulse responses shown by the joined-dots in figure 3.6. These impulse responses correspond to two FIR filters with 25 weights each. Each filter was excited with the same Gaussian white noise sequence to produce two dependent Gaussian processes, depicted in figure 3.7. Note that filter 2 has a sharper impulse response, and this becomes evident as a less smooth output process. Filter 1 produces a smoother output, because it has a smoother impulse response. In the extreme, if we had the sharpest possible impulse response, namely an impulse, then the output would be a Gaussian white noise process. Although it is difficult to see by inspecting the processes, they are in fact highly dependent - process 2 is negatively coupled to a lagged version of process 1. This is because $h_1(n)$ is positive and instant, but $h_2(n)$ is negative and lagged.

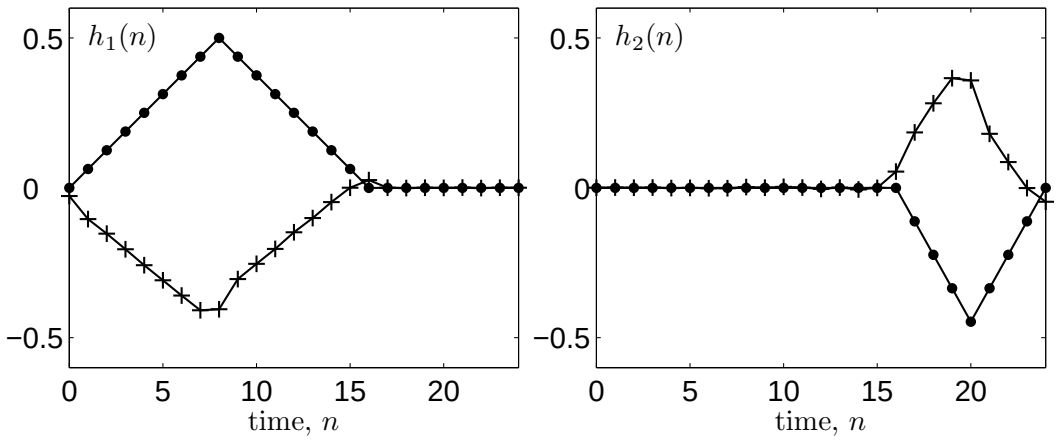

Figure 3.6: Impulse responses for the two FIR filters from the generative model (dots), and the filters that were learnt from the training data (crosses) as described in the text. The responses are very similar except for a -1 scaling factor. All filters consist of 25 weights.

The two dependent Gaussian processes were sampled to produce two sets of time series, each with 100 points. This data was used to find MAP filter weights for two FIR filters with 25 weights each, and Gaussian distributions over the weights $p(a_i), p(b_j) \sim \mathcal{N}(0, 1)$. In other words, two filters with 25 weights each were used to produce the training data in figure 3.7, which was then modelled using two discrete time dependent Gaussian processes, based on FIR filters with 25 weights each.

The MAP weights for the two filters were found using multiple runs of the conjugate gradient algorithm [54], starting from a random vector of weights,

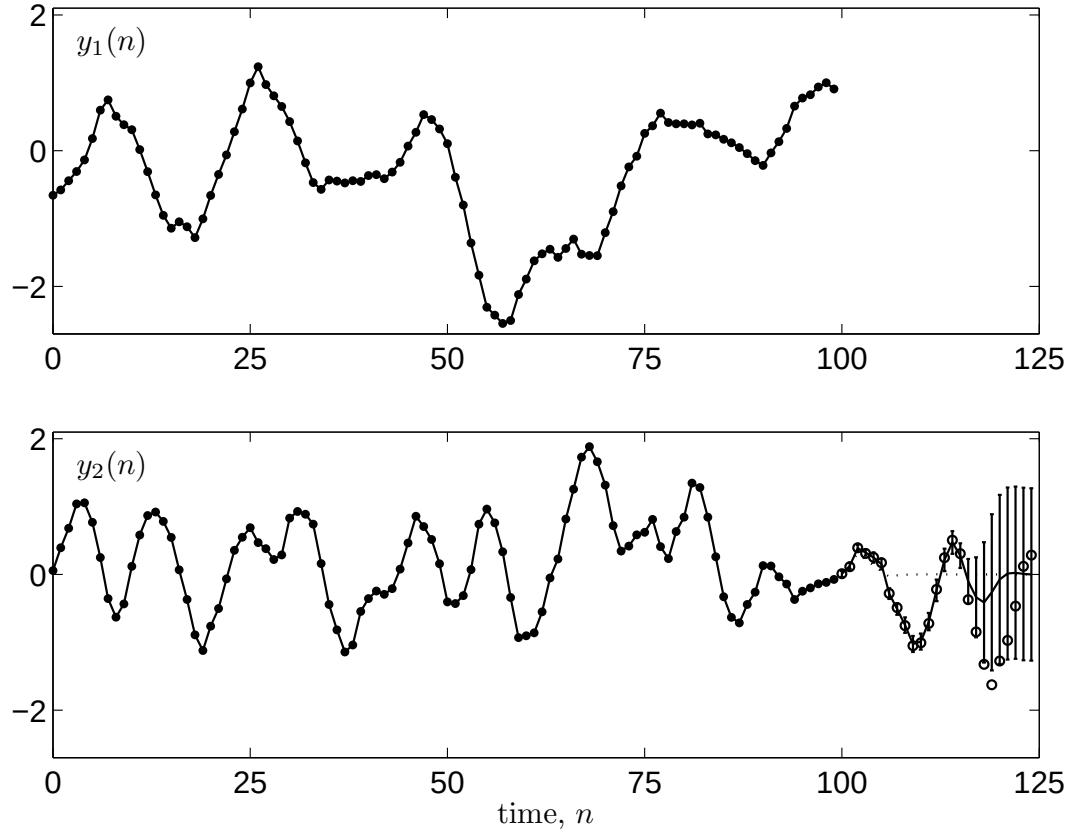

Figure 3.7: Discrete time dependent Gaussian processes produced from two FIR filters stimulated with the same noise sequence. The first 100 time steps form the training data. After 100 time steps, the bottom panel shows the continuation of series 2 (circles), along with the dependent GP prediction (solid line) and the independent GP prediction (dotted line). The error bars show the 95% confidence interval for the dependent GP's prediction.

$w_i \sim \mathcal{N}(0, 0.1^2)$. The MAP weight vectors were used to find the auto/cross covariance functions of the model (equations 3.39 to 3.42). These are plotted in figure 3.8, along with the corresponding functions from the generative model. From the similarity of the plots, we see that the model has learnt the correct dependencies between the training series.

It is interesting to examine the form of the impulse responses of the filters that have been learnt. These are shown together in figure 3.6, along with the impulse responses of the filters from the generative model. Note that the learnt impulse responses are inverted relative to the generative model. This does not affect the covariance structure however - all that is required is that the impulse responses are of opposite polarity, and have appropriate lags.

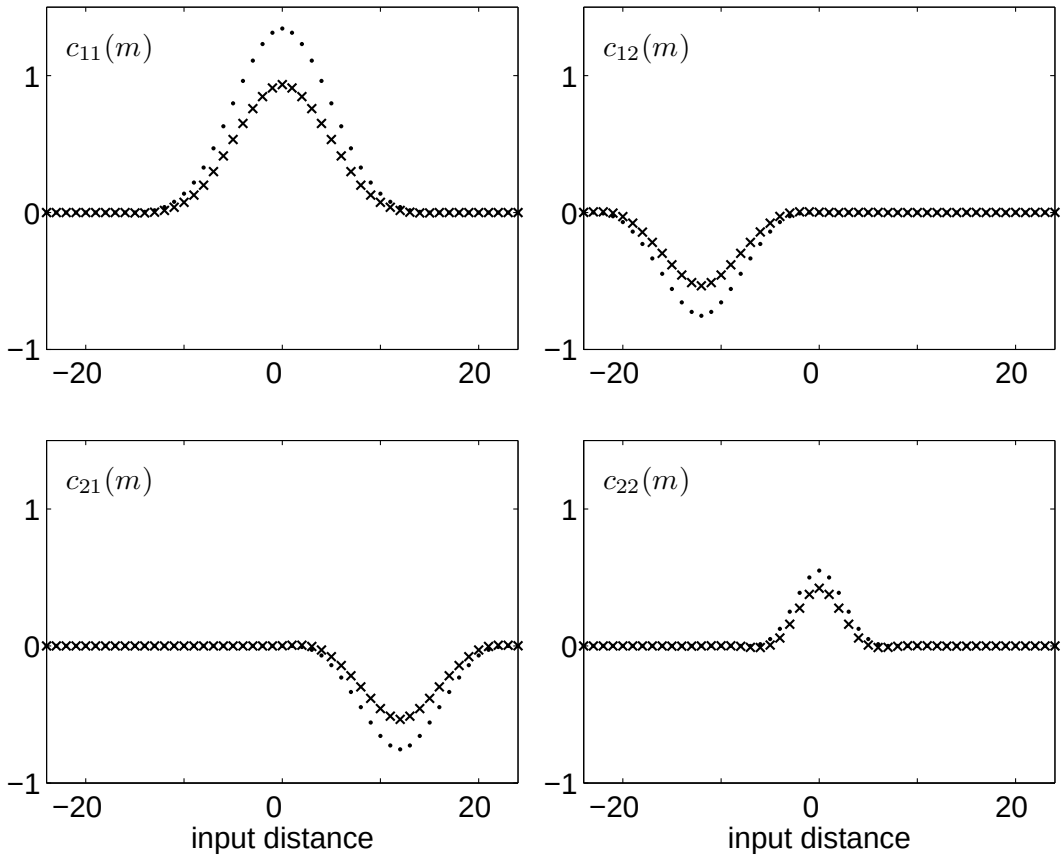

Figure 3.8: Auto and cross covariance functions for a discrete time Gaussian process, generated from the outputs of two FIR filters excited by the same noise source. The dots show the expected covariance for input distances $-24 \leq m \leq 24$, found from discretely correlating the appropriate impulse responses. The crosses show the covariance functions learnt from the data.

When the covariance functions are calculated, the -1 scaling factor between the empirical and generative impulse response cancels out.

Consider now using the dependent GP model just inferred to forecast the future 25 values for series 2. This forecast is shown in the bottom panel of figure 3.7. For comparison, the forecast using an independent GP model is also shown. It is clear that the dependent model does a far better job at predicting series 2, which is as expected given that series 2 is generated using a lagged version of the information used to produce series 1. Note that the model uncertainty is low for the first 16 steps of the forecast, but then increases for subsequent predictions. This is explained by the fact that the learnt impulse response for filter 2 lags by 16 time steps (figure 3.6). The independent model

forecasts well for only 6 future time steps. This is because points from series 2 that are separated by more than 6 time steps are uncorrelated, as can be seen by examining $c_{22}(m)$ in figure 3.8.

3.7.2 MIMO IIR Filters

Consider two IIR filters, with impulse responses

$$h_1(n) = \begin{cases} \alpha^n & n \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad h_2(n) = \begin{cases} \beta^{n-k} & n \geq k \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.43}$$

where $|\alpha|, |\beta| < 1$, and filter 2 exhibits a lag of k before responding to the impulse.

If we excite both filters with the same Gaussian white noise process, then we generate two dependent output processes, $y_1(n)$ and $y_2(n)$. The auto/cross covariance functions for input separation $-\infty < m < \infty$ are:

$$c_{11}(m) = \frac{\alpha^{|m|}}{1-\alpha^2} \tag{3.44}$$

$$c_{22}(m) = \frac{\beta^{|m|}}{1-\beta^2} \tag{3.45}$$

$$c_{12}(m) = \mathcal{Z}^{-1}[H_1(z^{-1})H_2(z)] = \begin{cases} \frac{\alpha^{|m-k|}}{1-\alpha\beta} & m \geq k \\ \frac{\beta^{|m-k|}}{1-\alpha\beta} & m < k \end{cases} \tag{3.46}$$

$$c_{21}(m) = \mathcal{Z}^{-1}[H_1(z)H_2(z^{-1})] = \begin{cases} \frac{\alpha^{|m+k|}}{1-\alpha\beta} & m \leq -k \\ \frac{\beta^{|m+k|}}{1-\alpha\beta} & m > -k \end{cases} \tag{3.47}$$

with $c_{21}(m) = c_{12}(-m)$ as required.

In general, we can assume M noise inputs, and generate N dependent Gaussian processes as outputs.

A multiple output IIR filter, stimulated with Gaussian white noise, outputs a vector of dependent Gaussian processes. Such a filter is otherwise known as a vector autoregressive moving-average (VARMA) model [32].

3.7.3 Toeplitz Matrices

The covariance matrix of a regularly sampled stationary time series is a symmetric Toeplitz matrix [77, 78] - a symmetric matrix with constant diago-

nals. For example, matrix $\mathbf{C}$ is a symmetric Toeplitz matrix if the $(i,j)^{th}$ entry $C_{i,j} = c(i-j) = c(j-i)$ where $c(\cdot)$ is a stationary covariance function. The diagonals are constant because $C_{i+k,j+k} = c(i-j) = C_{i,j}$. An $n \times n$ symmetric Toeplitz matrix has at most n distinct entries, $c(0) \dots c(n-1)$. As a result, it is not surprising that it is easier to invert than an arbitrary covariance matrix which may have up to $\frac{1}{2}n(n+1)$ distinct entries. In fact, a $n \times n$ Toeplitz matrix can be inverted exactly using the Trench algorithm in $\mathcal{O}(n^2)$ [79], and approximately in $\mathcal{O}(n \log n)$ [11]

For dependent Gaussian processes, produced from stationary FIR filters, the covariance matrix is block-Toeplitz. For example, for 2 stationary dependent processes the covariance matrix for all the data is

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{3.48}$$

where $\mathbf{C}_{11}, \mathbf{C}_{12}, \mathbf{C}_{21}, \mathbf{C}_{22}$ are Toeplitz matrices, but in general $\mathbf{C}$ is not Toeplitz. Therefore, the Trench algorithm is not applicable. Fortunately, for a set of m dependent processes, each with n regular samples, one can use a generalisation of the Trench algorithm [84] to find an inverse in $\mathcal{O}(m^3 n^2)$.

3.8 Multidimensional Digital Filters and Random Fields

The treatment of MIMO digital filters so far has assumed that the all filters are $1D$, hence filters of time signals. There may be circumstances when we wish to produce models of higher dimensional data sets, that have been sampled over discrete space. In such cases, we could specify our models as being the outputs of multidimensional MIMO digital filters. In other words, our models are Gaussian random fields.

A 2 dimensional FIR filter has an impulse response

$$h(n_1, n_2) = \sum_{i=0}^{M_1-1} \sum_{j=0}^{M_2-1} b_{ij} \delta(n_1 - i) \delta(n_2 - j) \tag{3.49}$$

The filter is fully characterised by the matrix $\mathbf{B}$ of $M_1 \times M_2$ coefficients, where the $(i,j)^{th}$ coefficient is b_{ij}.

An example of a $2D$ dependent Gaussian process constructed by exciting a two output FIR filter with Gaussian white noise is shown in figure 3.9. In this example, the impulse responses for each output are simply translated versions of one another. The result is a set of $2D$ Gaussian processes that are obviously dependent - the processes differ by a translation and a bit of independent noise.

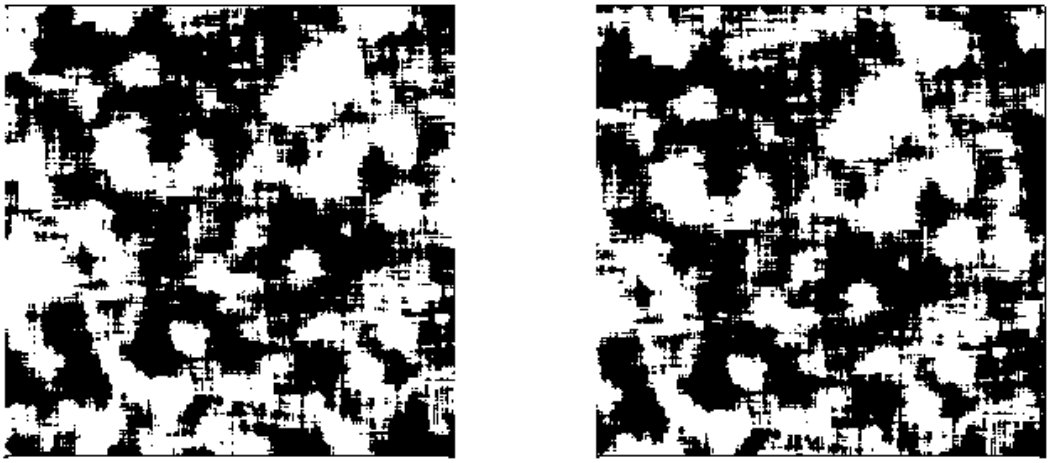

Figure 3.9: Dependent Gaussian processes generated by exciting a two-output $2D$ FIR filter with Gaussian white noise. The processes on the left and right were generated by FIR filters with the impulse responses differing only by a spatial translation. For display purposes, the output variables are thresholded, such that values less than zero are black, and greater than zero are white.

We can generalise to a D dimensional filter with the following impulse response

$$h(\mathbf{n}) = \sum_{i_1=0}^{M_1-1} \cdots \sum_{i_D=0}^{M_D-1} b_{i_1 \dots i_D} \prod_{j=1}^{D} \delta(n_j - i_j) \tag{3.50}$$

where $\mathbf{n} = [n_1 \dots n_D]^{\mathrm{T}} \in \mathbb{Z}^D$.

Unfortunately, the number of coefficients equals $\prod_{j=1}^{D} M_j$, which means directly parameterising a FIR filter is difficult in high dimensions. To alleviate such parameter growth with dimensionality, we could hyperparameterise the impulse responses via another function $f_\theta(i_1, \dots, i_D)$, which in turn defines the coefficients $b_{i_1 \dots i_D} = f_\theta(i_1, \dots, i_D)$, which can then be used in equation (3.50) to define the FIR impulse responses. This is equivalent to defining a continuous impulse response, and then sampling it over a regular lattice.

The auto and cross-covariance functions (3.39) - (3.42), for two $1D$ discrete time Gaussian processes generated by a FIR filter were derived in section 3.7.1 (page 46). The generalisation to the auto and cross-covariance functions for two Gaussian processes, produced by two D-dimensional FIR filters with impulse responses $h_1(\mathbf{n})$ and $h_2(\mathbf{n})$, is

$$c_{11}(\mathbf{m}) = \sum_{j_1=0}^{M-1} \cdots \sum_{j_D=0}^{M-1} h_1(\mathbf{j}) h_1(\mathbf{j}+\mathbf{m}) \tag{3.51}$$

$$c_{22}(\mathbf{m}) = \sum_{j_1=0}^{M-1} \cdots \sum_{j_D=0}^{M-1} h_2(\mathbf{j}) h_2(\mathbf{j}+\mathbf{m}) \tag{3.52}$$

$$c_{12}(\mathbf{m}) = \sum_{j_1=0}^{M-1} \cdots \sum_{j_D=0}^{M-1} h_1(\mathbf{j}) h_2(\mathbf{j}+\mathbf{m}) \tag{3.53}$$

$$c_{21}(\mathbf{m}) = \sum_{j_1=0}^{M-1} \cdots \sum_{j_D=0}^{M-1} h_2(\mathbf{j}) h_1(\mathbf{j}+\mathbf{m}) \tag{3.54}$$

where M is the order of the filters, $\mathbf{m} = [m_1, \ldots, m_D]^{\mathrm{T}} \in \mathbb{Z}^D$ is the input separation and $\mathbf{j} = [j_1, \ldots, j_D]^{\mathrm{T}}$ is the vector of summation indices.

For multidimensional IIR filters, we can find the covariance functions via the multidimensional z-transform (or the discrete space Fourier transform). However, for all but the simplest impulse responses (e.g. those that are isotropic and exponential), this becomes more challenging as the dimensionality increases. If we have discretely spaced data in high dimensions, it is simpler to model this with a FIR filter.

3.9 Multiple Output Low Pass Filters

In chapter 2, we derived the covariance function for a Gaussian process generated by stimulating a low pass filter with Gaussian white noise. In this section, we extend the low-pass filter framework by specifying ideal MIMO low-pass filters, with dependent Gaussian processes as outputs.

Firstly, consider a simple example with a single input noise source, and two output processes generated from two low pass filters with cutoff frequency $\frac{1}{2}$

Hz:

$$h_1(\mathbf{x}) = \text{sinc}(\pi\mathbf{x}) = \prod_{i=1}^{D} \text{sinc}(\pi x_i) \tag{3.55}$$

$$h_2(\mathbf{x}) = \text{sinc}(\pi(\mathbf{x} - \boldsymbol{\mu})) = \prod_{i=1}^{D} \text{sinc}(\pi(x_i - \mu_i)) \tag{3.56}$$

where filter h_2 translates its output by $\boldsymbol{\mu}$.

From appendix A.1, the auto/cross covariance functions for the outputs are:

$$\text{cov}_{ij}(\mathbf{d}) = \int_{\mathbb{R}^D} h_i(\mathbf{x}) h_j(\mathbf{x} + \mathbf{d}) d^D\mathbf{x} \tag{3.57}$$

from which we can find the two auto-covariance and two cross-covariance functions:

$$\begin{aligned}
\text{cov}_{11}(\mathbf{d}) &= \prod_{i=1}^{D} \int_{-\infty}^{\infty} \text{sinc}(\pi x_i)\, \text{sinc}(\pi(x_i + d_i)) dx_i \\
&= \prod_{i=1}^{D} \text{sinc}(\pi d_i) \qquad (3.58) \\
\text{cov}_{22}(\mathbf{d}) &= \prod_{i=1}^{D} \int_{-\infty}^{\infty} \text{sinc}(\pi(x_i - \mu_i))\, \text{sinc}(\pi(x_i + d_i - \mu_i)) dx_i \\
&= \prod_{i=1}^{D} \int_{-\infty}^{\infty} \text{sinc}(\pi q_i)\, \text{sinc}(\pi(q_i + d_i)) dq_i \\
&= \text{cov}_{11}(\mathbf{d}) \qquad (3.59) \\
\text{cov}_{12}(\mathbf{d}) &= \prod_{i=1}^{D} \int_{-\infty}^{\infty} \text{sinc}(\pi x_i)\, \text{sinc}(\pi(x_i + d_i - \mu_i)) dx_i \\
&= \prod_{i=1}^{D} \text{sinc}(\pi(d_i - \mu_i)) \qquad (3.60) \\
\text{cov}_{21}(\mathbf{d}) &= \prod_{i=1}^{D} \int_{-\infty}^{\infty} \text{sinc}(\pi(x_i - \mu_i))\, \text{sinc}(\pi(x_i + d_i)) dx_i \\
&= \prod_{i=1}^{D} \int_{-\infty}^{\infty} \text{sinc}(\pi q_i)\, \text{sinc}(\pi(q_i + d_i + \mu_i)) dq_i \\
&= \prod_{i=1}^{D} \text{sinc}(\pi(d_i + \mu_i)) = \prod_{i=1}^{D} \text{sinc}(\pi(-d_i - \mu_i)) \\
&= \text{cov}_{12}(-\mathbf{d}) \qquad (3.61)
\end{aligned}$$

Overall, output 1 is correlated to a translated version of output 2, where the translation (or lag) is given by $\boldsymbol{\mu}$. In fact, because both outputs are derived from the same noise source, and there are no other latent sources, outputs 2 is exactly equal to output 1 translated by $\boldsymbol{\mu}$.

The 1-input 2-output filter above can easily be generalised to $M \times N$ sinc impulse responses, each with a cutoff frequency $\frac{1}{2}$ Hz, but potentially different translations. A more interesting generalisation is to consider an anisotropic low-pass filter, with cutoff frequencies that may differ for each direction in space. Such an anisotropic filter with low cutoff frequency in direction i and high cutoff frequency in direction j will produce outputs that vary slowly in direction i, but rapidly in direction j. The cutoff frequency is thus analogous to the length scale in the squared exponential covariance function. To make this generalisation, we need to make use of some properties of the multidimensional Fourier transform (see [51]).

Informally, the multidimensional Fourier transform maps a function over space to a function over frequency, $\mathcal{F}[g(\mathbf{x})] = G(\mathbf{f})$, where $\mathbf{f}$ is a spatial frequency vector. For example, the Fourier transform of an LTI impulse response gives us the frequency response for that filter. We can use the properties of the Fourier transform to find the covariance function for the outputs of a MIMO filter.

Firstly, consider the following property of the Fourier transform

$$\mathcal{F}\left[\int_{\mathbb{R}^D} h_i(\mathbf{s}) h_j(\mathbf{s}+\boldsymbol{\tau}) d^D\mathbf{s}\right] = H_i^*(\mathbf{f}) H_j(\mathbf{f}) \tag{3.62}$$

where $*$ denotes the complex conjugate.

If the impulse responses $h_i(\mathbf{x})$ and $h_j(\mathbf{x})$ are such that the right hand side of equation (3.62) can be reduced to a simpler form, then it may be possible to find the inverse Fourier transform $\mathcal{F}^{-1}[\cdot]$ without having to perform the convolution integral. Consider a simple example over time t, with a single input noise source, and two outputs produced by filters with sinc impulse responses, $h_1(t) = \operatorname{sinc}(2\pi f_1 t)$ and $h_2(t) = \operatorname{sinc}(2\pi f_2(t-\mu))$, where f_1 and f_2 are the cutoff frequencies. The Fourier transforms are

$$H_1(f) = \frac{1}{2f_1} \operatorname{rect}\left(\frac{f}{2f_1}\right) = H_1^*(f) \tag{3.63}$$

$$H_2(f) = \frac{1}{2f_2} \operatorname{rect}\left(\frac{f}{2f_2}\right) \exp(-j2\pi f\mu) \tag{3.64}$$

where we have made use of the fact that $\mathcal{F}[g(t-\mu)] = \exp(-j2\pi f\mu)\mathcal{F}[g(t)]$.

Let $f_{min} = \min(f_1, f_2)$ and let $j = \sqrt{-1}$. The cross covariance functions can now be found from an inverse Fourier transform.

$$\begin{aligned}
\mathrm{cov}_{12}(d) &= \mathcal{F}^{-1}\left[H_1^*(f)H_2(f)\right] \\
&= \mathcal{F}^{-1}\left[\frac{1}{4f_1f_2}\,\mathrm{rect}\left(\frac{f}{2f_1}\right)\mathrm{rect}\left(\frac{f}{2f_2}\right)\exp(-j2\pi f\mu)\right] \\
&= \frac{1}{4f_1f_2}\mathcal{F}^{-1}\left[\mathrm{rect}\left(\frac{f}{2f_{min}}\right)\exp(-j2\pi f\mu)\right] \\
&= \frac{f_{min}}{2f_1f_2}\,\mathrm{sinc}(2\pi f_{min}(d-\mu)) \qquad (3.65) \\
\mathrm{cov}_{21}(d) &= \mathcal{F}^{-1}\left[H_1(f)H_2^*(f)\right] \\
&= \frac{f_{min}}{2f_1f_2}\,\mathrm{sinc}(2\pi f_{min}(d+\mu)) \qquad (3.66)
\end{aligned}$$

and the auto-covariance functions are

$$\mathrm{cov}_{11}(d) = \frac{1}{2f_1}\,\mathrm{sinc}(2\pi f_1 d) \qquad \mathrm{cov}_{22}(d) = \frac{1}{2f_2}\,\mathrm{sinc}(2\pi f_2 d) \qquad (3.67)$$

The generalisation to multidimensional anisotropic low pass filters, each with separate lags, requires the following property of the multidimensional Fourier transform [51]:

$$\mathcal{F}\left[h(\mathbf{As}-\boldsymbol{\mu})\right] = \frac{1}{|\det(\mathbf{A})|}\mathcal{F}\left[h(\mathbf{A}^{-\mathrm{T}}\mathbf{f})\right]\exp\left(-j2\pi\boldsymbol{\mu}\mathbf{A}^{-\mathrm{T}}\mathbf{f}\right) \qquad (3.68)$$

Here, we consider only axis aligned sinc impulse responses by setting $\mathbf{A} = \mathrm{diag}\left(\left[a_1^{-2}\ldots a_D^{-2}\right]^{\mathrm{T}}\right)$, and $h(\mathbf{x}) = \prod_{i=1}^{D}\mathrm{sinc}(\pi x_i)$. A small value of a_i will result in high frequency cutoff in direction i, meaning the filter output will vary rapidly in direction i.

So, consider two impulse responses $h_1(\mathbf{x}) = h(\mathbf{Ax}-\mathbf{u})$, and $h_2(\mathbf{x}) = h(\mathbf{Bx}-\mathbf{v})$, where $\mathbf{B} = \mathrm{diag}\left(\left[b_1^{-2}\ldots b_D^{-2}\right]^{\mathrm{T}}\right)$, and $\mathbf{u}, \mathbf{v}$ are translations. The Fourier transform of $h(\mathbf{x})$ is $H(\mathbf{f}) = \prod_{i=1}^{D}\mathrm{rect}(f_i)$. Given equation 3.68, the frequency responses for the filters are:

$$H_1(\mathbf{f}) = \frac{1}{\det(\mathbf{A})}H(\mathbf{A}^{-1}\mathbf{f})\exp\left(-j2\pi\mathbf{u}\mathbf{A}^{-1}\mathbf{f}\right) \qquad (3.69)$$

$$H_2(\mathbf{f}) = \frac{1}{\det(\mathbf{B})}H(\mathbf{B}^{-1}\mathbf{f})\exp\left(-j2\pi\mathbf{v}\mathbf{B}^{-1}\mathbf{f}\right) \qquad (3.70)$$

We wish to derive the auto and cross-covariance functions for the two outputs generated by exciting filters h_1 and h_2 with Gaussian white noise. We proceed by taking the Fourier transform of the covariance function:

$$\mathcal{F}[c_{ij}(\mathbf{d})] = \mathcal{F}\left[\int_{\mathbb{R}^D} h_i(\mathbf{x})h_j(\mathbf{x}+\mathbf{d})d\mathbf{x}\right] \tag{3.71}$$

$$= H_i^*(\mathbf{f})H_j(\mathbf{f}) \tag{3.72}$$

Initially, we examine the case when $i = j = 1$.

$$\begin{aligned}
\mathrm{cov}_{11}(\mathbf{d}) &= \mathcal{F}^{-1}\left[H_1(\mathbf{f})H_1^*(\mathbf{f})\right] \\
&= \mathcal{F}^{-1}\left[\frac{1}{\det(\mathbf{A})^2}H(\mathbf{A}^{-1}\mathbf{f})^2\right] \\
&= \frac{1}{\det(\mathbf{A})^2}\mathcal{F}^{-1}\left[\prod_{i=1}^{D}\mathrm{rect}(a_i^2 f_p)\right] \\
&= \frac{1}{\det(\mathbf{A})^2}\prod_{i=1}^{D} a_i^{-2}\,\mathrm{sinc}\left(\pi a_i^{-2} d_p\right) &(3.73)\\
&= \frac{1}{\det(\mathbf{A})}\,\mathrm{sinc}(\pi\mathbf{A}\mathbf{d}) &(3.74)
\end{aligned}$$

Similarly,

$$\mathrm{cov}_{22}(\mathbf{d}) = \frac{1}{\det(\mathbf{B})}\,\mathrm{sinc}(\pi\mathbf{B}\mathbf{d}) \tag{3.75}$$

So the auto-covariance functions are not dependent on the translation vectors.

The cross-covariance function requires us to define a matrix $\mathbf{Q} = \mathrm{diag}\left([q_1^{-2}\ldots q_D^{-2}]^{\mathrm{T}}\right)$, where $q_i^2 = \max(a_i^2, b_i^2)$, and a vector $\boldsymbol{\mu} = \mathbf{A}^{-1}\mathbf{u} + \mathbf{B}^{-1}\mathbf{v}$. Then

$$\begin{aligned}
\mathrm{cov}_{12}(\mathbf{d}) &= \mathcal{F}^{-1}\left[H_1^*(\mathbf{f})H_2(\mathbf{f})\right] \\
&= \mathcal{F}^{-1}\left[\frac{1}{\det(\mathbf{A}\mathbf{B})}H(\mathbf{A}^{-1}\mathbf{f})H(\mathbf{B}^{-1}\mathbf{f})\exp\left(-j2\pi(\mathbf{u}\mathbf{A}^{-1}+\mathbf{v}\mathbf{B}^{-1})\mathbf{f}\right)\right] \\
&= \frac{1}{\det(\mathbf{A}\mathbf{B})}\mathcal{F}^{-1}\left[\prod_{i=1}^{D}\mathrm{rect}(q_i^2 f_i)\exp(-j2\pi\mu_i f_i)\right] \\
&= \frac{1}{\det(\mathbf{A}\mathbf{B})}\prod_{i=1}^{D} q_i^{-2}\,\mathrm{sinc}(\pi q_i^{-2} d_i) \\
&= \frac{\det(\mathbf{Q})}{\det(\mathbf{A}\mathbf{B})}\,\mathrm{sinc}(\pi\mathbf{Q}(\mathbf{d}-\boldsymbol{\mu})) &(3.76)\\
&= \mathrm{cov}_{21}(-\mathbf{d})
\end{aligned}$$

Therefore, we can build a set of N dependent Gaussian processes over $\mathbb{R}^D$, where each process and its dependencies is characterised by sinc covariance functions. To do so, we made use of the (multidimensional) Fourier transform and the fact that the Fourier transform of a sinc function is a rect function. Furthermore, instead of performing a convolution integral to find the covariance functions, we made use of the fact that convolution in the spatial domain equates to multiplication in the frequency domain. That is, we found the covariance function by multiplying rect functions and then taking the Fourier inverse.

3.10 Multiple Output Gaussian Filters

The previous section showed how we could build dependent Gaussian processes models by defining a set of impulse responses, Fourier transforming those responses, and then finding covariance functions by an inverse Fourier transformation. This approach is limited to those impulse responses that have a Fourier transform of such a form that we can find the inverse Fourier transform of equation 3.62.

The Fourier transform of a Gaussian is itself a Gaussian. Furthermore, the product of Gaussians is also a Gaussian. Therefore, we can use Fourier transforms to derive dependent Gaussian processes with squared-exponential covariance functions. We do not make this derivation here as a derivation that achieves the same result by performing a convolution integral is given in appendix A.2.

3.11 Summary

We have shown how the Gaussian Process framework can be extended to inference concerning multiple output variables without assuming them to be independent. Multiple processes can be handled by inferring convolution kernels instead of covariance functions. This makes it easy to construct the required positive definite covariance matrices for co-varying outputs.

One application of this work is to learn the spatial translations between outputs. However the framework developed here is more general than this, as it can model data that arises from multiple sources, only some of which are shared. Our examples show the "in-filling" of sparsely sampled regions due to a learnt dependency between outputs. Another application is the forecasting of dependent time series. Our example shows how learning couplings between multiple time series may aid in forecasting, particularly when the series to be forecast is dependent on previous or current values of other series.

The methodology presented here also applies to digital filters, allowing the generation of discrete time and space dependent Gaussian processes.

Dependent Gaussian processes should be particularly valuable in cases where one output is expensive to sample, but co-varies strongly with a second that is cheap. By inferring both the coupling and the independent aspects of the data, the cheap observations can be used as a proxy for the expensive ones.

Chapter 4

Gaussian Processes for System Identification

In the previous chapters we have seen how to extend Gaussian processes for regression over a single output to the multiple output case. To do so, we interpreted Gaussian processes as the outputs of analog and digital filters. In this chapter, we show how to use this methodology for the purposes of system identification. That is, by observing the inputs and outputs of a system over time, we infer a model that suitably explains the observations. Here, it is assumed that the system under observation can be modelled by a linear time invariant system (section 2.1 (page 17)), an assumption that may be poor for many systems of interest, such as non-linear systems or time-variant systems.

4.1 System Identification

Consider a dynamical system for which we observe the system input and output over time, and wish to build a model of the system given those observations. The input/output data might be discrete or continuous in nature. In the continuous time case, we wish to identify an LTI system that explains the input/output relationship. In the discrete time case we can either treat the data as regularly sampled continuous data and again build an LTI model, or we can build a digital filter model that best fits the data. The models will be inferred by using maximum likelihood or maximum a posteriori procedures.

The system identification task (see [31, 30]) differs from the regression task because of dynamics. A regression task has inputs that map to outputs in a static fashion, such that it is assumed that there is an underlying unchanging mapping to be identified. In system identification, the output at a given time is assumed to be dependent not only on the input at that time, but also on previous inputs and outputs. The system to be identified thus maps input trajectories to output trajectories.

One way to approach system identification is to build a set of dependent Gaussian processes that model the dependencies across the inputs, across the outputs, and between the inputs and outputs. In what follows, we examine the discrete time and continuous time cases separately.

4.1.1 FIR identification

The output of a FIR filter is defined by equation (2.22) (page 24). For N time steps, this can be rewritten [70] using matrix notation as follows:

$$\mathbf{y} = \mathbf{X}\mathbf{h} \tag{4.1}$$

where $\mathbf{y} = [y(0) \dots y(N-1)]^{\mathrm{T}}$ is the vector of filter outputs over N time steps, $\mathbf{h} = [b_0 \dots b_{M-1}]^{\mathrm{T}}$ is the vector of M filter coefficients, and the $N \times M$ matrix $\mathbf{X}$ is formed from the inputs vector $\mathbf{x} = [x(0) \dots x(N-1)]^{\mathrm{T}}$

$$\mathbf{X} = \begin{bmatrix} x(0) & 0 & \dots & 0 \\ x(1) & x(0) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x(M-1) & x(M-2) & \dots & x(0) \\ x(M) & x(M-1) & \dots & x(1) \\ \vdots & \vdots & \dots & \vdots \\ x(N-1) & x(N-2) & \dots & x(N-M) \end{bmatrix} \tag{4.2}$$

The system identification task here is to identify the M filter coefficients in $\mathbf{h}$, given the input-output observations over N time steps.

In general, $\mathbf{X}$ is not square so has no inverse. However, one can use the Moore-Penrose pseudoinverse [85]

$$\mathbf{X}^{+} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}} \tag{4.3}$$

to recover $\mathbf{h}$ from $\mathbf{y}$

$$\mathbf{h} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \mathbf{X}^{+}\mathbf{y} \tag{4.4}$$

which is the least-squares solution[1] [31, 70] and has computational complexity $\mathcal{O}(NM^2)$.

Here, we generalise this to the MIMO situation. For example, consider a 2-input 2-output system, which we wish to model with 2×2 impulse responses $h_{11}, h_{12}, h_{21}, h_{22}$, each consisting of M coefficients. We have two $N \times 1$ input vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, and two $N \times 1$ output vectors, $\mathbf{y}_1$ and $\mathbf{y}_2$. From $\mathbf{x}_1$ and $\mathbf{x}_2$, we construct the $N \times M$ matrices $\mathbf{X}_1$ and $\mathbf{X}_2$, just as we constructed $\mathbf{X}$ from $\mathbf{x}$ in equation (4.2). The generative model is then

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix} \begin{bmatrix} \mathbf{h}_{11} \\ \mathbf{h}_{21} \\ \mathbf{h}_{12} \\ \mathbf{h}_{22} \end{bmatrix} \tag{4.5}$$

which is separable for each output

$$\mathbf{y}_i = \begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix} \begin{bmatrix} \mathbf{h}_{1i} \\ \mathbf{h}_{2i} \end{bmatrix} \tag{4.6}$$

and has least-squares solution

$$\begin{bmatrix} \mathbf{h}_{1i} \\ \mathbf{h}_{2i} \end{bmatrix} = \begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix}^{+} \mathbf{y}_i \tag{4.7}$$

The least-squares FIR system identification is computationally efficient (relative to the dependent Gaussian processes solution to follow), although for real-time applications $\mathcal{O}(M^2N)$ may be considered too expensive and one may prefer to use other methods such as recursive least squares [25]. Below, we show how discrete time dependent Gaussian processes can be used to solve for the system.

Consider a two-input two-output (2I2O) discrete time system, observed for N time steps as shown if figure 4.1. Call the observed input sequences $x_1(n)$

[1] the least-squares solution is that which minimises the sum of squared model errors across the output observations $\mathbf{y}$

and $x_2(n)$ and the output sequences $y_1(n)$ and $y_2(n)$. We assume a generative model where $x_1(n)$ and $x_2(n)$ are dependent Gaussian processes generated by exciting a 2I2O FIR filter with Gaussian white noise (the latent sources). We further assume that $y_1(n)$ and $y_2(n)$ are generated by filtering $x_1(n)$ and $x_2(n)$ by a second 2I2O FIR filter, which we wish to identify.

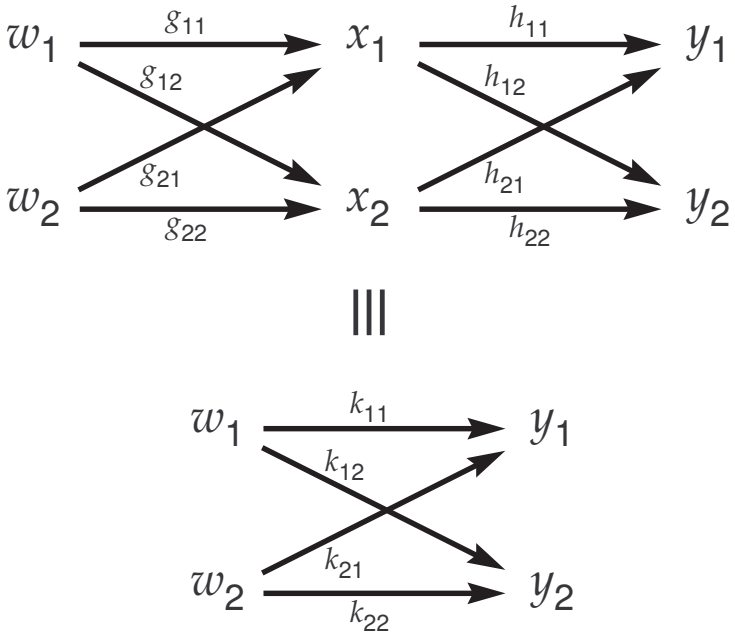

Figure 4.1: Two-input two-output discrete time filter, excited by noise sources w_1 and w_2 to produce outputs y_1 and y_2. At the top, the filter is shown as a cascade of two sub-filters with impulse responses g and h. The signals in between the two sub-filters are the observed 'input' sequences x_1 and x_2. The cascade can be simplified to a single filter with impulse responses k, shown at the bottom.

The latent noise sequences $w_1(n)$ and $w_2(n)$ excite a FIR filter characterised by impulse responses $g_{11}(n), g_{12}(n), g_{21}(n), g_{22}(n)$, to produce the sequences $x_1(n)$ and $x_2(n)$ so that

$$x_j(n) = \sum_{i=1}^{2} w_i(n) * g_{ij}(n)$$

The outputs are constructed by filtering $x_1(n)$ and $x_2(n)$, with impulse re-

sponses $h_{11}(n), h_{12}(n), h_{21}(n), h_{22}(n)$.

$$y_j(n) = \sum_{m=1}^{2} x_m(n) * h_{mj}(n) \tag{4.9}$$

$$= \sum_{m=1}^{2} \sum_{i=1}^{2} w_i(n) * g_{im}(n) * h_{mj}(n) \tag{4.10}$$

$$= \sum_{i=1}^{2} w_i(n) * k_{ij}(n) \tag{4.11}$$

where the FIR filter characterised by impulse responses $k_{11}(n)$, $k_{12}(n)$, $k_{21}(n)$, and $k_{22}(n)$ is formed by cascading the two filters with impulse responses g_{ij} and h_{ij}.

$$k_{ij}(n) = \sum_{m=1}^{2} g_{im}(n) * h_{mj}(n) \tag{4.12}$$

Given this generative model, and the analysis in section 3.7.1 (page 46), we can now find the auto/cross covariance functions for the sequences $x_1(n)$, $x_2(n)$, $y_1(n)$ and $y_2(n)$:

$$\mathrm{cov}_{ij}^{xx}(m) = E\left\{x_i(n+m)x_j(n)\right\} \tag{4.13}$$

$$\mathrm{cov}_{ij}^{yy}(m) = E\left\{y_i(n+m)y_j(n)\right\} \tag{4.14}$$

$$\mathrm{cov}_{ij}^{xy}(m) = E\left\{x_i(n+m)y_j(n)\right\} \tag{4.15}$$

$$\mathrm{cov}_{ij}^{yx}(m) = E\left\{y_i(n+m)x_j(n)\right\} \tag{4.16}$$

With these, and observations

$$\mathbf{x}_1 = [x_1(0) \dots x_1(N-1)]^{\mathrm{T}} \qquad \mathbf{y}_1 = [y_1(0) \dots y_1(N-1)]^{\mathrm{T}}$$
$$\mathbf{x}_2 = [x_2(0) \dots x_2(N-1)]^{\mathrm{T}} \qquad \mathbf{y}_2 = [y_2(0) \dots y_2(N-1)]^{\mathrm{T}}$$

we can go on to build covariance matrices and find the maximum likelihood or maximum *a posteriori* impulse responses g_{ij} and h_{ij}. Of course, it is the set h_{ij} that models, or *identifies* the system in question.

The computational complexity of building the M output dependent Gaussian processes system model for N time step observations is $\mathcal{O}(M^3N^3)$, or $\mathcal{O}(M^3N^2)$ if there is no missing data which makes the covariance matrices block-Toeplitz (see section 3.7.3). In any case, the dependent Gaussian processes solution is computationally more expensive than the least-squares method. Furthermore, the dependent Gaussian processes solution assumes the input

and output sequences are stationary Gaussian processes, which may not be a valid assumption in some situations. Given the computational expense and potential limitations, what is the advantage to building the dependent Gaussian processes model? In doing so we can find the predictive *distribution* for *any* time step, rather than just a point estimate. That is, we can find the predictive mean and variance for any input or output at any point in time, past or future. This is obviously of value if we wish to make predictions as well as identifying the underlying system.

4.1.2 Analog Filter Identification

The FIR system identification method above is useful if the system to be modelled is linear with a finite memory and hence looks like a FIR filter. The FIR method also requires data to be discretely spaced with different observations separated in time by some integer multiple of the sampling period.

In many cases, this FIR methodology might be insufficient. For example, if the underlying system has an infinite memory then modelling with a FIR filter effectively truncates that memory. A simple example of an infinite memory system is an IIR filter. If the IIR filter's impulse response decays rapidly, then FIR modelling will suffice. On the other hand, for example, if the impulse response is $h(t) = sin(t)$, then a FIR model might be quite poor. Another situation that is poorly modelled by a FIR filter is time series data that is irregularly sampled over time. To build a FIR model, the sampling rate must be increased until the observation points approximately coincide with sample points. We then interpret the data as being regularly sampled with a high sampling rate and lots of missing data points. However, increasing the sampling rate increases the number of FIR coefficients required to define impulse responses of a certain duration. For example, if we were to double the sample rate, we would need to double the number of coefficients to maintain the impulse response duration. In such cases, it is simpler to use an analog model.

For analog filter identification, it would be nice to use Gaussian filters, with Gaussian impulse response. The beauty of this is that a cascade of Gaussian filters is still a Gaussian filter[1]. For system identification this would mean

[1] A Gaussian convolved with a Gaussian is another Gaussian. Furthermore, a Gaussian filter has a Gaussian frequency response, and the multiplication of two Gaussians is another

that our generative model consists of white noise sources driving Gaussian filters to produce the inputs which drive more Gaussian filters to produce the outputs. Finally, the dependent Gaussian model would have Gaussian auto and cross covariance functions, simplifying the analysis. Unfortunately, Gaussian filters are acausal, and therefore unrealisable. It seems unreasonable to identify a system as unrealisable.

Consider a single-input single-output (SISO) system where the input $x(t)$ and output $y(t)$ functions are related by an ordinary linear differential equation with constant coefficients [2]:

$$\sum_{m=0}^{M} a_m \frac{d^m x(t)}{dt^m} = \sum_{n=0}^{N} b_n \frac{d^n y(t)}{dt^n} \tag{4.17}$$

If we take the Laplace transform, $\mathcal{L}[\cdot]$, of both sides and rearrange we find the system transfer function $H(s)$ as a real rational function of complex frequency s,

$$H(s) = \frac{Y(s)}{X(s)} = \frac{\sum_{m=0}^{M} a_m s^m}{\sum_{n=0}^{N} b_n s^n} \tag{4.18}$$

where $X(s) = \mathcal{L}[x(t)]$ and $Y(s) = \mathcal{L}[y(t)]$. The impulse response is in general infinite in duration and is found from the inverse Laplace transform $h(t) = \mathcal{L}^{-1}[H(s)]$. For $H(s)$ to be BIBO stable, the roots of the denominator polynomial $\sum_{n=0}^{N} b_n s^n$ must have negative real parts [16]. That is, the poles of the system transfer function must lie in the left half of the s-plane. We can enforce this by choosing a parameterisation that directly encodes the position of each pole. That is, we factorise the denominator and parameterise it as $\prod_{n=0}^{N}(s - \beta_n)$, where the parameters $\beta_n \in \mathbb{C}$ are the system poles with $\Re(\beta_n) \leq 0$ (in the left half s-plane).

The transfer function $H(s)$ is the ratio of two polynomials with real, constant coefficients. If we cascade two such systems $G(s)$ and $H(s)$ then the combined transfer function is equal to the product $K(s) = G(s)H(s)$. This combined transfer function is also the ratio of two polynomials in s, and therefore maps the input to output via an ordinary differential equation with constant coefficients.

We can now specify a generative model by starting with a Gaussian white noise input $w(t)$ filtered by G to produce the input function $x(t)$, which is

Gaussian.

then filtered by H to form the output function $y(t)$. Overall, $y(t)$ is equivalent to the output of filter K when excited by $w(t)$. Given observations of $x(t)$ and $y(t)$, we can build a dependent Gaussian process model which has parameters equal to the coefficients in the differential equations that describe filters G and H. Of course, when we infer the parameters for H we are effectively building a model of the system we are interested in.

The covariance functions for the dependent Gaussian processes model are most easily constructed in the frequency domain. For example, for an input separation of τ

$$\mathrm{cov}^{xy}(\tau) = \mathcal{F}^{-1}\left[G^*(j\omega)K(j\omega)\right](\tau) \tag{4.19}$$

where $\omega = 2\pi f$ and G^* is the complex conjugate of G. In words, the cross covariance function between $x(t)$ and $y(t+\tau)$ is found from the inverse Fourier transform of the product of frequency response functions of the filters G^* and K. We find $G(j\omega)$ from the Laplace transform $G(s)$ by letting $s \to j\omega$. Finding $\mathcal{F}^{-1}[H(j\omega)]$ is most easily accomplished by computing the partial fraction expansion of $H(j\omega)$, and then referring to a table of Fourier transform pairs.

As a simple example, consider a SISO system where the filters G and H are given by

$$G(s) = \frac{1}{s+a} \qquad\qquad H(s) = \frac{1}{s+b} \tag{4.20}$$

with $a, b \geq 0$ to ensure stability. The cascade filter K is equal to the product

$$K(s) = G(s)H(s) = \frac{1}{(s+a)(s+b)} \tag{4.21}$$

To build the dependent Gaussian processes model we require covariance functions $\mathrm{cov}(x(t+\tau), x(t))$, $\mathrm{cov}(x(t+\tau), y(t))$, $\mathrm{cov}(y(t+\tau), x(t))$ and $\mathrm{cov}(y(t+\tau)y(t))$. These are as found via the inverse Fourier transform as described above, giving

$$\mathrm{cov}^{xx}(\tau) = \frac{1}{2a}\exp(-a|\tau|) \tag{4.22}$$

$$\mathrm{cov}^{xy}(\tau) = \frac{1}{2a(a+b)(a-b)} \begin{cases} 2a\exp(-b\tau) - (a+b)\exp(-a\tau) & \tau \geq 0 \\ (a-b)\exp(a\tau) & \tau < 0 \end{cases} \tag{4.23}$$

$$\mathrm{cov}^{yx}(\tau) = \frac{1}{2a(a+b)(a-b)} \begin{cases} (a-b)\exp(-a\tau) & \tau \geq 0 \\ 2a\exp(b\tau) - (a+b)\exp(a\tau) & \tau < 0 \end{cases} \tag{4.24}$$

$$\mathrm{cov}^{yy}(\tau) = \frac{1}{2ab(a+b)(a-b)}\left(a\exp(-b|\tau|) - b\exp(-a|\tau|)\right) \tag{4.25}$$

The covariance functions are continuous, but not differentiable at $\tau = 0$ because the underlying filters have impulse responses with a discontinuity at $t = 0$. For example,

$$h(t) = \begin{cases} \exp(-at) & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{4.26}$$

This framework can be used to specify more complicated models, with higher order filters and multiple inputs and outputs. However, doing so results in increasingly complicated covariance functions and increasing numbers of parameters. For very complicated models one would want to automate the derivation of covariance functions given the filter transfer functions.

4.1.3 IIR Identification

If we wish to model a system with a digital IIR filter, then we can use a similar process to that of modelling with an analog filter. The difference is that we use the z-transform in place of the Laplace transform, and the discrete-time Fourier transform in place of the Fourier transform. In other words, we parameterise two cascaded digital filters with impulse responses $g(n)$ and $h(n)$. Instead of computing the Laplace transform, we use the z-transform to find $G(z)$ and $H(z)$. The z-transform of the combined cascade is $K(z) = G(z)H(z)$. We map $z \to e^{j\omega T}$, where T is the sampling period to find the Discrete Time Fourier Transform (DTFT). The covariance functions can then be calculated using the inverse DTFT.

To ensure the filters are stable, we choose a parameterisation that forces the system poles to lie within the unit circle on the z-plane. For example, if the transfer function of a filter is $H(z) = \frac{N(z)}{D(z)}$, then we parameterise $D(z) = \prod_{i=1}^{N}(z - c_i)$ such that the roots $|c_i| \leq 1$.

4.2 Summary

This chapter, has shown how we can identify multiple output systems using dependent Gaussian processes. To do so, we made use of the direct relationship between dependent Gaussian processes and multiple output filters.

Chapter 5

Reduced Rank Gaussian Processes

Using Gaussian processes for regression requires the inversion of an $n \times n$ covariance matrix, where n is the number of observations. Exact computation of this matrix is an $O(n^3)$ operation, meaning that Gaussian process regression becomes infeasible for large data sets [17]. In this chapter, we examine one method for approximate Gaussian process regression that can significantly reduce the computational cost compared to the exact implementation. This method uses *Reduced Rank Gaussian Processes* and was first introduced by Quiñonero-Candela and Rasmussen [55]. First, we review this method and then describe an extension allowing the construction of *non-stationary* reduced rank Gaussian processes.

5.1 Generalised Linear Models

To begin the analysis, firstly consider a generalised linear model, with m basis functions, $\phi_1(\cdot) \dots \phi_m(\cdot)$, used to model n observations, consisting of input-output pairs, $\mathcal{D} = \{(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n)\}$. Using the basis functions and data, we can construct an $m \times n$ design matrix $\mathbf{\Phi} = [\boldsymbol{\phi}(\mathbf{x}_1) \,|\, \dots \,|\, \boldsymbol{\phi}(\mathbf{x}_n)]$, where $\boldsymbol{\phi}(\mathbf{x}_i) = [\phi_1(\mathbf{x}_i) \dots \phi_m(\mathbf{x}_i)]^{\mathrm{T}}$.

The generalised linear model is $\mathbf{f} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{w}$ and the generative model is:

$$\mathbf{y} = \mathbf{f} + \boldsymbol{\epsilon} \tag{5.1}$$

$$= \mathbf{\Phi}^{\mathrm{T}}\mathbf{w} + \boldsymbol{\epsilon} \tag{5.2}$$

where $\mathbf{y} = [y_1 \dots y_n]^{\mathrm{T}}$ is the vector of targets, $\mathbf{w}$ is an m vector of weights, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ is a Gaussian noise vector. If we place a Gaussian prior distribution over the weights, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_p)$, then the Gaussian predictive distribution at a test point $\mathbf{x}_*$ can be found analytically [61] and is[1]

$$y_*|\mathbf{x}_*, X, \mathbf{y} \;\sim\; \mathcal{N}\left(\frac{1}{\sigma^2}\boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}}\mathbf{A}^{-1}\boldsymbol{\Phi}\mathbf{y},\;\; \sigma^2 + \boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}}\mathbf{A}^{-1}\boldsymbol{\phi}(\mathbf{x}_*)\right) \tag{5.3}$$

where $\mathbf{A} = \sigma^{-2}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \boldsymbol{\Sigma}_p^{-1}$.

Given the prior distribution over weights is Gaussian with a covariance matrix $\boldsymbol{\Sigma}_p$, the induced prior distribution over function values is $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Sigma}_p\boldsymbol{\Phi})$. This is a Gaussian process prior distribution, meaning the generalised linear model specified here is a Gaussian process model. However, as pointed out by Quiñonero-Candela and Rasmussen [55] this Gaussian process is *degenerate*. Degenerate Gaussian process models have the undesirable feature of a decrease in predictive variance as the test point moves away from the training data. For example, if we use Gaussian basis functions in equation (5.3), then as the test point moves away from the basis functions ($x_* \to \infty$), we find that $\boldsymbol{\phi}(\mathbf{x}_*) \to \mathbf{0}$ and the predictive variance approaches the prior noise variance ($v(\mathbf{x}_*) \to \sigma^2$). That is, the predictive variance decreases to a minimum at $\mathbf{x}_* = \infty$ regardless of the training data, which seems a strange prior belief to hold.

Now consider the generalised linear model built from $m = n$ basis functions, with $\phi_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i)$ where $k(\cdot, \cdot)$ is a positive definite kernel. This equates to a generalised linear model with a basis function centred on each training input. We set the prior distribution over weights to $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}^{-1})$, where $\mathbf{K}$ is the $n \times n$ *Gram matrix* with entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The predictive distribution becomes:

$$y_*|\mathbf{x}_*, X, \mathbf{y} \;\sim\; \mathcal{N}\Big(\mathbf{k}_*^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y},\;\; \sigma^2 + \sigma^2\mathbf{k}_*^{\mathrm{T}}\mathbf{K}^{-1}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_*\Big) \tag{5.4}$$

where $\mathbf{k}_*^{\mathrm{T}} = [k(\mathbf{x}_*, \mathbf{x}_1) \dots k(\mathbf{x}_*, \mathbf{x}_n)]$ is constructed by evaluating each basis function at the test point.

We see that mean predictor is now equal to that of a full Gaussian process, [69] with kernel $k(\cdot, \cdot)$. Furthermore, since $n = m$ we find that $\boldsymbol{\Phi} = \mathbf{K}$, resulting

[1] Note the dual meaning of the symbol $|$. In some cases it is used to concatenate elements in a vector or matrix. In others, it means "conditioned on". Disambiguation is usually possible given the context.

in a prior distribution over functions $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}\mathbf{K}^{-1}\mathbf{K}) = \mathcal{N}(\mathbf{0}, \mathbf{K})$, as is the case with a full GP model. However, note that the predictive variance differs from that of a full GP model. Even with this new specification, the model remains a generalised linear model, and the resulting GP model is degenerate. A symptom of this degeneracy is that as the test point $\mathbf{x}_*$ moves away from the training data, the predictive variance approaches σ^2.

In summary, by centring a basis function $k(\mathbf{x}, \mathbf{x}_i)$ on each training input and setting a prior distribution over weights, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}^{-1})$, we can construct a Gaussian process with a predictive mean equal to that of a Gaussian process with kernel $k(\cdot, \cdot)$, as in equation (1.12) (page 7). However, this new Gaussian process is degenerate.

5.2 Subset of Regressors

The previous section showed how we can find the mean GP predictor by building a generalised linear model with n basis functions and an appropriate prior distribution. Predicting from generalised linear models with m basis functions (equation (5.3)), has complexity $\mathcal{O}(m^3)$, due the fact that we need to invert the $m \times m$ matrix $\mathbf{A}$. If we use n basis functions centred on the training inputs, we recover the full GP mean predictor. The complexity of doing this is $\mathcal{O}(n^3)$ - the same as for the full GP.

As stated above, for large n, this cost is unacceptable. To reduce the computational expense, we may consider using a reduced set of m basis functions. Without loss of generality, we can choose to centre the basis functions on the first m training inputs $\mathbf{x}_1 \dots \mathbf{x}_m$. We then set a prior distribution over the weights $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{mm}^{-1})$, where $\mathbf{K}_{mm}$ is the upper-left $m \times m$ block of $\mathbf{K}$. Predictions which approximate (5.4) can then be made in $\mathcal{O}(nm^2)$. This method is known as the *subset of regressors* (SR) [61], and can be used for approximate Gaussian process regression over large data sets.

The SR mean prediction approaches the full GP prediction as $m \to n$, but the predictive variance at points far removed from the training data approaches σ^2 even if $m = n$. One way to overcome this problem is to augment the generalised linear model with an extra weight and basis function centred on the test point, as described in the next section. In doing so, we produce a

reduced rank Gaussian process.

5.3 From Linear Models to Reduced Rank GPs

This section reviews reduced rank Gaussian processes (RRGPs) as presented by Quiñonero-Candela and Rasmussen [55].

Consider a data set $\mathcal{D}$, from which we wish to make a prediction $p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y})$ at a test point $\mathbf{x}_*$. To do so, we will build a generalised linear model with $m \leq n$ basis functions centred on the first m data points which we will refer to as either support points or support inputs. Furthermore, we add an *extra* basis function which we centre on the test point. Overall we have $m+1$ basis functions, each with an associated weight to give a model

$$\mathbf{f} = \begin{bmatrix} \mathbf{K}_{nm} & \mathbf{k}_* \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ w_* \end{bmatrix} \tag{5.5}$$

where w_* is the extra weight, and $\mathbf{k}_* = [k(\mathbf{x}_1, \mathbf{x}_*) \dots k(\mathbf{x}_n, \mathbf{x}_*)]^{\mathrm{T}}$ is formed by evaluating the extra basis function over the training inputs. $\mathbf{K}_{nm}$ is the upper-left $n \times m$ block of $\mathbf{K}$.

We place a prior distribution over the augmented weights vector

$$\begin{bmatrix} \mathbf{w} \\ w_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}_{mm} & \mathbf{k}(\mathbf{x}_*) \\ \mathbf{k}(\mathbf{x}_*)^{\mathrm{T}} & k_{**} \end{bmatrix}^{-1}\right) \tag{5.6}$$

where the scalar $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$, and $\mathbf{k}(\mathbf{x}_*) = [k(\mathbf{x}_1, \mathbf{x}_*) \dots k(\mathbf{x}_m, \mathbf{x}_*)]^{\mathrm{T}}$ is the $m \times 1$ vector of covariances between $\mathbf{x}_*$ and the m support points.

For $m < n$, the reduced rank Gaussian process predictive distribution is

$$y_*|\mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(\bar{f}(\mathbf{x}_*),\, v(\mathbf{x}_*)\right) \tag{5.7}$$

$$\text{with} \qquad \bar{f}(\mathbf{x}_*) = \frac{1}{\sigma^2}\, \mathbf{q}(\mathbf{x}_*)^{\mathrm{T}} \boldsymbol{\Sigma}_* \left[\mathbf{K}_{nm} \,|\, \mathbf{k}_*\right]^{\mathrm{T}} \mathbf{y} \tag{5.8}$$

$$\text{and} \qquad v(\mathbf{x}_*) = \sigma^2 + \mathbf{q}(\mathbf{x}_*)^{\mathrm{T}} \boldsymbol{\Sigma}_* \mathbf{q}(\mathbf{x}_*) \tag{5.9}$$

where $\mathbf{q}(\mathbf{x}_*)^{\mathrm{T}} = \left[\mathbf{k}(\mathbf{x}_*)^{\mathrm{T}} \mid k_{**}\right]$.

The posterior distribution over the augmented weights vector is Gaussian

$$\begin{bmatrix} \mathbf{w} \\ w_* \end{bmatrix} \sim \mathcal{N}\left(\frac{1}{\sigma^2} \boldsymbol{\Sigma}_* \left[\mathbf{K}_{nm} \,|\, \mathbf{k}_* \right]^{\mathrm{T}} \mathbf{y} \,,\, \boldsymbol{\Sigma}_* \right) \tag{5.10}$$

The $(m+1) \times (m+1)$ matrix $\boldsymbol{\Sigma}_*$ is the covariance matrix of the posterior distribution over the augmented weights, and is equal to

$$\boldsymbol{\Sigma}_* = \begin{bmatrix} \boldsymbol{\Sigma}^*_{11} & \boldsymbol{\Sigma}^*_{12} \\ \boldsymbol{\Sigma}^*_{21} & \boldsymbol{\Sigma}^*_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Sigma}^{-1} & \mathbf{k}(\mathbf{x}_*) + \sigma^{-2} \mathbf{K}_{nm}^{\mathrm{T}} \mathbf{k}_* \\ \mathbf{k}(\mathbf{x}_*)^{\mathrm{T}} + \sigma^{-2} \mathbf{k}_*^{\mathrm{T}} \mathbf{K}_{nm} & k_{**} + \sigma^{-2} \mathbf{k}_*^{\mathrm{T}} \mathbf{k}_* \end{bmatrix}^{-1} \tag{5.11}$$

where the $m \times m$ matrix $\boldsymbol{\Sigma}^{-1} = \sigma^{-2} \mathbf{K}_{nm}^{\mathrm{T}} \mathbf{K}_{nm} + \mathbf{K}_{mm}$

The computational cost of prediction is an initial $\mathcal{O}(nm^2)$ to compute $\boldsymbol{\Sigma}$. Then, a prediction at each test point has a cost of $\mathcal{O}(nm)$, due to the most expensive computation $\mathbf{K}_{nm}^{\mathrm{T}} \mathbf{k}_*$. For multiple predictions we calculate $\boldsymbol{\Sigma}$ once, and then compute $\boldsymbol{\Sigma}_*$ by inversion by partitioning [55]. i.e.

$$\begin{aligned} \mathbf{r} &= \mathbf{k}(\mathbf{x}_*) + \sigma^{-2} \mathbf{K}_{nm}^{\mathrm{T}} \mathbf{k}_* \\ \rho &= k_{**} + \sigma^{-2} \mathbf{k}_*^{\mathrm{T}} \mathbf{k}_* - \mathbf{r}^{\mathrm{T}} \boldsymbol{\Sigma} \mathbf{r} \\ \boldsymbol{\Sigma}^*_{11} &= \boldsymbol{\Sigma} + \boldsymbol{\Sigma} \mathbf{r} \mathbf{r}^{\mathrm{T}} \boldsymbol{\Sigma} / \rho \\ \boldsymbol{\Sigma}^*_{12} &= -\boldsymbol{\Sigma} \mathbf{r} / \rho = \boldsymbol{\Sigma}^{*\mathrm{T}}_{21} \\ \boldsymbol{\Sigma}^*_{22} &= 1/\rho \end{aligned}$$

Note that the RRGP model is computed using a finite number of basis functions. In fact, in computing $\boldsymbol{\Sigma}$ we consider only m basis functions defined by the support points. In making a prediction at a test point, we add an extra basis function to the model. We can query the model at an infinite number of test points, so the model would appear to have an infinite number of basis functions. By adding a basis function "on demand" we have effectively built an infinite generalised linear model.

5.4 From RRGP to Full GP

It is interesting to examine what happens to the reduced rank GP if we let $m = n$, the result of which is not made explicit in [55]. This section examines this situation by using all n training inputs as support points.

The linear model's output at the training and test inputs becomes

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} = \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^{\mathrm{T}} & k_{**} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ w_* \end{bmatrix} \tag{5.12}$$

and the prior distribution over weights becomes

$$\begin{bmatrix} \mathbf{w} \\ w_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^{\mathrm{T}} & k_{**} \end{bmatrix}^{-1}\right) \tag{5.13}$$

The induced prior distribution over functions is therefore

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \Bigg| \, \mathbf{x}_*, \mathbf{X} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^{\mathrm{T}} & k_{**} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^{\mathrm{T}} & k_{**} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^{\mathrm{T}} & k_{**} \end{bmatrix}\right) \tag{5.14}$$

$$\sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^{\mathrm{T}} & k_{**} \end{bmatrix}\right) \tag{5.15}$$

We can then find the distribution of f_* conditioned on $\mathbf{f}$

$$f_* | \mathbf{f}, \mathbf{x}_*, \mathbf{X} \;\; \sim \;\; \mathcal{N}\left(\mathbf{k}_*^{\mathrm{T}} \mathbf{K}^{-1} \mathbf{f}, \;\; k_{**} - \mathbf{k}_*^{\mathrm{T}} \mathbf{K}^{-1} \mathbf{k}_*\right) \tag{5.16}$$

and the predictive distribution at $\mathbf{x}_*$

$$y_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y} \;\; \sim \;\; \mathcal{N}\Big(\mathbf{k}_*^{\mathrm{T}} (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} \,, \;\; \sigma^2 + k_{**} - \mathbf{k}_*^{\mathrm{T}} (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_*\Big) \tag{5.17}$$

which is *exactly* the same as the predictive distribution for the full Gaussian process, given by equations (1.13) and (1.14) on page 8.

In summary, a reduced rank Gaussian process with n support points set equal to all n training inputs is equivalent to a full Gaussian process.

5.5 From Linear Models to Non-stationary Gaussian Processes

Section 5.3 showed how we could construct a (non-degenerate) reduced rank Gaussian process by adding an extra weight and basis function for every test

point. In section 5.4 we saw how the full GP is recovered when we use all of the training inputs as support points, and use equation (5.13) as the prior distribution over the augmented weights. The treatment in these sections assumed that all basis functions were defined according to a single kernel $k(\cdot,\cdot)$, where the i^{th} basis function $\phi_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i)$. In this section, we construct non-stationary Gaussian processes, by allowing the basis functions to vary depending on their locations. We therefore describe another method for constructing non-stationary Gaussian processes, which is distinct from the non-linear mapping approach of Gibbs [17], and the spatially variable covariance function approach described by Paciorek [52, 53].

We continue with a set of m support points, associated with m basis functions and weights. However, we set the i^{th} basis function $\phi_i(\mathbf{x})$ as a Gaussian with covariance matrix $\mathbf{B}_i$ centred on the i^{th} support point

$$\phi_i(\mathbf{x}) = \exp\left(-\tfrac{1}{2}(\mathbf{x}-\mathbf{x}_i)^{\mathrm{T}}\mathbf{B}_i^{-1}(\mathbf{x}-\mathbf{x}_i)\right) \tag{5.18}$$

For each test point $\mathbf{x}_*$ we add an extra weight w_* and basis function $\phi_*(\mathbf{x})$ with covariance matrix $\mathbf{B}_*$, centred on $\mathbf{x}_*$. The vector $\boldsymbol{\phi}_*^{\mathrm{T}} = [\phi_*(\mathbf{x}_1)\ldots\phi_*(\mathbf{x}_n)]^{\mathrm{T}}$ is constructed from the new basis function evaluated at the training inputs. We set a spherical Gaussian prior distribution over the augmented weights vector

$$\begin{bmatrix}\mathbf{w}\\ w_*\end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \alpha^2\mathbf{I}\right) \tag{5.19}$$

where α^2 is the hyperparameter encoding the prior weights variance.

The predictive distribution at $\mathbf{x}_*$ becomes

$$y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y} \;\sim\; \mathcal{N}(\bar{f}(\mathbf{x}_*), v(\mathbf{x}_*)) \tag{5.20}$$

$$\text{with} \qquad \bar{f}(\mathbf{x}_*) = \frac{1}{\sigma^2}\mathbf{q}(\mathbf{x}_*)^{\mathrm{T}}\mathbf{A}_*^{-1}[\boldsymbol{\Phi}^{\mathrm{T}}|\boldsymbol{\phi}_*]^{\mathrm{T}}\mathbf{y} \tag{5.21}$$

$$\text{and} \qquad v(\mathbf{x}_*) = \sigma^2 + \mathbf{q}(\mathbf{x}_*)^{\mathrm{T}}\mathbf{A}_*^{-1}\mathbf{q}(\mathbf{x}_*) \tag{5.22}$$

where $\mathbf{q}(\mathbf{x}_*)^{\mathrm{T}} = [\boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}} \quad \phi_*(\mathbf{x}_*)]^{\mathrm{T}}$ is the vector formed by evaluating the $m+1$ basis functions at $\mathbf{x}_*$. The posterior distribution over the augmented weights is Gaussian with covariance matrix

$$\mathbf{A}_* = \begin{bmatrix}\mathbf{A} & \sigma^{-2}\boldsymbol{\Phi}\boldsymbol{\phi}_*\\ \sigma^{-2}\boldsymbol{\phi}_*^{\mathrm{T}}\boldsymbol{\Phi}^{\mathrm{T}} & \sigma^{-2}\boldsymbol{\phi}_*^{\mathrm{T}}\boldsymbol{\phi}_* + \alpha^{-2}\end{bmatrix} \tag{5.23}$$

where $\mathbf{A} = \sigma^{-2}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}} + \alpha^{-2}\mathbf{I}$ is the posterior covariance over the (non-augmented) weights $\mathbf{w}$ and is independent of any test point $\mathbf{x}_*$. For prediction $\mathbf{A}^{-1}$ can be precomputed and $\mathbf{A}_*^{-1}$ can be calculated as a partitioned inverse.

As the test point moves away from the training data, $\mathbf{x}_* \to \infty$, the predictive variance $v(\mathbf{x}_*) \to \sigma^2 + \alpha^2$, which seems a sensible prior belief.

If we parameterise the basis function covariance matrices with parameters $\boldsymbol{\theta}$, then the marginal likelihood is

$$\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, \sigma^2, \alpha^2 \quad \sim \quad \mathcal{N}(\mathbf{0}\,,\, \alpha^2\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \sigma^2\mathbf{I}) \tag{5.24}$$

which provides us with a mechanism to learn $\boldsymbol{\theta}, \sigma^2$ and α^2. That is, we learn the basis function covariance matrices $\mathbf{B}_i$, the noise variance σ^2 and the prior weight variance α^2 by maximising the marginal likelihood.

Making a prediction using equation (5.20) requires us to specify a new Gaussian basis function, defined by $\mathbf{B}_*$. There are many ways to do this, but we would like $\mathbf{B}_* \to \mathbf{B}_i$ as $\mathbf{x}_* \to \mathbf{x}_i$. One approach is to let $\mathbf{B}_*$ equal the weighted sum of the covariances of the m basis functions, with the i^{th} weight $u_i = \frac{1}{|\mathbf{x}_* - \mathbf{x}_i| + \epsilon}$

$$\mathbf{B}_* = \lim_{\epsilon \to 0} \frac{\sum_{j=1}^{M} u_j \mathbf{B}_j}{\sum_{i=1}^{M} u_i} \tag{5.25}$$

So, if $\mathbf{x}_*$ is coincident with $\mathbf{x}_i$ but separate from all other training inputs then $\mathbf{B}_* = \mathbf{B}_i$. If $\mathbf{x}_*$ is coincident with $\mathbf{x}_i$ and $\mathbf{x}_j$ then $\mathbf{B}_* = \frac{1}{2}\mathbf{B}_i + \frac{1}{2}\mathbf{B}_j$.

More generally, we could specify a parametric function $\mathbf{g}_\theta : \mathbb{R}^D \to \mathbb{R}^D \times \mathbb{R}^D$, that defines a covariance matrix for any input vector, $\mathbf{B} = \mathbf{g}_\theta(\mathbf{x})$. The range of $\mathbf{g}$ must be such that all $\mathbf{B}$ are positive definite.

A simple illustration of a non-stationary Gaussian process generated in this way is shown in figure 5.1. This example has 24 basis functions, each centred on a training input. The i^{th} basis function $\phi_i(x)$ at point x_i is a squared exponential:

$$\phi_i(x) = \exp\left(-\tfrac{1}{2}\frac{(x - x_i)^2}{r(x_i)^2}\right) \tag{5.26}$$

$r(x_i)$ is the length scale for the i^{th} basis function:

$$r(x_i) = r_1 + \frac{r_2 - r_1}{1 + \exp(-5x_i)} \tag{5.27}$$

where $r_1 = 0.75$ and $r_2 = 0.075$.

So as $x_i \to \infty$, $r(x_i) \to 0.075$, and as $x_i \to -\infty$, $r(x_i) \to 0.75$. The function is therefore smooth for $x < 0$ and becomes rapidly varying for $x > 0$. Note that the predictive variance as x moves away from data approaches a constant non-zero value, and does not decrease to the noise level, unlike the degenerate Gaussian processes discussed earlier.

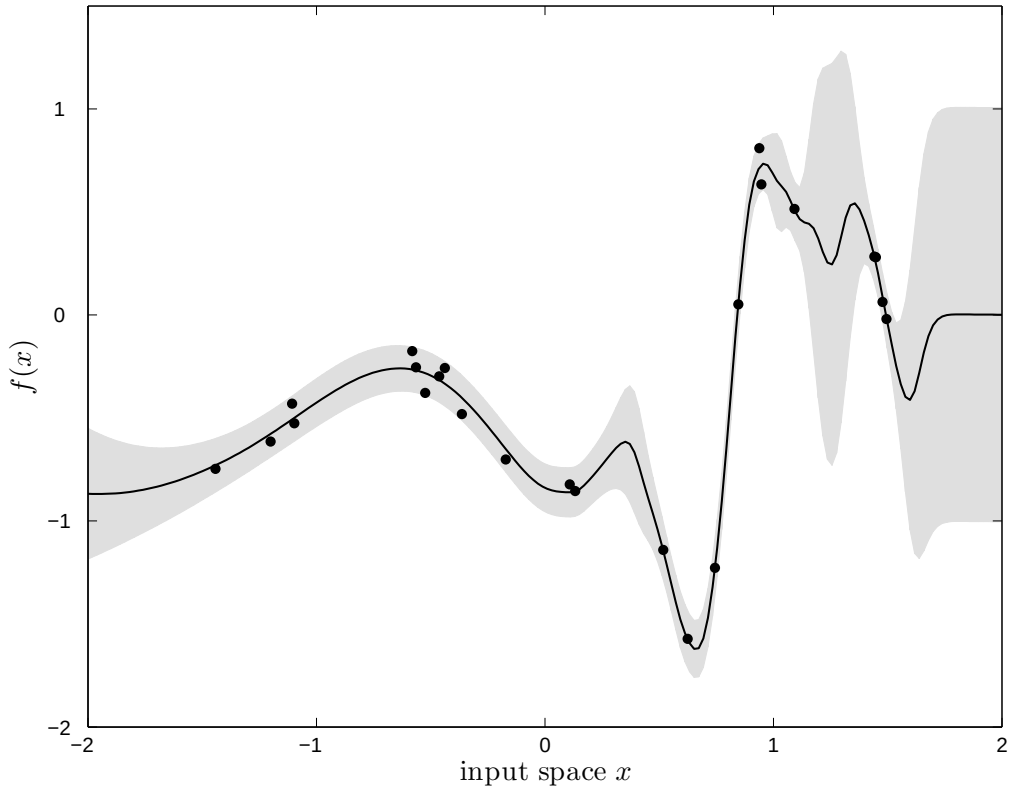

Figure 5.1: Example of a non-stationary Gaussian process constructed by augmenting a non-stationary generalised linear model with an extra weight and basis function at every test point. Each of the 24 basis functions are centred on the training inputs (dots). The predictive mean is shown by the black line and the 95% confidence interval is shaded grey.

In summary, we have shown here how we can use the reduced rank Gaussian processes framework to produce non-stationary GP models.

5.6 Discrete Process Convolution Models

In the previous sections, in an effort to reduce the computational cost of GP regression, we examined approximations to the full GP predictive distribution. In this section, we consider yet another method of approximation, known as the discrete process convolution model.

A continuous Gaussian process can be constructed by convolving a kernel $h(t)$, with a noise process $w(t)$ that is non-zero at a number of discrete points [24, 10, 29]. The result is a discrete process convolution (DPC) model. For example, if we use the noise process $w(t) = \sum_{j=1}^{m} w_j \delta(t - t_j)$, the output is

$$y(t) = w(t) * h(t) \tag{5.28}$$

$$= \sum_{j=1}^{m} w_j h(t - t_j) \tag{5.29}$$

where $t_1 \dots t_m$ are support points and $w_1 \dots w_m$ are the Gaussian distributed weights of the latent noise process. The motivation for building such models is that the computational cost of making a prediction is $\mathcal{O}(m^2 n)$.

Upon inspection, we see that equation (5.29) is a generalised linear model with $h(t - t_1) \dots h(t - t_m)$ as its basis functions. We have a Gaussian prior distribution over the weights $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, so from the discussion in the previous sections, $y(t)$ is a Gaussian process. If we set the support points equal to m of the training inputs then the DPC model is equivalent to the SR model in section 5.2. Like the SR method, the DPC Gaussian process is a degenerate Gaussian process. To make it non-degenerate, we add an extra weight w_* when making predictions at a test point t_*. We redefine the noise process as $w(t) = w_* \delta(t - t_*) + \sum_{j=1}^{m} w_j \delta(t - t_j)$. The resulting model is then equivalent to a reduced rank Gaussian process model as defined above.

5.7 Summary

A reduced rank Gaussian process can be constructed by augmenting a generalised linear model at test time. That is, before querying a RRGP model with a test input, we add an extra weight and basis function to the model. Using this augmentation results in a reduced rank Gaussian process that is

non-degenerate. Furthermore, we can construct non-stationary Gaussian processes using augmented generalised linear models with basis functions that vary over space.

Reduced rank Gaussian processes exist as a method to reduce the computational cost of Gaussian process regression. There are of course many other methods for doing just this, as outlined in section 1.4 on page 10. However, in this chapter the focus has been on the reduced rank method mainly because of the result in section 5.4, where it was shown that the full Gaussian process predictive distribution is recovered from a reduced rank model by setting the support set equal to the training input set. Reduced rank models are themselves obtained by augmenting generalised linear models with extra weights at test time. Overall, there is a natural link between generalised linear models, reduced rank Gaussian processes, and full rank Gaussian processes. In fact, one might like to think of a Gaussian process as an augmented generalised linear model with positive definite kernels centred on the training inputs as basis functions.

Chapter 6

Reduced Rank Dependent Gaussian Processes

Chapter 3 showed how to construct sets of dependent Gaussian processes and use them to perform multivariate regression. To do so required the inversion of $n \times n$ matrices, where n was the total number of observations across all outputs. Exact inversion of these matrices has complexity $\mathcal{O}(n^3)$, and becomes prohibitive for large n.

The previous chapter showed how to reduce the computational cost of Gaussian process regression by using approximations known as reduced rank Gaussian processes. This chapter extends this approximation to the multiple output case by defining reduced rank dependent Gaussian processes.

6.1 Multiple Output Linear Models

Consider a generalised linear model over two outputs. We have n_x observations of the first output $\mathcal{D}_x = \{(\mathbf{x}_1, y_1^x) \dots (\mathbf{x}_{n_x}, y_{n_x}^x)\}$, and n_z observations of the second $\mathcal{D}_z = \{(\mathbf{z}_1, y_1^z) \dots (\mathbf{z}_{n_z}, y_{n_z}^z)\}$. For notational convenience, we combine the input vectors into matrices; $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_{n_x}], \mathbf{Z} = [\mathbf{z}_1 \dots \mathbf{z}_{n_z}]$, and combine the targets into vectors $\mathbf{y}_x = [y_1^x \dots y_{n_x}^x]^{\mathrm{T}}, \mathbf{y}_z = [y_1^z \dots y_{n_z}^z]^{\mathrm{T}}$.

The model consists of a set of $2(m_x + m_z)$ basis functions and $m_x + m_z$ weights, where $m_x \leq n_x$ and $m_z \leq n_z$. The basis functions are divided into two sets:

$2m_x$ of these are centred on the first m_x training inputs in $\mathbf{X}$, and the remaining $2m_z$ basis functions are centred on the first m_z training inputs in $\mathbf{Z}$. Overall, there are $m_x + m_z$ support points and each is associated with two basis functions - one for each output.

The weights are also divided into two sets: m_x weights are collected into the vector $\mathbf{v}$ and control the contribution of the m_x support points from $\mathbf{X}$, and the remaining m_z weights in the vector $\mathbf{w}$ control the contribution from the m_z support points in $\mathbf{Z}$.

Consider two test points, $\mathbf{x}_*$ and $\mathbf{z}_*$ (one for each output), and add an extra basis function and weight for each (the extra weights are v_* and w_* respectively). The augmented model evaluated over the training inputs produces $\mathbf{f}_x = [f_x(\mathbf{x}_1) \dots f_x(\mathbf{x}_{n_x})]^{\mathrm{T}}$ and $\mathbf{f}_z = [f_z(\mathbf{z}_1) \dots f_z(\mathbf{z}_{n_z})]^{\mathrm{T}}$ as follows:

$$\begin{bmatrix} \mathbf{f}_x \\ \mathbf{f}_z \end{bmatrix} = [\mathbf{\Phi} \mid \mathbf{A}]\, \mathbf{u} \tag{6.1}$$

$$= \mathbf{\Phi} \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix} + \mathbf{A} \begin{bmatrix} v_* \\ w_* \end{bmatrix} \tag{6.2}$$

where the augmented weights vector $\mathbf{u} = \begin{bmatrix} \mathbf{v}^{\mathrm{T}} & \mathbf{w}^{\mathrm{T}} & v_* & w_* \end{bmatrix}^{\mathrm{T}}$, and the remaining components are defined in what follows.

The design matrix $\mathbf{\Phi}$ is independent of the test points and is block partitioned as:

$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{\Phi}_{xx} & \mathbf{\Phi}_{xz} \\ \mathbf{\Phi}_{zx} & \mathbf{\Phi}_{zz} \end{bmatrix} \tag{6.3}$$

where the partitions are built by evaluating the set of $2(m_x + m_z)$ basis functions over the training inputs, as in equations (6.4) to (6.7) below. The set of basis functions is: $\{k_{xx}(\mathbf{x}, \mathbf{x}_1) \dots k_{xx}(\mathbf{x}, \mathbf{x}_{m_x})$, $k_{xz}(\mathbf{x}, \mathbf{z}_1) \dots k_{xz}(\mathbf{x}, \mathbf{z}_{m_z})$, $k_{zx}(\mathbf{z}, \mathbf{x}_1) \dots k_{zx}(\mathbf{z}, \mathbf{x}_{m_x})$, $k_{zz}(\mathbf{z}, \mathbf{z}_1) \dots k_{zz}(\mathbf{z}, \mathbf{z}_{m_z})\}$.

$$\mathbf{\Phi}_{xx} = \begin{bmatrix} k_{xx}(\mathbf{x}_1, \mathbf{x}_1) & \dots & k_{xx}(\mathbf{x}_1, \mathbf{x}_{m_x}) \\ \vdots & \ddots & \vdots \\ k_{xx}(\mathbf{x}_{n_x}, \mathbf{x}_1) & \dots & k_{xx}(\mathbf{x}_{n_x}, \mathbf{x}_{m_x}) \end{bmatrix} \tag{6.4}$$

$$\mathbf{\Phi}_{xz} = \begin{bmatrix} k_{xz}(\mathbf{x}_1, \mathbf{z}_1) & \dots & k_{xz}(\mathbf{x}_1, \mathbf{z}_{m_z}) \\ \vdots & \ddots & \vdots \\ k_{xz}(\mathbf{x}_{n_x}, \mathbf{z}_1) & \dots & k_{xz}(\mathbf{x}_{n_x}, \mathbf{z}_{m_z}) \end{bmatrix} \tag{6.5}$$

$$\mathbf{\Phi}_{zx} = \begin{bmatrix} k_{zx}(\mathbf{z}_1, \mathbf{x}_1) & \dots & k_{zx}(\mathbf{z}_1, \mathbf{x}_{m_x}) \\ \vdots & \ddots & \vdots \\ k_{zx}(\mathbf{z}_{n_z}, \mathbf{x}_1) & \dots & k_{zx}(\mathbf{z}_{n_z}, \mathbf{x}_{m_x}) \end{bmatrix} \tag{6.6}$$

$$\mathbf{\Phi}_{zz} = \begin{bmatrix} k_{zz}(\mathbf{z}_1, \mathbf{z}_1) & \dots & k_{zz}(\mathbf{z}_1, \mathbf{z}_{m_z}) \\ \vdots & \ddots & \vdots \\ k_{zz}(\mathbf{z}_{n_z}, \mathbf{z}_1) & \dots & k_{zz}(\mathbf{z}_{n_z}, \mathbf{z}_{m_z}) \end{bmatrix} \tag{6.7}$$

In general, the design matrix does not have to be built from positive definite functions so the basis functions can be arbitrary. Nevertheless, in this section we use valid kernel $k_{xx}(\cdot,\cdot), k_{zz}(\cdot,\cdot)$ and *cross-kernel* functions $k_{xz}(\cdot,\cdot), k_{zx}(\cdot,\cdot)$ as basis functions. For example, the kernel functions could be set equal to the covariance and cross-covariance functions derived from Gaussian impulse responses as in appendix A.2.

The $(n_x + n_z) \times 2$ matrix $\mathbf{A}$ is the contribution to the model of the two extra weights and basis functions:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_x \\ \mathbf{A}_z \end{bmatrix} = \left[\begin{array}{cc} k_{xx}(\mathbf{x}_1, \mathbf{x}_*) & k_{xz}(\mathbf{x}_1, \mathbf{z}_*) \\ \vdots & \vdots \\ k_{xx}(\mathbf{x}_{n_x}, \mathbf{x}_*) & k_{xz}(\mathbf{x}_{n_x}, \mathbf{z}_*) \\ \hline k_{zx}(\mathbf{z}_1, \mathbf{x}_*) & k_{zz}(\mathbf{z}_1, \mathbf{z}_*) \\ \vdots & \vdots \\ k_{zx}(\mathbf{z}_{n_z}, \mathbf{x}_*) & k_{zz}(\mathbf{z}_{n_z}, \mathbf{z}_*) \end{array}\right] \tag{6.8}$$

We define a Gaussian prior distribution over the augmented weights vector:

$$\mathbf{u} \big| \mathbf{x}_*, \mathbf{z}_*, \mathbf{X}, \mathbf{Z} \;\sim\; \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{\Omega} & \mathbf{B} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{\Lambda} \end{bmatrix}^{-1}\right) \tag{6.9}$$

where $\mathbf{\Omega}$ is formed by evaluating the basis functions at the support points, rather than all the training inputs and is block partitioned as:

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{\Omega}_{xx} & \mathbf{\Omega}_{xz} \\ \mathbf{\Omega}_{zx} & \mathbf{\Omega}_{zz} \end{bmatrix} \tag{6.10}$$

$\mathbf{\Omega}_{ij}$ is the upper-most $m_i \times m_j$ sub-block of $\mathbf{\Phi}_{ij}$ (for $i, j \in \{x, z\}$), meaning that $\mathbf{\Omega}$ is a submatrix of $\mathbf{\Phi}$.

The matrix $\mathbf{B}^{\mathrm{T}} = \begin{bmatrix} \mathbf{B}_x^{\mathrm{T}} & \mathbf{B}_z^{\mathrm{T}} \end{bmatrix}$, where $\mathbf{B}_x$ equals the top m_x rows of $\mathbf{A}_x$, and $\mathbf{B}_z$ equals the top m_z rows of $\mathbf{A}_z$.

The 2×2 matrix $\mathbf{\Lambda}$ has the following elements:

$$\mathbf{\Lambda} = \begin{bmatrix} k_{xx}(\mathbf{x}_*, \mathbf{x}_*) & k_{xz}(\mathbf{x}_*, \mathbf{z}_*) \\ k_{zx}(\mathbf{z}_*, \mathbf{x}_*) & k_{zz}(\mathbf{z}_*, \mathbf{z}_*) \end{bmatrix} \tag{6.11}$$

and is independent of the training data.

The model assumes Gaussian noise on the outputs, where each output has a separate noise variance, σ_x^2 and σ_z^2. This assumption allows us to find the likelihood function

$$\mathbf{y} \big| \mathbf{u}, \mathbf{x}_* \mathbf{z}_*, \mathbf{X}, \mathbf{Z} \;\sim\; \mathcal{N}\left(\mathbf{F}\mathbf{u}\,,\; \mathbf{\Psi}\right) \tag{6.12}$$

where $\mathbf{y}^{\mathrm{T}} = [\mathbf{y}_x^{\mathrm{T}} \quad \mathbf{y}_z^{\mathrm{T}}]$, and $\mathbf{F} = \begin{bmatrix} \mathbf{\Phi} \,|\, \mathbf{A} \end{bmatrix}$. The covariance matrix for this Gaussian likelihood function is

$$\mathbf{\Psi} = \begin{bmatrix} \sigma_x^2 \mathbf{I}_x & \mathbf{0} \\ \mathbf{0} & \sigma_z^2 \mathbf{I}_z \end{bmatrix} \tag{6.13}$$

where $\mathbf{I}_x$ and $\mathbf{I}_z$ are $n_x \times n_x$ and $n_z \times n_z$ identity matrices.

From the likelihood function of the augmented weights and the prior distribution over augmented weights, we can find the posterior distribution over the augmented weights:

$$\mathbf{u} \big| \mathbf{y}, \mathbf{x}_* \mathbf{z}_*, \mathbf{X}, \mathbf{Z} \sim \mathcal{N}\left(\boldsymbol{\mu}, \mathbf{\Sigma}_*\right) \tag{6.14}$$

where

$$\mathbf{\Sigma}_*^{-1} = \mathbf{F}^{\mathrm{T}} \mathbf{\Psi}^{-1} \mathbf{F} + \begin{bmatrix} \mathbf{\Omega} & \mathbf{B} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{\Lambda} \end{bmatrix}^{-1} \tag{6.15}$$

$$\boldsymbol{\mu} = \mathbf{\Sigma}_* \mathbf{F}^{\mathrm{T}} \mathbf{\Psi}^{-1} \mathbf{y} \tag{6.16}$$

To simplify, define the matrix $\mathbf{Q}$ as

$$\mathbf{Q} = \begin{bmatrix} \mathbf{B}^{\mathrm{T}} & \mathbf{\Lambda} \end{bmatrix} \tag{6.17}$$

Then, the predictive distribution at test points $\mathbf{x}_*$ and $\mathbf{z}_*$ is Gaussian with predictive mean

$$\begin{bmatrix} m_x(\mathbf{x}_*) \\ m_z(\mathbf{z}_*) \end{bmatrix} = \mathbf{Q}\boldsymbol{\mu} \tag{6.18}$$

and predictive variance

$$\mathbf{V}(\mathbf{x}_*, \mathbf{z}_*) = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_z^2 \end{bmatrix} + \mathbf{Q}\mathbf{\Sigma}_*\mathbf{Q}^{\mathrm{T}} \tag{6.19}$$

An illustration of a reduced-rank dependent Gaussian process is shown in figure 6.1. This example has two outputs, with $n_1 = n_2 = 10$ and $m_1 = m_2 = 8$, and noise variance $\sigma_1^2 = \sigma_2^2 = 0.025^2$. The model was constructed using Gaussian basis functions equivalent to the auto and cross-covariance functions derived in appendix A.2. The training data for output 1 is restricted to $x > 0$, but output 2 has training inputs spread across the whole input space. Note that the model of output 1 gives predictions with low uncertainty at certain points, even though there is no output 1 training data associated with those points. Predictions at these points have low uncertainty because of a dependency on the training data from output 2.

6.2 Reduced Rank Dependent GP for $n = m$

Here, we show that the predictive distribution for a reduced rank dependent GP with all of the training inputs used as support points is equivalent to the full dependent GP predictive distribution. (In fact we show it for the two output situation, but the result trivially extends to the general case). See section 6.1 for definitions of notations if required.

Consider the model evaluated over the training data and two test points

$$\begin{bmatrix} \mathbf{f}_x^{\mathrm{T}} & \mathbf{f}_z^{\mathrm{T}} & f_{x*} & f_{z*} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \mathbf{\Phi} & \mathbf{A} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{\Lambda} \end{bmatrix} . \mathbf{u} \tag{6.20}$$

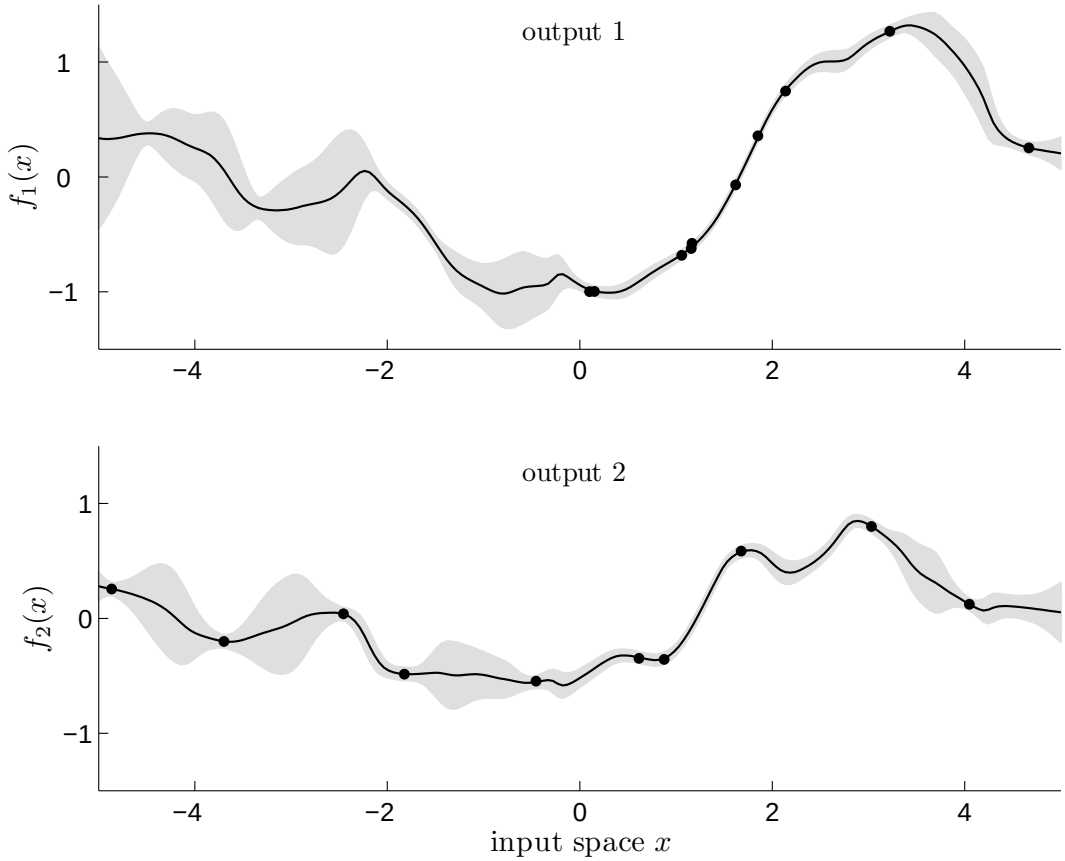

Figure 6.1: Example of a two-output reduced-rank dependent Gaussian process constructed by augmenting a two-output generalised linear model with an extra weight and basis function at every test point. Each output has 10 training inputs (black dots), and the reduced rank model is built from a subset of 8 training inputs from each output. The predictive mean is shown by the black lines and the 95% confidence intervals are shaded grey.

If we set $m_x = n_x$ and $m_z = n_z$ (use all of the training inputs as support points), then $\mathbf{B} = \mathbf{A}$ and $\mathbf{\Omega} = \mathbf{\Phi} = \mathbf{K}$ where $\mathbf{K}$ is the full Gram matrix produced from all $n_x + n_z$ training inputs (equivalent to the Gram matrix we would obtain if we were building the full dependent GP model). Equation (6.20) then becomes

$$\begin{bmatrix}\mathbf{f}_x^{\mathrm{T}} & \mathbf{f}_z^{\mathrm{T}} & f_{x*} & f_{z*}\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}\mathbf{K} & \mathbf{A} \\ \mathbf{A}^{\mathrm{T}} & \mathbf{\Lambda}\end{bmatrix} . \mathbf{u} = \mathbf{K}_* \mathbf{u} \tag{6.21}$$

The prior distribution over weights becomes

$$\mathbf{u}\big|\mathbf{x}_*, \mathbf{z}_*, \mathbf{X}, \mathbf{Z} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{\Omega} & \mathbf{B} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{\Lambda} \end{bmatrix}^{-1}\right) \tag{6.22}$$

$$\sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{A} \\ \mathbf{A}^{\mathrm{T}} & \mathbf{\Lambda} \end{bmatrix}^{-1}\right) \tag{6.23}$$

$$\sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}_*^{-1}\right) \tag{6.24}$$

Therefore, the prior distribution over function values is

$$\begin{bmatrix}\mathbf{f}_x & \mathbf{f}_z & f_{x_*} & f_{z_*}\end{bmatrix}^{\mathrm{T}}\big|\mathbf{x}_*, \mathbf{z}_*, \mathbf{X}, \mathbf{Z} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}_*\right) \tag{6.25}$$

We can find the distribution of f_{x_*} and f_{z_*} conditioned on $\mathbf{f}_x$ and $\mathbf{f}_z$

$$\begin{bmatrix}f_{x_*} & f_{z_*}\end{bmatrix}^{\mathrm{T}}\left|\mathbf{f}_x, \mathbf{f}_z, \mathbf{x}_*, \mathbf{z}_*, \mathbf{X}, \mathbf{Z}\right. \sim \mathcal{N}\left(\mathbf{A}^{\mathrm{T}}\mathbf{K}^{-1}\begin{bmatrix}\mathbf{f}_x \\ \mathbf{f}_z\end{bmatrix}, \mathbf{\Lambda} - \mathbf{A}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{A}\right) \tag{6.26}$$

resulting in a predictive mean

$$\begin{bmatrix} m_x(\mathbf{x}_*) \\ m_z(\mathbf{z}_*) \end{bmatrix} = \mathbf{A}^{\mathrm{T}}(\mathbf{K} + \mathbf{\Psi})^{-1}\begin{bmatrix}\mathbf{y}_x \\ \mathbf{y}_z\end{bmatrix} \tag{6.27}$$

and predictive variance

$$\mathbf{V}(\mathbf{x}_*, \mathbf{z}_*) = \begin{bmatrix}\sigma_x^2 & 0 \\ 0 & \sigma_z^2\end{bmatrix} + \mathbf{\Lambda} - \mathbf{A}^{\mathrm{T}}(\mathbf{K} + \mathbf{\Psi})^{-1}\mathbf{A} \tag{6.28}$$

The predictive mean and variance in equations (6.27) and (6.28) are the same as those obtained from a dependent GP model with two outputs and covariance matrix $\mathbf{K}$. (Compare to equation (5.17) (page 76) and equations (3.26) and (3.27) (page 43)).

Therefore, if we use all of the training inputs as support points, (i.e. set $m_x = n_x$ and $m_z = n_z$), then we recover the full dependent GP predictive distribution. That is, the reduced rank dependent GP predictive distribution approximates the full dependent GP predictive distribution, and the approximation becomes exact when we use all of the training inputs as support points.

6.3 Multivariate DPC Models

In section 5.6 (page 80), we saw how DPC models can be defined which reduce the complexity of GP regression to $\mathcal{O}(m^2 n)$. The DPC framework can be extended to the multivariate case [29], so it is possible to make predictions across multiple dependent outputs in $\mathcal{O}(m^2 n)$. However, as discussed in section 5.6, the DPC model is a degenerate Gaussian process, and can therefore underestimate the predictive variance, especially at test points far removed from the training inputs. This can be corrected by adding additional weights to the latent noise source, which results in a model equivalent to the reduced rank dependent Gaussian process model described above.

6.4 Non-stationary Dependent GPs

In section 5.5 we saw how we can construct non-stationary GPs from non-stationary generalised linear models. The same methodology can be applied to reduced rank dependent Gaussian processes. That is, we can define generalised linear models over multiple outputs with basis functions that vary across the input space. If we add extra basis functions and weights when making predictions we construct a set of non-stationary, reduced-rank dependent Gaussian process. Of course, as in section 6.2, if we use the all of the training inputs as support points then we recover a set of non-stationary, full-rank dependent Gaussian process.

6.5 Summary

In this chapter, we have shown how the reduced rank Gaussian process regression methodology can be extended to the multiple output case. We have also shown that in the limit, when the support input set is equal to the training input set, the resulting reduced rank process is in fact full rank, and is equivalent to a dependent Gaussian process as introduced in chapter 3

Chapter 7

Rotated Covariance Functions

7.1 Introduction

Thus far, this thesis has discussed and demonstrated Gaussian processes with relatively simple (either isotropic or axis-aligned), squared-exponential (Gaussian) covariance functions. In this chapter, we discuss and introduce parameterisations of the squared-exponential function that provide a greater degree of modelling power.

A simple GP model over $\mathbb{R}^D$ is defined by the following covariance function:

$$\mathrm{cov}(\mathbf{x}_i, \mathbf{x}_j) = \exp(v) \exp\left(-\tfrac{1}{2} \sum_{d=1}^{D} \frac{\left(x_i^{(d)} - x_j^{(d)} \right)^2}{\exp(2r_d)} \right) + \delta_{ij} \exp(2\beta) \qquad (7.1)$$

where $x_i^{(d)}$ is the d^{th} component of $\mathbf{x}_i$. That is, $\mathbf{x}_i = [x_i^{(1)} \ldots x_i^{(D)}]^{\mathrm{T}}$.

This covariance function has a squared exponential, or Gaussian component defined by the parameters $r_1 \ldots r_D$, where $\exp(r_d)$ is the length scale associated with the d^{th} axis direction. So if r_d is small, the model will vary rapidly in the direction parallel to the d^{th} axis since the covariance drops rapidly in this direction. Conversely, if r_d is large, the model will be very smooth in the d^{th} axis direction, and the d^{th} components of the inputs $\mathbf{x}_i$ and $\mathbf{x}_j$ could be said to be largely irrelevant, as in the automatic relevance determination (ARD) method for neural networks [37, 46, 34]. This covariance function is referred to as "axis-aligned" in the sense that the Gaussian component has principal axes aligned with the coordinate axes.

In this thesis, hyperparameters such as v, r_d, and β usually appear in covariance functions in exponentiated form. This ensures that the resulting covariance function is positive definite, even if any of these hyperparameters are negative. We are therefore free to use Gaussian prior distributions over the hyperparameters, meaning quantities such as $\exp(v)$, or $\exp(r_d)$ are log-normal distributed, and hence naturally positive.

Samples can be drawn from the distribution of functions defined by any GP covariance function. To do so, we select a set of n test points, $\mathbf{x}_1 \dots \mathbf{x}_n$, and generate the covariance matrix $\mathbf{C}$ that has its (i^{th}, j^{th}) entry defined by the covariance function $\mathrm{cov}(\mathbf{x}_i, \mathbf{x}_j)$. A vector of function values $\mathbf{f}$ can then be generated by drawing a sample from a zero-mean multivariate Gaussian with covariance matrix equal to $\mathbf{C}$. If we calculate the eigen-decomposition $\mathbf{C} = \mathbf{V}^{\mathrm{T}}\mathbf{E}\mathbf{V}$, then $\mathbf{f}$ can be generated simply as $\mathbf{f} = \mathbf{V}^{\mathrm{T}}\mathbf{E}^{\frac{1}{2}}\mathbf{V}\mathbf{z}$, where $\mathbf{z}$ is a normally distributed vector of length n. Alternatively, we can use the Cholesky decomposition [91], $\mathbf{C} = \mathbf{L}^{\mathrm{T}}\mathbf{L}$ and generate samples $\mathbf{f} = \mathbf{L}^{\mathrm{T}}\mathbf{z}$, [61].

Figure 7.1(left) shows a sample function generated by sampling from a GP prior distribution with the above covariance function. Notice how the features of this sample are axis aligned. We see no particular pattern as we move in the direction that bisects the axes. Compare this to the second GP sample shown in 7.1, which was generated by simply rotating the coordinate axis after generating a sample from the GP. Clearly, this function has features that are not aligned to the coordinate axes. In a regression context, we could say that the dependent (output) variables are correlated. That is, the behaviour of one output provides information about the behaviour of the other.

If the function to be modelled contains features that are not axis aligned (the output variables are correlated), then it makes sense to use a GP with a covariance function that is capable of capturing this relationship. To do so, consider the following covariance function defined not by a length scale in each direction, but by a full covariance matrix $\mathbf{\Sigma}$.

$$\mathrm{cov}(\mathbf{x}_i, \mathbf{x}_j) = \exp(v) \exp\left(-\tfrac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{\Sigma}(\mathbf{x}_i - \mathbf{x}_j)\right) + \delta_{ij}\exp(2\beta) \qquad (7.2)$$

$\mathbf{\Sigma}$ is a positive definite matrix with eigenvalues corresponding to inverse length scales in the directions defined by the eigenvectors. To parameterise this rotated covariance function, we need a parameterisation that only allows positive definite $\mathbf{\Sigma}$. The following sections will examine various methods to

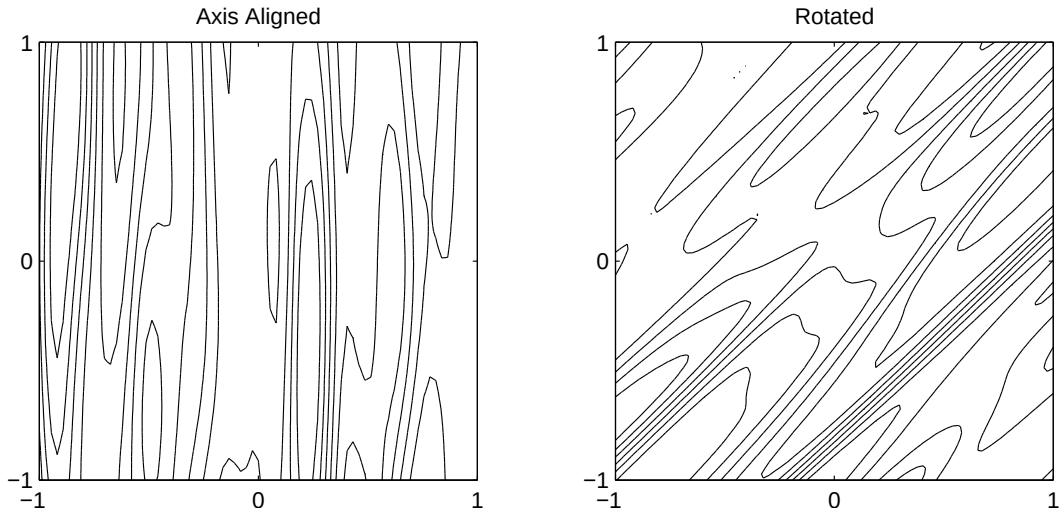

Figure 7.1: Contours of the samples generated from a GP with an axis-aligned (left) and rotated (right) squared-exponential covariance function

---

do this.

7.2 Parameterisations based on the Cholesky Decomposition

Vivarelli and Williams [82] describe a parameterisation of $\mathbf{\Sigma}$ based on the Cholesky decomposition, $\mathbf{\Sigma} = \mathbf{U}^{\mathrm{T}}\mathbf{U}$, where $\mathbf{U}$ is an upper triangular matrix with positive entries on the diagonal:

$$\mathbf{U} = \begin{bmatrix} \exp(u_{1,1}) & u_{1,2} & \dots & u_{1,D} \\ 0 & \exp(u_{2,2}) & \dots & u_{2,D} \\ 0 & 0 & \dots & u_{3,D} \\ \dots & \dots & \dots & \exp(u_{D,D}) \end{bmatrix} \tag{7.3}$$

Thus we require $D(D+1)/2$ parameters to fully parameterise an arbitrary positive definite matrix $\mathbf{\Sigma}$.

When maximising the likelihood function of hyperparameters or the posterior density over hyperparameters, it is generally useful to make use of gradient information. To do so requires the calculation of the derivative of the covari-

ance function with respect to the hyperparameters. For the Cholesky decomposition parameterisation this is relatively easy. In fact, for hyperparameter $u_{i,j}$, we find $\partial \mathbf{U}/\partial u_{i,j}$ is a matrix of zeroes apart from the $(i,j)^{th}$ element which equals $\exp(u_{i,j})$ if $i = j$ and 1 otherwise.

One problem with this parameterisation is that it is hard to specify a prior distribution over parameters u_{ij} that reflects our prior beliefs about smoothness of the objective function. We can write $\mathbf{\Sigma} = \mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}$, where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues, and the columns of $\mathbf{V}$ are the associated eigenvectors. We would like to place a prior distribution over the eigenvalues in $\mathbf{\Lambda}$, which are the length scales that control the function's smoothness in the directions contained in $\mathbf{V}$. The problem is that there is no obvious way of isolating these 'length scale components' when we work with $\mathbf{U}$. A similar problem is that there is no obvious way of specifying a prior distribution over covariance function rotations (angles of the eigenvectors in $\mathbf{V}$). For example, it is not clear what prior distribution over the elements of $\mathbf{U}$ should be used to encode the prior belief that all covariance function rotations are equally probable. Vivarelli and Williams do not encounter this problem as they use maximum likelihood methods to learn the parameters, rather than maximum *a posteriori* or fully Bayesian methods.

The following example illustrates the problem. A Gaussian prior distribution was placed over the elements of $\mathbf{U}$ for $D = 2$. In particular, $u_{11} \sim \mathcal{N}(0, 0.5^2)$, $u_{22} \sim \mathcal{N}(0, 0.5^2)$, and $u_{12} \sim \mathcal{N}(0, 0.5^2)$. 5×10^6 samples of $\mathbf{U}$ and hence $\mathbf{\Sigma}$ were generated by sampling from these prior distributions. For each sample, the eigendecomposition $\mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}}$ was calculated, and from this the angle of rotation of the principal eigenvector (the eigenvector with the largest eigenvalue) was found. The experiment was repeated with prior distributions set to $u_{11} \sim \mathcal{N}(0, 0.25^2)$, $u_{22} \sim \mathcal{N}(0, 0.25^2)$, and $u_{12} \sim \mathcal{N}(0, 0.75^2)$. Figure 7.2 shows how both prioritisations result in a non-uniform distribution of covariance function rotations, as measured by the rotation of the principal eigenvector. Overall, it is not clear what prior distribution over $\mathbf{U}$ corresponds to a *uniform* prior distribution of rotations.

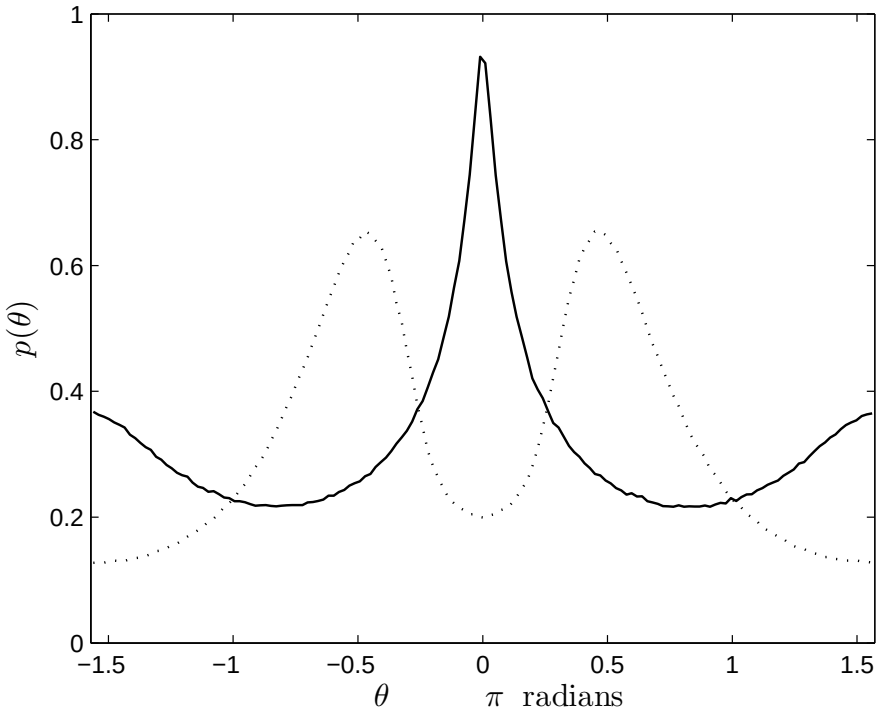

Figure 7.2: Prior probability of rotation angle for a positive definite matrix parameterised via the Cholesky decomposition, with Gaussian prior distributions over the elements of the Cholesky factors. Two cases are shown: the solid line shows the results when $u_{11} \sim \mathcal{N}(0, 0.5^2)$, $u_{22} \sim \mathcal{N}(0, 0.5^2)$, $u_{12} \sim \mathcal{N}(0, 0.5^2)$, and the dotted line is for $u_{11} \sim \mathcal{N}(0, 0.25^2)$, $u_{22} \sim \mathcal{N}(0, 0.25^2)$, $u_{12} \sim \mathcal{N}(0, 0.75^2)$.

7.3 Parameterisations based on Givens Angles

Any positive definite matrix of real numbers $\mathbf{\Sigma}$ can be decomposed into a set of eigenvector and eigenvalue matrices:

$$\mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}} \tag{7.4}$$

where $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$ (orthonormal) and $\mathbf{\Lambda}$ is a diagonal matrix of positive eigenvalues.

Following Paciorek [52], we proceed by parameterising the eigenvectors using Givens angles [21]. $\mathbf{V}$ can be parameterised as a product of Givens matrices:

$$\mathbf{V} = \prod_{i=1}^{D-1} \prod_{j=i+1}^{D} \mathbf{G}_{ij} \tag{7.5}$$

where

$$\mathbf{G}_{ij} = \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & & i & & j & \end{array} \\ \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ 0 & \cos(\rho_{ij}) & 0 & -\sin(\rho_{ij}) & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & \sin(\rho_{ij}) & 0 & \cos(\rho_{ij}) & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix} \end{array} \tag{7.6}$$

The parameter ρ_{ij} sets the angle of rotation of the plane defined by the i^{th} and j^{th} coordinate axes.

$\mathbf{\Lambda}$ can be parameterised quite simply as $\mathbf{\Lambda} = \mathrm{diag}(\exp(\lambda_1) \ldots \exp(\lambda_D))$, where the exponential function forces the eigenvalues to be positive. We could set a uniform prior distribution over rotations ρ_{ij} together with a prior distribution over the eigenvalues λ_i that reflects a belief about the smoothness of the problem.

The total number of parameters needed to fully parameterise the $D \times D$ matrix $\mathbf{\Sigma}$ is D eigenvalues plus $\frac{D(D-1)}{2}$ rotations, which equals $\frac{D(D+1)}{2}$, the same as the Cholesky decomposition parameterisation. Note this is the minimum number of parameters required to fully encode an arbitrary real and positive definite $\mathbf{\Sigma}$. For problems involving only a few dimensions, the total number of parameters required is manageable. However, as the dimensionality grows it comes as no surprise that the situation rapidly deteriorates; the number of parameters grows as $\mathcal{O}(D^2)$. Additionally, to learn the set of angles $\left\{\left\{\rho_{ij}\right\}_{i=1}^{D-1}\right\}_{j=i+1}^{D}$ efficiently, we would like to make use of gradient information. Finding the gradient of (7.5) requires application of the product rule. Although possible, the resulting expressions and implementation become cumbersome and computationally expensive as the dimensionality grows. Ideally, we would like to parameterise an orthonormal $\mathbf{V}$ directly as a single matrix, or as a sum of matrices. Unfortunately, it is not clear if this is possible.

7.4 Factor Analysis Parameterisation

To fully parameterise an arbitrary positive definite matrix $\mathbf{\Sigma}$ requires $\frac{D(D+1)}{2}$ parameters, which for very high dimensional problems may prove an unac-

ceptable cost. For example, encoding a $50D$ positive definite matrix requires 1275 parameters. If we wish to learn the values of the parameters by maximum likelihood methods, then we have to optimise the likelihood function over a 1275 dimensional space. Furthermore, if we have a relatively small data set then a huge number of hyperparameters seems excessive, and we might consider the resulting models too complex. In such cases we might prefer to use some intermediate parameterisation that allows $\mathbf{\Sigma}$ to vary somewhat from the identity matrix, but is not complex enough to encode all possible positive definite matrices.

One way to do this is with the factor analysis form $\mathbf{\Sigma} = \mathbf{R}\mathbf{R}^{\mathrm{T}} + \mathbf{\Lambda}$, where $\mathbf{R}$ is a $D \times k$ matrix ($D \geq k$), and $\mathbf{\Lambda}$ is a diagonal matrix with positive entries. The first component $\mathbf{R}\mathbf{R}^{\mathrm{T}}$, has a rank k and is capable of defining a k dimensional subspace over which the objective function varies significantly. $\mathbf{\Sigma}$ is made full rank by the addition of $\mathbf{\Lambda}$. This parameterisation uses $D(k+1)$ parameters, so for $k \ll D$ we get quite a reduction in the number of parameters required.

We could set prior distributions over $\mathbf{\Lambda}$ by setting the j^{th} element on the diagonal equal to $\exp(\lambda_j)$, and setting a Gaussian prior distribution over λ_j. Furthermore, we could set Gaussian prior distributions directly over the elements of $\mathbf{R}$. However, it is not clear how such a "prioritisation" corresponds to length scales in the problem space. In other words, if we wish to use the factor analysis form for $\mathbf{\Sigma}$, and wish to restrict the length scales to sensible values using a prior distribution, then it is not clear how to do this with Gaussian prior distributions on $\mathbf{R}$ and $\lambda_1 \ldots \lambda_D$.

7.5 Random Rotations

Instead of attempting to learn the orthonormal matrix of eigenvectors $\mathbf{V}$ in equation (7.5), we can try a few randomly generated orthonormal matrices, and use the one that has the greatest posterior probability after the remaining hyperparameters are learnt. For a data set $\mathbf{X}$ and $\mathbf{y}$ we use the following method:

(1) Generate a random orthonormal matrix $\mathbf{V}$

(2) Rotate the data to produce $\mathbf{Z} = \mathbf{V}\mathbf{X}$

(3) Find $\boldsymbol{\theta}_{\text{max}}$ that maximises $p(\boldsymbol{\theta}|\mathbf{Z}, \mathbf{y})$

(4) Use the $\mathbf{V}$ that has the greatest $p(\boldsymbol{\theta}_{\text{max}}|\mathbf{Z}, \mathbf{y})$

In step 1 we need to generate a random orthonormal matrix. A simple way of doing this is to generate a matrix $\mathbf{R}$ with normally distributed elements, and then find the set of eigenvectors of $\mathbf{R}^{\mathrm{T}}\mathbf{R}$. These eigenvectors $\{\mathbf{v}_1 \dots \mathbf{v}_D\}$ are mutually orthogonal and $|\mathbf{v}_i| = 1, \forall i$. The required random orthonormal matrix is $\mathbf{V} = \left[\mathbf{v}_1 | \dots | \mathbf{v}_D\right]$.

In step 3 we find the hyperparameters that have maximum posterior probability, given the rotated data. We use an axis-aligned covariance function, by just learning the length scales in each direction encoded in a diagonal matrix $\boldsymbol{\Lambda}$.

If

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\boldsymbol{\Lambda}(\mathbf{x}_i - \mathbf{x}_j)\right) \tag{7.7}$$

is a valid covariance function, then so is

$$k(\mathbf{V}\mathbf{x}_i, \mathbf{V}\mathbf{x}_j) = \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{V}^{\mathrm{T}}\boldsymbol{\Lambda}\mathbf{V}(\mathbf{x}_i - \mathbf{x}_j)\right) \tag{7.8}$$

So rotating the data before learning is equivalent to learning a rotated covariance function with principal axes determined by $\mathbf{V}$. The trade off is that we do not attempt to learn an optimal $\mathbf{V}$, but simply one that is the best of many, or at least better than the identity matrix corresponding to an axis aligned covariance function.

We can modify this method to attempt to learn the optimal $\mathbf{V}$. Instead of generating a random orthonormal matrix at step 1, we randomly perturb the current $\mathbf{V}$ and accept the perturbation if it results in greater posterior probability. A random perturbation that maintains the orthonormality of $\mathbf{V}$ can be achieved by multiplication by a rotation matrix consisting of small random angles. This rotation matrix can be generated using (7.5), with Givens angles selected from a normal distribution with a small standard deviation (e.g. 5°).

7.6 Summary

In this chapter, we have discussed parameterisations of squared-exponential covariance functions that are powerful enough to allow the covariance function to rotate and stretch in high dimensional space. The purpose of doing so is to enable Gaussian processes to model data with features that are not aligned to the coordinate axes. Some of these parameterisations will be further applied in later chapters.

Chapter 8

"Bayesian" Gaussian Processes

In this chapter, we examine Gaussian processes for regression using the methods of Bayesian inference. In earlier chapters, we decided on a single parameterised covariance function, and then found the most likely or most probable hyperparameters which were then used to make predictions. In the Bayesian methodology, we can avoid having to make predictions using just one covariance function, with a single set of best hyperparameters. Instead, we *marginalise* over all uncertainty - uncertainty in the covariance function, and uncertainty in the hyperparameters.

In a sense, one may consider regression with a single covariance function and optimised hyperparameters as Bayesian regression. In fact, Gaussian process regression is equivalent to Bayesian parametric regression using a generalised linear model with an infinite number of basis functions (e.g. see [55]). However, this method is not considered *fully* Bayesian, as it does not marginalise over uncertainty in the hyperparameters.

Overall, there are different methods that vary in the extent to which they adhere to the principles of Bayesian inference. At the simplest level, there is the example just described which makes use of a single covariance function with optimised hyperparameters. A more principled Bayesian method would use a single covariance function, but would marginalise across uncertainty in the hyperparameters. Strictly, however, we should express prior uncertainty about the form of the covariance function and then marginalise over this uncertainty *and* the uncertainty in the hyperparameters associated with the various forms that the covariance function might take. In this chapter, we

adopt an intermediate position where we optimise the hyperparameters of a set of chosen covariance functions, and then marginalise over the uncertainty within that chosen set.

Below, we briefly examine Bayesian prediction by marginalising over hyperparameters with a fixed covariance function, and then show how to marginalise over multiple covariance functions.

8.1 Marginalising over Hyperparameters

To carry out Bayesian prediction using Gaussian processes, we use the following integral which marginalises over the posterior uncertainty in the hyperparameters:

$$p(y_*|\mathbf{x}_*, \mathcal{D}, \mathcal{H}) = \int p(y_*|\mathbf{x}_*, \mathcal{D}, \boldsymbol{\theta}, \mathcal{H})\, p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{H}) d\boldsymbol{\theta} \tag{8.1}$$

where $\mathcal{H}$ represents the assumed covariance function parameterised by $\boldsymbol{\theta}$.

However, in most cases this integral is intractable so we are forced to make an approximation. If we have a lot of data relative to the number of hyperparameters, then we usually expect the posterior density $p(\boldsymbol{\theta}|\mathcal{D})$ to be concentrated around a single value $\boldsymbol{\theta}_{MAP}$. If so, then we can make predictions using

$$p(y_*|\mathbf{x}_*, \mathcal{D}, \mathcal{H}) \;\approx\; p(y_*|\mathbf{x}_*, \mathcal{D}, \boldsymbol{\theta}_{MAP}, \mathcal{H}) \tag{8.2}$$

If we have relatively few data points, or many hyperparameters, then this approximation is not expected to be as good. In such cases, the posterior probability may be spread more diffusely over $\boldsymbol{\theta}$, or there may even be multiple isolated modes separated by regions of very small probability. If so, then we might expect improvement by making a better approximation to the integral in (8.1).

The standard way to approximate the integral is to use numerical Monte Carlo methods [45, 41, 34]. More specifically, we can generate samples $\boldsymbol{\theta}_1 \ldots \boldsymbol{\theta}_M$ from the posterior distribution with density $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{H})$ and then approximate the integral using:

$$p(y_*|\mathbf{x}_*, \mathcal{D}, \mathcal{H}) \;\approx\; \frac{1}{M} \sum_{m=1}^{M} p(y_*|\mathbf{x}_*, \mathcal{D}, \boldsymbol{\theta}_m, \mathcal{H}) \tag{8.3}$$

Monte Carlo sampling of the posterior distribution has been demonstrated and discussed by a number of authors [86, 57, 47]. Here, we assume that the posterior density over hyperparameters is sufficiently concentrated around $\boldsymbol{\theta}_{MAP}$ such that equation (8.2) is a good approximation. Therefore, in the remainder of this chapter we focus attention on marginalising over differing covariance functions.

8.2 Marginalising over Covariance Functions

If we attempt to model data with a Gaussian process defined by a single, relatively inflexible covariance function, we may find that the predictive performance is rather poor. In such cases, we might say that the model is too simple or does not have enough explanatory power to capture whatever features underly the observations.

Alternatively, if we model relatively simple data with a Gaussian process defined by a complex covariance function, then we might expect good predictive performance on the training data but poor generalisation performance. In such cases, the Gaussian process covariance function has enough complexity and flexibility to fit the data very well. However, a side effect of this is the potential to overfit.

The performance we want from our model lies somewhere in between. That is, we want to use a model that is simple enough to generalise well, but complex enough to capture underlying trends. The remainder of this chapter will examine methods to do this for Gaussian process models.

8.2.1 Model Comparison

In the Bayesian methodology, one way to marginalise over many Gaussian process models (each defined by a covariance function and hyperparameters), is to weight each model according to its posterior probability. In this way, if a simple model models the data sufficiently, it will be that model which is predominantly used to make a prediction. If the simple model accounts poorly for the data, then a more complex model that provides a better fit will be favoured.

Consider the situation where we have multiple models $\mathcal{H}_1 \dots \mathcal{H}_M$, each consisting of a covariance function and a set of hyperparameters. Initially, we have prior uncertainty about the probability of each of model. The predictive distribution for a test point $\mathbf{x}_*$ is a sum over models [34]

$$p(y_*|\mathbf{x}_*, \mathcal{D}) = \sum_{i=1}^{M} p(y_*|\mathcal{D}, \mathcal{H}_i)p(\mathcal{H}_i|\mathcal{D}) \tag{8.4}$$

which consists of a weighted sum of the predictive distributions from each model, weighted by the posterior probability $p(\mathcal{H}_i|\mathcal{D})$ of each. By Bayes' Theorem, the posterior probability of the i^{th} model is related to the model *evidence*, $p(\mathcal{D}|\mathcal{H}_i)$ as follows

$$p(\mathcal{H}_i|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{H}_i)p(\mathcal{H}_i)}{p(\mathcal{D})} \tag{8.5}$$

If we have a prior belief that each model is equally plausible, $p(\mathcal{H}_i) = p(\mathcal{H}_j) = \frac{1}{M}$, then the posterior probability of each model is simply proportional to the evidence for that model. In this thesis, for simplicity, it is assumed that all models have equal prior probability, so the evidence $p(\mathcal{D}|\mathcal{H}_i)$ is used to weight models rather than the posterior probability $p(\mathcal{H}_i|\mathcal{D})$.

In a Gaussian process regression context, this method allows us to avoid having to select *a priori* a single covariance function that suits a particular problem. Instead, we can use a range of covariance functions and let the evidence for each determine their relative importance.

In cases where we have just two models (or covariance functions), we might like to use the evidence to compare the models to decide which is best. In this case we can calculate the Bayes factor $B_{1/2}$ which is equal to the ratio of the evidence for model 1 and 2, $B_{1/2} = p(\mathcal{D}|\mathcal{H}_1)/p(\mathcal{D}|\mathcal{H}_2)$. Jeffreys' scale of evidence for Bayes' factors [26] can then be used to interpret the result. This scale is presented in the figure 8.1.

8.2.2 Evaluating the Evidence

The hard part of model comparison is evaluation of the evidence. The evidence, or marginal likelihood, for model $\mathcal{H}$ is equivalent to the normalising constant of the posterior density, $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{H})$:

$$p(\mathcal{D}|\mathcal{H}) = \int p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{H})p(\boldsymbol{\theta}|\mathcal{H})d\boldsymbol{\theta} \tag{8.6}$$

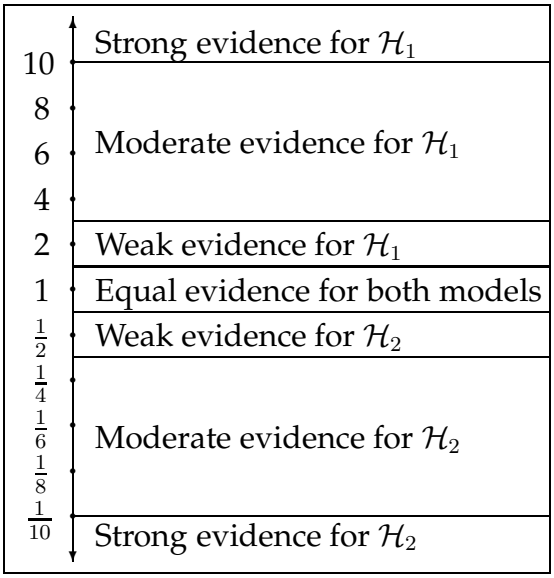

Figure 8.1: Jeffrey's scale of evidence for Bayes factors

As with the predictive integral in equation (8.1), this integral cannot be calculated directly for Gaussian processes. Instead, we must use numerical or approximate methods. MacKay [34] describes how the Laplace approximation can be used if the posterior distribution is well approximated by a Gaussian. Briefly, the Laplace method approximates the likelihood-prior product $p(\mathcal{D}|\boldsymbol{\theta},\mathcal{H})p(\boldsymbol{\theta}|\mathcal{H})$ as a Gaussian centred at $\boldsymbol{\theta}_{MAP}$ with covariance matrix $\boldsymbol{\Sigma}$, where

$$\boldsymbol{\Sigma}^{-1} = -\nabla\nabla \ln p(\boldsymbol{\theta}|\mathcal{D},\mathcal{H})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{MAP}} \tag{8.7}$$

The evidence is then approximated by integrating the Gaussian approximation analytically to give:

$$p(\mathcal{D}|\mathcal{H}) \;\approx\; p(\mathcal{D}|\boldsymbol{\theta}_{MAP},\mathcal{H})\, p(\boldsymbol{\theta}_{MAP}|\mathcal{H})(2\pi)^{\frac{D}{2}} \det(\boldsymbol{\Sigma})^{\frac{1}{2}} \tag{8.8}$$

For a Gaussian process (with covariance matrix C), we can form the approximation by noting that:

$$-\nabla\nabla \ln p(\boldsymbol{\theta}|\mathcal{D},\mathcal{H}) = -\nabla\nabla \ln p(\mathcal{D}|\boldsymbol{\theta},\mathcal{H}) - \nabla\nabla \ln p(\boldsymbol{\theta}|\mathcal{H}) \tag{8.9}$$

and

$$\begin{aligned}\frac{\partial^2 \ln p(\mathcal{D}|\boldsymbol{\theta},\mathcal{H})}{\partial\theta_i\partial\theta_j} &= \tfrac{1}{2}\,\mathrm{Tr}\left(\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta_j}\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta_i} \;-\; \mathbf{C}^{-1}\frac{\partial^2\mathbf{C}}{\partial\theta_i\partial\theta_j}\right)\\ &+ \tfrac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{C}^{-1}\left(\frac{\partial^2\mathbf{C}}{\partial\theta_i\partial\theta_j} - 2\frac{\partial\mathbf{C}}{\partial\theta_i}\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta_j}\right)\mathbf{C}^{-1}\mathbf{y}\end{aligned} \tag{8.10}$$

For small amounts of data and complex models with a large number of hyperparameters, this may not be an effective method. Informally, it is expected that the posterior distribution will resemble a Gaussian only when the ratio of data to hyperparameters is high. As a simple example, consider a $2D$ isotropic, noiseless GP model with a single hyperparameter r, and covariance function:

$$\mathrm{cov}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\tfrac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)\exp(-2r)) + 0.05^2\delta_{ij} \tag{8.11}$$

Two sets of data were generated: $\mathcal{D}_{18} = \{(\mathbf{x}_1, y_1)\ldots(\mathbf{x}_{18}, y_{18})\}$ and $\mathcal{D}_6 = \{(\mathbf{x}_1, y_1)\ldots(\mathbf{x}_6, y_6)\}$, such that $\mathcal{D}_6 \subset \mathcal{D}_{18}$. The input points $\mathbf{x}_1 \ldots \mathbf{x}_{18}$ were sampled from a bivariate normal distribution, and the i^{th} output value was set to $y_i = \exp(-\frac{1}{2}\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.05^2)$. For each data set, a Gaussian process model using the above covariance function was built by finding the maximum *a posteriori* value or r given a prior distribution $p(r) \sim \mathcal{N}(0, 1)$. For each model, the Laplace approximation to the evidence was calculated as described above.

Table 8.1 shows how close the Laplace approximation is to the true evidence for each model. With $N = 6$ data points, the approximation is quite inaccurate, while with $N = 18$ data points, it is very good. Therefore, at least for this simple case, the Laplace approximation is better with more data, as expected.

trial	$N = 6$	$N = 18$
1	0.82	0.94
2	0.86	0.98
3	0.88	1.02
4	0.85	0.99
5	0.83	1.01

Table 8.1: Accuracy of the Laplace method for estimating the model evidence. The table values are equal to the evidence estimate divided by the true evidence, so a value close to 1 indicates a good approximation. The true evidence was estimated by brute force numerical integration.

Alternatively, one might attempt to calculate the evidence using MCMC meth-

ods. A Monte Carlo approximation of the evidence for model $\mathcal{H}$ is

$$p(\mathcal{D}|\mathcal{H}) \approx \frac{1}{R}\sum_{r=1}^{R} p(\mathcal{D}|\boldsymbol{\theta}_r, \mathcal{H}) \tag{8.12}$$

where $\boldsymbol{\theta}_r$ is the r^{th} sample drawn from the prior distribution over hyperparameters, defined by the density $p(\boldsymbol{\theta}|\mathcal{H})$.

This looks promising initially, and may work well in cases with small amounts of data, or few hyperparameters. However, if there is a lot of data $p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{H})$ will be concentrated around $\boldsymbol{\theta}_{MAP}$. Furthermore, if there are a lot of hyperparameters or the prior distribution is quite uninformative, then $p(\boldsymbol{\theta}|\mathcal{H})$ will be spread quite thinly across a high dimensional space. The overall effect is such that a sample drawn from the prior distribution with density $p(\boldsymbol{\theta}|\mathcal{H})$ is very unlikely to fall within the typical set of $p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{H})$. The resulting evidence approximation will therefore be poor unless we take a very large number of samples. An illustration of this problem is shown in figure 8.2.

8.2.3 Annealed Importance Sampling

If calculating the evidence for data-sparse, hyperparameter-rich problems is non-trivial, how might we proceed? The remaining option would be to resort to numerical methods that are designed to cope with the problems discussed above. One such method is "Annealed Importance Sampling" introduced by Neal [49].

Consider a (possibly unnormalised) probability distribution $p(x)$ over a quantity x, and some function $a(x)$ of which we wish to estimate the expected value

$$E_p[a(x)] = \frac{\int a(x)p(x)dx}{\int p(x)dx} \tag{8.13}$$

Importance sampling (e.g. [34]) is a simple method that provides such an estimate. It is not a method for generating a set of equally weighted, independent samples from $p(x)$, like other Monte Carlo methods such as the Metropolis-Hastings, or Gibbs sampling algorithms.

Importance sampling requires the use of a sampler density $q(x)$ that has non-zero value whenever $p(x)$ is non-zero. We proceed by generating a set of n independent samples from $q(x)$, and for each sample $x^{(i)}$ calculating a weight

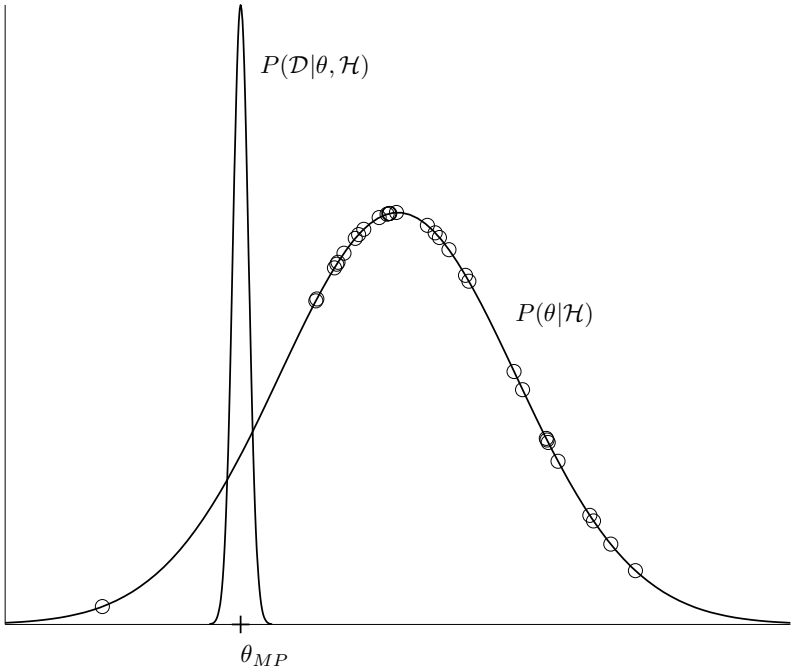

Figure 8.2: Illustration of a potential problem in calculating a MC approximation to the model evidence. The prior density $p(\boldsymbol{\theta}|\mathcal{H})$ has relatively high variance compared to the likelihood function, $p(\mathcal{D}|\boldsymbol{\theta},\mathcal{H})$ which is concentrated around $\boldsymbol{\theta}_{MAP}$ and is shown unnormalised. Samples from the prior distribution are shown as circles. Note that even in this one dimensional case, none of the samples fall within the typical set of the likelihood function, and so the approximation in (8.12) will be poor.

$w^{(i)} = p(x^{(i)})/q(x^{(i)})$. The estimate of $E[a(x)]$ is then:

$$\hat{a} = \frac{\sum_{i=1}^{n} w^{(i)} a(x^{(i)})}{\sum_{i=1}^{n} w^{(i)}} \tag{8.14}$$

If we know, or can calculate the normalisation for $q(x)$, then importance sampling also provides an estimate, $\hat{Z}_p$ of the normalisation constant $Z_p = \int p(x)dx$:

$$\hat{Z}_p = \frac{1}{n} \sum_{i=1}^{n} w^{(i)} \int q(x)dx \tag{8.15}$$

The accuracy of the estimates depends on how closely the sampler density $q(x)$ matches the density of interest $p(x)$. The degree of closeness between $q(x)$ and $p(x)$ can be roughly measured with the ratio $w^{rat} = w^{max}/w^{min}$, where w^{min} and w^{max} are the minimum and maximum weights. If $q(x) = p(x)$, then

$w^{rat} = 1$, and the estimates of $E[a(x)]$ and Z_P will be very good. If $q(x)$ is very similar to $p(x)$ (in shape, ignoring normalisation) then w_{rat} is close to 1, and the estimates will be good. If $q(x)$ is very different to $p(x)$ then $w^{rat} \gg 1$ and the estimates will be poor. Alternatively, the standard error of the normalisation estimate $\hat{Z}_p$ is found by dividing the sample variance of w by n. So, if the sample variance of w is high, then the expected error of our estimate is relatively high. By keeping w^{rat} low, we expect to reduce the variance of w.

As the problem dimensionality increases, it becomes harder to specify a sampler density $q(x)$ that closely matches $p(x)$. In this situation, w^{rat} will be large, and our importance sampling estimates will be dominated by a few large weights.

Annealed Importance Sampling [49] is a method designed to help overcome the problem of having to find a sampler density that closely resembles the density of interest. This method works by setting up a sequence of m possibly unnormalised densities, $q_0(x) \dots q_{m-1}(x)$ where each successive density is 'closer' to $p(x)$ than the previous. One possible sequence is as follows:

$$q_j(x) = q_0(x)^{(1-\beta_j)} p(x)^{\beta_j} \tag{8.16}$$

where β_j is defined by the annealing schedule, $0 = \beta_0 < \beta_1 < \dots \beta_j < \dots < \beta_m = 1$.

We start by generating n independent samples from $q_0(x)$ and then generate successive sample sets from $q_1(x), \dots q_j(x) \dots q_m(x)$. Let $x_j^{(i)}$ be the i^{th} sample generated from $q_j(x)$. Since in general we cannot sample from $q_j(x)$, we resort to some other sampling method. For simplicity, assume the Metropolis algorithm with spherical Gaussian proposal densities. At iteration j, we generate n samples from $q_j(x)$ using Metropolis sampling[1] with the n samples from iteration $j-1$ as starting points for the Markov chains. In other words, we maintain a set of n independent Markov chains, each corresponding to a separate independent annealing run.

For each sample we calculate an importance weight, with the i^{th} weight as follows:

$$w^{(i)} = \frac{q_1(x_0^{(i)})}{q_0(x_0^{(i)})} \frac{q_2(x_1^{(i)})}{q_1(x_1^{(i)})} \dots \frac{q_m(x_{m-1}^{(i)})}{q_{m-1}(x_{m-1}^{(i)})} \frac{p(x_m^{(i)})}{q_m(x_m^{(i)})} \tag{8.17}$$

[1] note that $q_j(x)$ may not be normalised, so we are actually generating samples from the distribution that has a probability density function that is proportional to $q_j(x)$.

We then calculate $\hat{a}$ and $\hat{Z}_p$ as in equations 8.14 and 8.15. (see [49] for proof).

Neal [49] suggests using the prior density $p(\boldsymbol{\theta}|\mathcal{H})$ as a starting sampler density, given that it is usually has a form which is straightforward to generate independent samples from. However, if this prior density is vague then many annealing runs may be required before $q_j(\boldsymbol{\theta})$ comes close to the posterior density $p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{H})$. In this case it may be useful to use an initial density $q_0(\boldsymbol{\theta})$ that more closely resembles the posterior density. An example of such a density is the Gaussian density that approximates the posterior density according to the Laplace method described above. Thus, the evidence evaluation might start with the Laplace approximation, and then improve upon this with annealed importance sampling.

8.2.4 An Heuristic Annealing Schedule

In this section we introduce an heuristic method for automatically building an annealing schedule.

One difficulty encountered when applying annealed importance sampling is specifying an annealing schedule. Intuitively, we want a schedule that results in $q_{j+1}(x)$ that closely resembles $q_j(x)$. If successive densities are very dissimilar, then w^{rat} for this iteration will be large. If successive densities are very similar, then w^{rat} will be close to 1. Following is a description of an heuristic to automatically build a schedule based on a choice for a maximum tolerable w^{rat} at any given iteration; let this limit be w^{rat}_{max}.

We start with $\beta_0 = 0$, which forces sampling to start from $q_0(x)$. Then, for any iteration we calculate:

$$g^{(i)} = \frac{p(x^{(i)})}{q_0(x^{(i)})} \tag{8.18}$$

for all samples $x^{(1)} \ldots x^{(n)}$ and then find

$$g^{rat} = \frac{\max(g^{(1)} \ldots g^{(n)})}{\min(g^{(1)} \ldots g^{(n)})} \tag{8.19}$$

and set

$$\beta_{j+1} = \min(1, \beta_j + \frac{\log(w^{rat}_{max})}{\log(g^{rat})}) \tag{8.20}$$

and stop when $\beta_{j+1} = 1$.

By setting w_{max}^{rat} to a low value we make slow but accurate progress because $q_{j+1}(x)$ is not much different from $q_j(x)$. For higher values we make faster but less accurate progress. This heuristic replaces the need to define an annealing schedule with the selection of a single constant, w_{max}^{rat} that trades off speed versus accuracy.

Let us now return to the notion of 'closeness' between two probability densities. One way of measuring such closeness is with the KullbackLeibler divergence, or relative entropy[34]:

$$\tilde{F} = \int q(x) \ln \frac{q(x)}{p(x)} \, dx \tag{8.21}$$

which is equal to the expected value of $\ln \frac{q(x)}{p(x)}$ with respect to the density $q(x)$. The relative entropy is non-negative, and equals zero if and only if $q(x)$ equals $p(x)$. Note that for a given sample $x^{(i)}$ from $q(x)$ the ratio $\frac{q(x^{(i)})}{p(x^{(i)})}$ is equal to the inverse of the weight were that sample to be used in importance sampling of $p(x)$ with $q(x)$ as a sampler density.

As an example, consider the normalised Gaussian densities

$$q(x) = (2\pi)^{-\frac{1}{2}} \exp(-\tfrac{1}{2}x^2) \tag{8.22}$$

$$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp(-\tfrac{1}{2}\frac{x^2}{\sigma^2}) \tag{8.23}$$

We take n samples from $q(x)$, and for various σ from 0.75 to 1.5 we calculate the relative entropy and the value of w^{rat}. The mean results after repeating this process 100 times are shown in figure 8.3. Clearly, the relative entropy increases with w^{rat}. That is, if w^{rat} is large, then the relative entropy between $p(x)$ and $q(x)$ is large. Although this result has been generated by using Gaussian densities with the same mean, it provides support to the notion that w^{rat} provides an approximate measure of closeness between two densities.

One may ask why we don't just use a fixed change in relative entropy to find β_{j+1} given β_j. There is a problem in doing so with unnormalised densities. If $q(x)$ is normalised, but $p(x)$ is not, then the relative entropy will not be minimised even if $q(x) \propto p(x)$ (in which case w^{rat} would equal 1). For example, if

$$q(x) = (2\pi)^{-\frac{1}{2}} \exp(-\tfrac{1}{2}x^2) \tag{8.24}$$

$$p(x) = \exp(-\tfrac{1}{2}x^2) \tag{8.25}$$

then $q(x) \propto p(x)$ but $\tilde{F} = \frac{1}{2}\log 2\pi \neq 0$ indicating that $q(x)$ and $p(x)$ are dissimilar even though $w^{rat} = 1$.

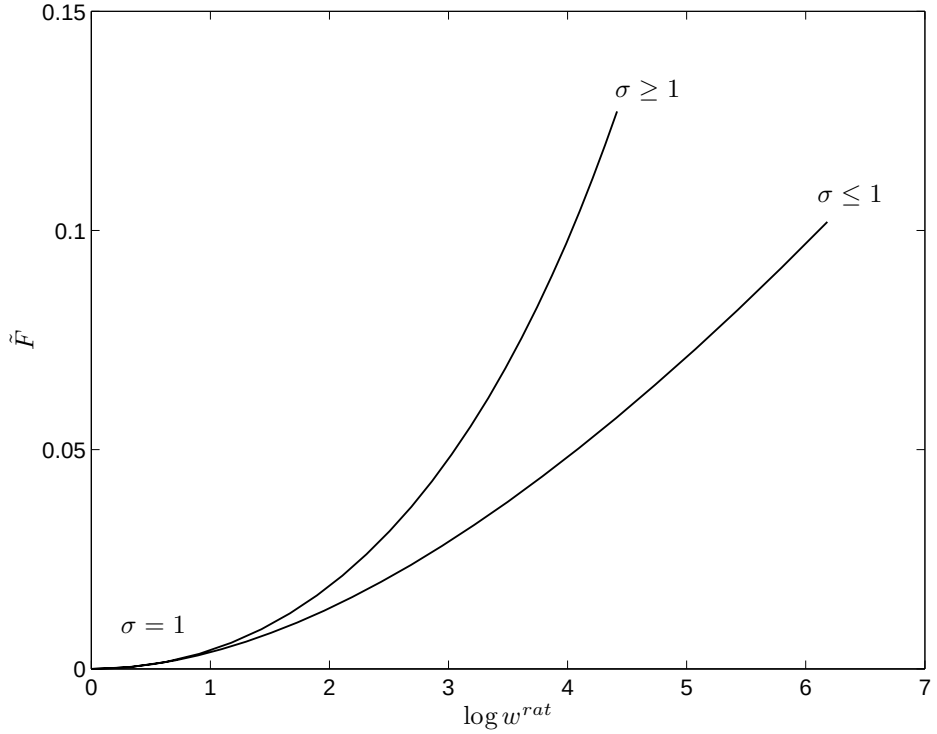

Figure 8.3: Mean relative entropy, $\tilde{F}$ against mean $\log w^{rat}$. The upper line shows the results when $\sigma \geq 1$. The lower plot is for $\sigma \leq 1$. The lines converge at the origin when $\sigma = 1$, meaning $\tilde{F} = 0$ and $w^{rat} = 1$

8.2.5 Sequential Evidence Evaluation

Annealed importance sampling requires the specification of a path of densities from an initial normalised density $q_0(x)$ to the final density $p(x)$ which is to be normalised. Consider now the case of a model with parameters $\boldsymbol{\theta}$ and n observed data points $\mathcal{D}$, where we wish to calculate the evidence $p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$. Note that the integrand can be factored as follows:

$$p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = p(\mathcal{D}_n|\mathcal{D}_1 \ldots \mathcal{D}_{n-1}, \boldsymbol{\theta})p(\mathcal{D}_{n-1}|\mathcal{D}_1 \ldots \mathcal{D}_{n-2}, \boldsymbol{\theta}) \ldots p(\mathcal{D}_1|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{8.26}$$

The factorisation suggests an online, or sequential method for updating the evidence when new data arrives.

Starting with the first data point, we estimate the model evidence $p(\mathcal{D}_1)$ using annealed importance sampling, with the $p(\boldsymbol{\theta})$ as the initial density and $p(\mathcal{D}_1|\boldsymbol{\theta})p(\boldsymbol{\theta})$ as the integrand; call the output of this calculation Z_1, which is an estimate of the evidence $p(\mathcal{D}_1)$. When the next datum arrives, $\frac{1}{Z_1}p(\mathcal{D}_1|\boldsymbol{\theta})p(\boldsymbol{\theta})$ can be used as the initial density and $Z_1 p(\mathcal{D}_2|\mathcal{D}_1\boldsymbol{\theta})p(\boldsymbol{\theta})$ becomes the integrand. Annealed importance sampling gives an output of Z_2, which is an estimate of $p(\mathcal{D}_1\mathcal{D}_2)$.

In general, after datum k arrives, we update the estimate of the evidence via annealed importance sampling. To do so, $\frac{1}{Z_{k-1}}p(\mathcal{D}_1 \ldots \mathcal{D}_{k-1}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ is used as the initial density and $Z_{k-1}p(\mathcal{D}_1 \ldots \mathcal{D}_k|\boldsymbol{\theta})p(\boldsymbol{\theta})$ is the integrand. This requires a set of importance samples from $\frac{1}{Z_{k-1}}p(\mathcal{D}_1 \ldots \mathcal{D}_{k-1}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. These samples and corresponding importance weights can be obtained by continuing to simulate the set of Markov chains available from the previous iteration.

This method may have problems if for some reason $p(\mathcal{D}_1 \ldots \mathcal{D}_{k+1}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ is very similar to $p(\mathcal{D}_1 \ldots \mathcal{D}_k|\boldsymbol{\theta})p(\boldsymbol{\theta})$ apart from having an extra isolated mode. A fictitious case is depicted in figure 8.4. If samples are taken from the initial sampler density $p(\mathcal{D}_1|\theta)p(\theta)$, it is very unlikely that any of these will "land" on the leftmost mode of the target density $p(\mathcal{D}_1, \mathcal{D}_2|\theta)p(\theta)$. In this contrived case, the estimate of the normalisation constant of the target density will be half the true value. The only way to overcome this problem would be to use many intermediate densities and simulate the Markov chain samplers for a very long time to allow transitions to occur to the leftmost mode.

If we wish to allow for the possibility of new isolated modes that appear upon observation of new data, then we should not use the sequential method. Instead, we should resort to using the prior density $p(\boldsymbol{\theta})$ as the initial sampler. Since the prior density is diffuse and data independent, we should expect to capture information from all modes even if isolated. The disadvantage of this is that it seems to take more time to anneal from $p(\boldsymbol{\theta})$ to $p(\mathcal{D}_k|\boldsymbol{\theta})p(\boldsymbol{\theta})$ than it does from $p(\mathcal{D}_{k-1}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ to $p(\mathcal{D}_k|\boldsymbol{\theta})p(\boldsymbol{\theta})$. In many cases, $p(\mathcal{D}_{k-1}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ is very close to $p(\mathcal{D}_k|\boldsymbol{\theta})p(\boldsymbol{\theta})$, and the sequential iteration is very quick.

8.2.6 Model Comparison Examples

Here, we examine two toy examples to demonstrate the use of annealed importance sampling and sequential annealed importance sampling to calculate

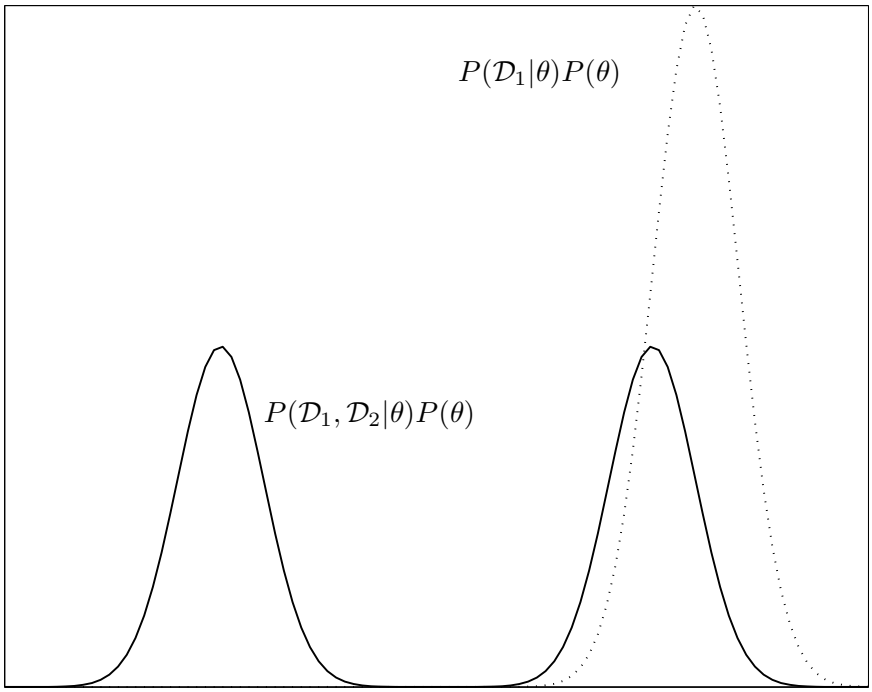

Figure 8.4: Illustration of a potential problem with sequential evidence evaluation with annealed importance sampling. The dotted line shows $p(\mathcal{D}_1|\theta)p(\theta)$ and the solid line $p(\mathcal{D}_1, \mathcal{D}_2|\theta)p(\theta)$, which consists of two isolated modes.

---

the evidence and compare models.

Firstly, consider two GP models - one simple and isotropic $\mathcal{H}_s$, and the other more complex and anisotropic $\mathcal{H}_c$. The models are defined by the following covariance functions:

$$k_s(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\tfrac{1}{2}\sum_{d=1}^{2}\frac{(x_i^{(d)} - x_j^{(d)})^2}{\exp(2r_0)}\right) \tag{8.27}$$

$$k_c(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\tfrac{1}{2}\sum_{d=1}^{2}\frac{(x_i^{(d)} - x_j^{(d)})^2}{\exp(2r_d)}\right) \tag{8.28}$$

where r_0 and $\mathbf{r} = \{r_1, r_2\}$ are hyperparameters controlling smoothness. $\mathcal{H}_s$ has a single hyperparameter controlling smoothness in all directions (spherical covariance), while $\mathcal{H}_c$ has two hyperparameters controlling smoothness in each direction aligned with the axes (axis-aligned elliptical covariance).

16 input points from $\mathbb{R}^2$ were generated by sampling from a bivariate normal distribution. Two data sets were generated by sampling the two Gaussian processes with the covariance functions above (with $r_0 = \log(0.5), r_1 =$

$\log(0.1), r_2 = \log(1.0)$) at the 16 input points to give a simple data set $\mathcal{D}_s$ and a complex data set $\mathcal{D}_c$. Gaussian prior distributions were set over r_0 and $\mathbf{r}$ such that $r_i \sim \mathcal{N}(\log(0.5), 1)$, and their most probable values were found by maximising the posterior density. For each data set, the Laplace method was used to find a Gaussian approximation to the posterior distribution. These approximations were used as the initial sampler densities for generating samples with annealed importance sampling with $w_{max}^{rat} = 1.5$. The importance weights were used to estimate the evidence for each model, over both data sets. The experiment was repeated 5 times, and Bayes' factor was calculated for each trial. The results for the simple data are shown in table 8.2. The evidence is mostly strongly in favour of $\mathcal{H}_s$, with trial 4 giving evidence moderately in favour.

trial	AIS	Laplace	True
1	14.0	14.4	14.1
2	15.0	15.9	15.4
3	13.5	14.4	13.9
4	3.9	4.0	3.9
5	13.4	13.7	13.5

Table 8.2: Bayes' Factor in comparison of $\mathcal{H}_s$ and $\mathcal{H}_c$, for the **simple** dataset. Values greater than 1 are in favour of $\mathcal{H}_s$

The results for the complex data shown in table 8.3. The evidence is extremely strong in favour of $\mathcal{H}_c$. The table shows Bayes' factors calculated using annealed importance sampling, the Laplace approximation and brute force numerical integration (true value). We see that all the results are comparable, indicating that the Laplace approximation is sufficient to calculate the evidence in this example.

Our second model comparison example consists of a simple model, $\mathcal{H}_s$, with an axis aligned covariance function defined by (7.1) (page 91), and complex model, $\mathcal{H}_c$, with a rotated covariance function defined by (7.2) where $\mathbf{\Sigma}$ is parameterised via Givens angles (section 7.3). For both models, equal prior distributions were placed over length scales, vertical scales, and noise. The complex model had a uniform prior distribution over Givens angles $p(\rho_{ij}) = \frac{1}{2\pi}$ for $-\pi \leq \rho_{ij} \leq \pi$. Two sets of $2D$ data were generated by stretching and

trial	AIS	Laplace	True
1	6.39×10^{-12}	6.36×10^{-12}	6.37×10^{-12}
2	5.66×10^{-14}	5.63×10^{-14}	5.52×10^{-14}
3	8.93×10^{-13}	9.24×10^{-13}	9.00×10^{-13}
4	6.44×10^{-12}	6.54×10^{-12}	6.42×10^{-12}
5	1.40×10^{-13}	1.42×10^{-13}	1.37×10^{-13}

Table 8.3: Bayes' Factor in comparison of $\mathcal{H}_s$ and $\mathcal{H}_c$, for the **complex** dataset. Values less than 1 are in favour of $\mathcal{H}_c$

---

rotating a set $\mathbf{X}_s$ of 16 samples from a spherical Gaussian distribution with variance $\mathbf{I}$. The first data set, $\mathbf{X}_a$ was generated by multiplying each element in $\mathbf{X}_s$ by a diagonal matrix $\mathbf{E} = \mathrm{diag}(0.1^2, 1.0^2)$. The second data set, $\mathbf{X}_r$ was generated by multiplying the elements in $\mathbf{X}_s$ by a matrix $\mathbf{V}^{\mathrm{T}}\mathbf{E}\mathbf{V}$ with $\mathbf{V}$ an orthonormal rotation matrix with Givens angles equal to $\frac{\pi}{4}$. The result is that $\mathbf{X}_a$ is Gaussian distributed with covariance matrix $\mathbf{\Lambda} = \mathbf{E}^2$, and $\mathbf{X}_r$ has covariance matrix $\mathbf{\Sigma} = \mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}$. In other words, the first set looks like samples from an axis aligned elliptical Gaussian distribution, and the second set looks like samples from a rotated elliptical Gaussian distribution; the second set is simply a rotation of the first set.

Sequential annealed importance sampling was run on both data sets, for both the simple and complex models. 100 annealing runs were made in parallel, using a $w_{max}^{rat} = 2$ heuristic to design the annealing schedule. The evidence for each model was recorded for all densities $p(\boldsymbol{\theta})$, $p(\mathcal{D}_1|\boldsymbol{\theta})p(\boldsymbol{\theta})$, $\ldots$ $p(\mathcal{D}_1, \ldots \mathcal{D}_{16}|\boldsymbol{\theta})p(\boldsymbol{\theta})$, and the posterior probability that the model was $\mathcal{H}_s$ was found from:

$$p(\mathcal{H} = \mathcal{H}_s) = \frac{p(\mathcal{H}_s|\mathcal{D})}{p(\mathcal{H}_s|\mathcal{D}) + p(\mathcal{H}_c|\mathcal{D})} \tag{8.29}$$

$$= \frac{p(\mathcal{D}|\mathcal{H}_s)}{p(\mathcal{D}|\mathcal{H}_s) + p(\mathcal{D}|\mathcal{H}_c)} \tag{8.30}$$

given equal prior probability of each model. This process was repeated 5 times and the results are shown in figure 8.5. It is clear that once enough data has been observed that the evidence clearly favours the simple model for the axis aligned data set. Similarly, the evidence for the complex model is strong

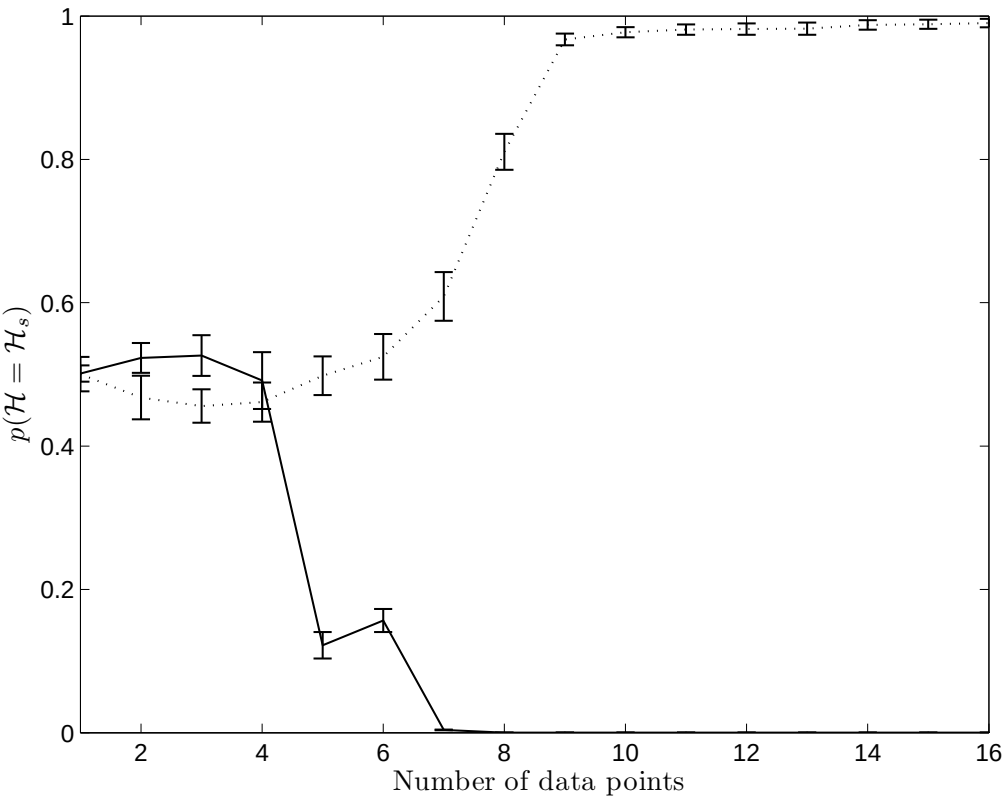

Figure 8.5: Model comparison using sequential annealed importance sampling with a simple and complex GP model (see text). The posterior probability that the model is simple is shown as each point was sequentially added to the evaluation. The experiment was repeated 5 times and shown is the mean with 1σ error bars for axis aligned (dotted) and rotated data (solid).

---

for the rotated data set. Furthermore, the extent of the 1σ error bars indicate that, for this example at least, that sequential annealed importance sampling is a reasonably precise tool for model comparison.

8.3 Summary

In this chapter, we have discussed and demonstrated methods to calculate the evidence for Gaussian process models with differing covariance functions. We have described a new method - sequential annealed importance sampling - which can be used to produce an evidence estimate that is updated with the addition of new data. In later chapters, we demonstrate how model com-

parisons done in this way can be applied to enhance the performance of the Gaussian process optimisation algorithm.

Chapter 9

Gaussian Processes for Optimisation

9.1 Introduction

Until now, we have discussed and demonstrated basic Gaussian processes for regression and introduced some enhancements allowing the construction of more sophisticated models. In this chapter, we shift focus and examine the application of some of these methods. In particular, we discuss and demonstrate the use of Gaussian processes for the efficient solution of optimisation problems.

9.2 Response Surface Methodology

One potentially efficient way to perform optimisation is to use the data collected so far to build a predictive model, and use that model to select subsequent search points. In an optimisation context, this model is often referred to as a *response surface*. This method is potentially efficient if data collection is expensive relative to the cost of building and searching a response surface. In many such cases, it can be beneficial to use relatively cheap computing resources to build and search a response surface, rather than incur large costs by directly searching in the problem space.

In the response surface methodology (see [44] for an introduction), we construct a response surface and search that surface for likely candidate points, measured according to some criterion. Jones [27] provides a summary of many such methods and discusses their relative merits. As a simple example, consider a noiseless optimisation problem where we currently have a set of N samples and proceed with the following method:

(1) Fit a basis function model to the data by minimising the sum of squared errors.

(2) Find an optimum point of the model and call this point $\mathbf{x}_{\text{new}}$

(3) Sample the problem at $\mathbf{x}_{\text{new}}$, and add the result to the current data set.

(4) Repeat until satisfied or until is no satisfactory progress is being made.

Unfortunately, this simple method is not suitable as a general purpose optimisation algorithm. In may cases it rapidly converges to suboptimal solutions. The reason for this is that the model can be easily deceived and overly commit to an erroneous belief about the structure of the underlying problem. If this happens, the model will continually exploit this belief and continue to gather sub-optimal data.

For more general purposes, we require more sophisticated search methods that are capable of exploiting information gained from samples gathered thus far, but also have the capability to *explore* unchartered territory and collect new information about the problem's structure. A natural way of doing this is to use statistical models, where we do not merely have a single prediction at each search point, but have a full predictive distribution at each search point.

Jones [27] describes kriging models and shows how they can be used to optimise 1 dimensional problems. This chapter extends this work, and consists of a discussion and demonstration of how Gaussian process models can be used to solve problems with the response surface methodology.

9.3 Expected Improvement

If the model we build provides a predictive distribution at any test point, then we can use it to ask what improvement, over our current best sample, do we expect to get from sampling at any test point. Such a measure is known as the expected improvement (e.g. see Jones 2001, [27]). If the predictive distribution is Gaussian, then expected improvement is straightforward to calculate.

For a Gaussian process model, at any test point $\mathbf{x}$ we have a predictive distribution that is Gaussian with mean $y(\mathbf{x}) = \mathbf{k}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{y}$, and variance $s^2(\mathbf{x}) = \kappa - \mathbf{k}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{k}$. Therefore, for a Gaussian process model, we can calculate the expected improvement at any test point.

For a minimisation problem, the predicted improvement at $\mathbf{x}$ is defined as $I = I(\mathbf{x}) = f_{\text{best}} - \hat{y}(\mathbf{x})$, where f_{best} is the current best score and $\hat{y}(\mathbf{x})$ is the model's prediction at $\mathbf{x}$. The prediction is Gaussian distributed as $\hat{y}(\mathbf{x}) \sim \mathcal{N}(y(\mathbf{x}), s^2(\mathbf{x}))$, as is the improvement: $I \sim \mathcal{N}(f_{\text{best}} - y(\mathbf{x}), s^2(\mathbf{x}))$.

The expected improvement at $\mathbf{x}$ for models with Gaussian predictive distributions is defined as [27]:

$$EI_{\text{min}}(\mathbf{x}) = \int_{I=0}^{I=\infty} Ip(I)dI \tag{9.1}$$

$$= \int_{I=0}^{I=\infty} I\left\{\frac{1}{\sqrt{2\pi}s(\mathbf{x})}\exp\left[-\frac{(I-(f_{\text{best}}-y(\mathbf{x})))^2}{2s^2(\mathbf{x})}\right]\right\}dI \tag{9.2}$$

$$= s(\mathbf{x})\left[u\,\Phi(u) + \phi(u)\right] \tag{9.3}$$

where $u = \frac{f_{\text{best}} - y(\mathbf{x})}{s(\mathbf{x})}$.

The functions $\Phi(\cdot)$ and $\phi(\cdot)$ are the normal cumulative distribution and normal density function respectively.

$$\Phi(u) = \frac{1}{2}\,\text{erf}\left(\frac{u}{\sqrt{2}}\right) + \frac{1}{2} \qquad \phi(u) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{u^2}{2}\right) \tag{9.4}$$

For a maximisation problem, the predicted improvement at $\mathbf{x}$ is defined as $I = I(\mathbf{x}) = \hat{y}(\mathbf{x}) - f_{\text{best}}$, $u = \frac{y(\mathbf{x}) - f_{\text{best}}}{s(\mathbf{x})}$ and the expected improvement is:

$$EI_{\text{max}}(\mathbf{x}) = s(\mathbf{x})\left[u\,\Phi(u) + \phi(u)\right] \tag{9.5}$$

Figure 9.1 illustrates the concept of expected improvement for a GP model in a maximisation context. Observe the following:

- Expected improvement is maximum when the model's prediction is better than f_{best} *and* the predictive variance is high.

- When the model prediction is low *and* the predictive variance is low, the expected improvement is almost zero.

- When the model prediction is low *and* the predictive variance is high, the expected improvement is quite high.

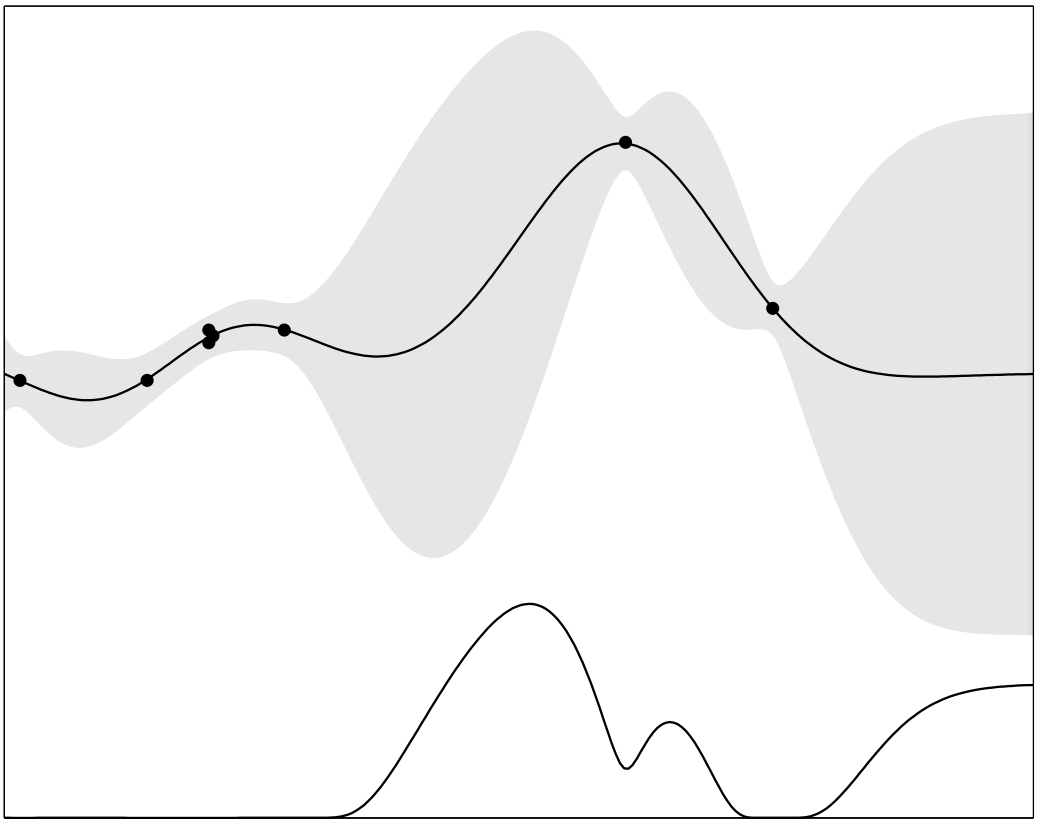

Figure 9.1: Expected Improvement for a GP model in a maximisation context. Data points are shown as black dots. The GP model's predictive distribution has a mean shown by the black line that fits the data, and a standard deviation shown by the shaded region surrounding the mean. The black line at the bottom shows the expected improvement given this particular GP model.

9.4 Gradient of Expected Improvement

To find a new search point that maximises the expected improvement, we can make use of gradient information. The gradient of $EI_{\max}$ with respect to x is:

$$\frac{\partial EI_{\max}(\mathbf{x})}{\partial \mathbf{x}} = \left[u\Phi(u) + \phi(u)\right]\frac{\partial s(\mathbf{x})}{\partial \mathbf{x}} + s(\mathbf{x})\Phi(u)\frac{\partial u}{\partial \mathbf{x}} \tag{9.6}$$

where

$$\frac{\partial s(\mathbf{x})}{\partial \mathbf{x}} = -\left(\frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}}\mathbf{C}^{-1}\mathbf{k}\right) \Big/ s(\mathbf{x}) \tag{9.7}$$

$$\frac{\partial u}{\partial \mathbf{x}} = \left(\frac{\partial y(\mathbf{x})}{\partial \mathbf{x}} - u\frac{\partial s(\mathbf{x})}{\partial \mathbf{x}}\right) \Big/ s(\mathbf{x}) \tag{9.8}$$

and

$$\frac{\partial y(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}}\mathbf{C}^{-1}\mathbf{y} \tag{9.9}$$

The $D \times N$ Jacobian $\frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}}$ is dependent on the form of the covariance function $k(\cdot,\cdot)$ and is constructed as follows:

$$\frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial k(\mathbf{x},\mathbf{x}_1)}{\partial x_1} & \cdots & \frac{\partial k(\mathbf{x},\mathbf{x}_N)}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial k(\mathbf{x},\mathbf{x}_1)}{\partial x_D} & \cdots & \frac{\partial k(\mathbf{x},\mathbf{x}_N)}{\partial x_D} \end{bmatrix} \tag{9.10}$$

where $\mathbf{x} = [x_1 \dots x_D]^{\mathrm{T}}$.

9.5 GPO

In this section, the Gaussian Processes for Optimisation (GPO) algorithm is described. This is not the first description of GPO. Jones [27] first introduced Kriging for optimisation using expected improvement to select the next iterate. Given the near equivalence of Kriging and Gaussian process regression, we consider this the first description of the algorithm. Büche, Schraudolph and Koumoutsakos [8] explicitly used Gaussian processes for optimisation, and demonstrated the algorithm's effectiveness on a number of benchmark

problems. This work did not make use of expected improvement, did not place prior distributions over the hyperparameters, and did not consider the deficiencies of using an axis-aligned covariance function to optimise objective functions with correlated output (dependent) variables. The algorithm presented below, and enhanced in the next chapter, takes these factors into account.

9.5.1 Standard GPO

Our goal is find an optimum $\mathbf{x}_{\text{opt}}$ of a problem $f^*(\cdot)$ with as few samples as possible. GPO begins by assuming a start point $\mathbf{x}_0$, which in general is sampled from some distribution reflecting our prior beliefs about $f^*(\cdot)$. At each iteration, the algorithm builds a GP model of the current data, and then finds the next search point $\mathbf{x}_{\text{new}}$ that has maximum expected improvement according to that model. A sample is taken at $\mathbf{x}_{\text{new}}$ and the result is added to the data set. Iterations continue until some stopping criterion is met, to be discussed below. Algorithm 9.1 describes GPO more specifically.

Algorithm 9.1: GPO for Maximisation

Input: optimisation problem $f^*(\cdot)$, and starting point $\mathbf{x}_0$

$\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}_0$, $y_{\text{best}} \leftarrow f^*(\mathbf{x}_0)$;
$\mathbf{y} \leftarrow [y_{\text{best}}]$, $\mathbf{X} \leftarrow \mathbf{x}_{\text{best}}$;
repeat
  $\boldsymbol{\theta}_{MAP} \longleftarrow \arg\max_{\theta} p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})$;
  $\mathbf{x}_{\text{new}} \longleftarrow \arg\max_{x} EI_{max}(\mathbf{x}|\mathbf{X}, \mathbf{y}, \boldsymbol{\theta}_{MAP})$;
  $y_{\text{new}} \leftarrow f^*(\mathbf{x}_{new})$;
  if $y_{new} \geq y_{best}$ **then**
    $y_{\text{best}} \leftarrow y_{\text{new}}$, $\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}_{\text{new}}$;
  $\mathbf{X} \leftarrow [\mathbf{X} \,|\, \mathbf{x}_{\text{new}}]$, $\mathbf{y} \leftarrow \left[\mathbf{y}^{\mathrm{T}} \,|\, y_{\text{new}}\right]^{\mathrm{T}}$;
until *(stopping criteria satisfied)* ;
return $\mathbf{x}_{best}$

The GP model is built by maximising the posterior density[1] $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})$, which is defined as the product of the likelihood function and the prior density over the hyperparameters. If multimodal posterior distributions are considered

[1] In practise, we maximise the log posterior density

a problem, then the GP model can be built by restarting the log posterior density maximisation multiple times from samples drawn from $p(\boldsymbol{\theta})$.

At each iteration, the resultant GP model is used to select the next search point $\mathbf{x}_{\text{new}}$ by finding a point that maximises the expected improvement. This can be achieved by using the gradient of the expected improvement as input to, for example, the conjugate gradient algorithm [54, 68]. To overcome the problem of suboptimal local maxima, multiple restarts are made starting from randomly selected points in the current data set $\mathbf{X}$. The new observation y_{new} is found from $f^*(\mathbf{x}_{\text{new}})$ and the results are added to the current data set.

9.5.2 Standard GPO Example - Linesearch

Figure 9.2 shows the results of running standard GPO on a simple $1D$ toy problem. Notice how the search initially focuses on the suboptimal maximum on the left. However, once the algorithm has sampled here a few times, the expected improvement of sampling in this suboptimal region diminishes quickly. At this point, the search expands into new regions. This demonstrates the principle of exploring versus exploiting - the algorithm will exploit local knowledge to make improvement, but will also explore when the expected returns of this exploitation reduce.

9.5.3 Bounded GPO

Standard GPO (algorithm 9.1) is has no bound on the region where the next sample point is selected from. The next point is chosen to maximise expected improvement, and no other constraint exists unless the particular application limits the domain of the decision variables. Therefore, in principle, standard GPO is capable of global optimisation. The GPO example in figure 9.2 demonstrates this. The domain from where the next sample point is selected from is limited only by the left and right limits, and the algorithm chooses the point within these limits that maximises expected improvement. The result is that the global optimum is found, even though the initial samples are in the vicinity of a suboptimal solution.

Sometimes we might only be interested in performing local optimisation, or we may have reason to believe that global optimisation would be extremely

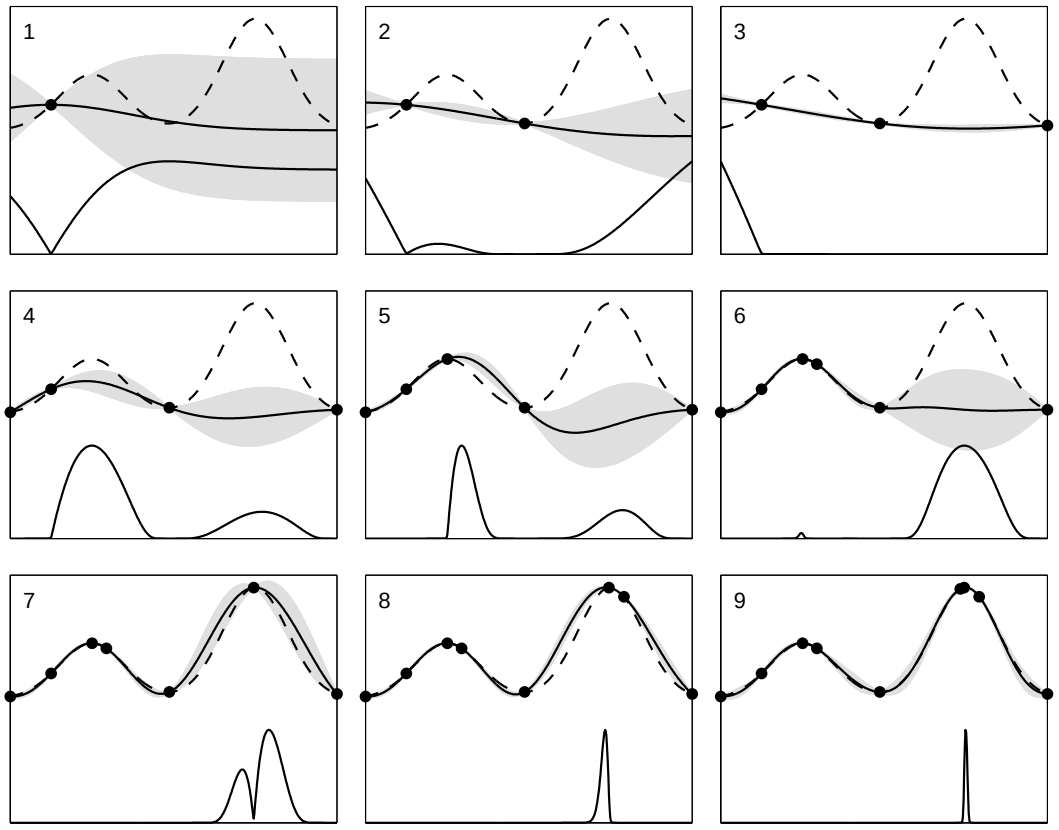

Figure 9.2: GPO applied to a $1D$ function for 9 iterations. The underlying function to be optimised is shown as a dashed line. The sample points are shown as black dots, along with the model's predictive distribution (the line surrounded by the shaded area shows the mean and standard deviation). The expected improvement is rescaled and shown by the solid line at the bottom of each window.

---

costly. In such cases we would like to limit the domain from which the next sample point can be taken. One way of doing this is to select as the next sample point the point that maximises expected improvement within some distance ϵ of the current best point. If we set ϵ to a small value we force GPO to search locally and exploit the current model information. However, exploration is limited and we would not expect to make huge improvements with each iteration. Instead, progress will be made slowly but steadily, climbing toward a local optimum. If ϵ is set to a large value, then GPO will tend to explore more and has the potential to make large gains per iteration, and might potentially discover new areas associated with separate local optima. Another difference is that progress tends to not be as steady, but occurs in a series of sudden leaps.

As an example, consider maximising a noiseless $6D$ elliptical Gaussian objective function with two different bounds on search: a small bound $\epsilon = 0.25$ and a large bound $\epsilon = 2.5$. I used an objective function with covariance matrix $\mathbf{\Sigma} = \mathrm{diag}(0.1, 0.28, 0.46, 0.64, 0.82, 1.0)^{-2}$. The starting point $\mathbf{x}_0$ for the optimisation had a distance of 1.54 to the maximum at $\mathbf{0}$, such that $f(\mathbf{x}_0) = 0.05$. Bounded GPO, with a target of $f(\mathbf{x}) \geq 0.975$ was run 50 times using both the small and large values of ϵ. The average results are shown in figure 9.3. Note that Bounded GPO with localised search $(\epsilon = 0.25)$ performs better than the less localised version $(\epsilon = 2.5)$. Furthermore, by observing the extent of the error bars, the optimisation can be seen to be less variable when $\epsilon = 0.25$.

This result suggests that if we wish to locally optimise, then it is better to use bounded GPO rather than global GPO. However, this raises the question of what bound to choose. If we have some prior knowledge about the form of the objective function, then we might have reasons for choosing a particular ϵ. If we have little or no prior knowledge then we take a risk in setting a fixed ϵ. That is, if ϵ is too small relative to the distance that must be covered to reach an optimum, then many samples will be required. Conversely, if ϵ is too large, the search might be excessively global as demonstrated in the current example.

The experiment was repeated with $\epsilon = 0.25$ for the first 8 samples, and then raised to $\epsilon = 2.5$ for the remainder of the optimisation. The results are shown in figure 9.4, and compared with the $\epsilon = 0.25$ case from the previous experiment. Now, the average progress in both cases is not notably different. Overall, it seems as though the best we can do is to initially perform very local optimisation (small ϵ), and then increase the search bound to make the search more global.

Another method for setting ϵ is to allow the search bound to adapt according to the recent progress of the algorithm. The search bound increases if an improvement is made and decreases otherwise. Call the factor by which ϵ increases or decreases ϵ_{inc}. Therefore, the search bound at iteration $n + 1$ is:

$$\epsilon_{n+1} = \begin{cases} \epsilon_n \times \epsilon_{inc}, & \text{if } f(\mathbf{x}_n) \text{ is better than } f_{best} \\ \epsilon_n / \epsilon_{inc}, & \text{otherwise} \end{cases} \tag{9.11}$$

where f_{best} is the best objective function value prior to $f(\mathbf{x}_n)$.

In practise, due to domain knowledge, we might set upper (ϵ_{max}) and lower

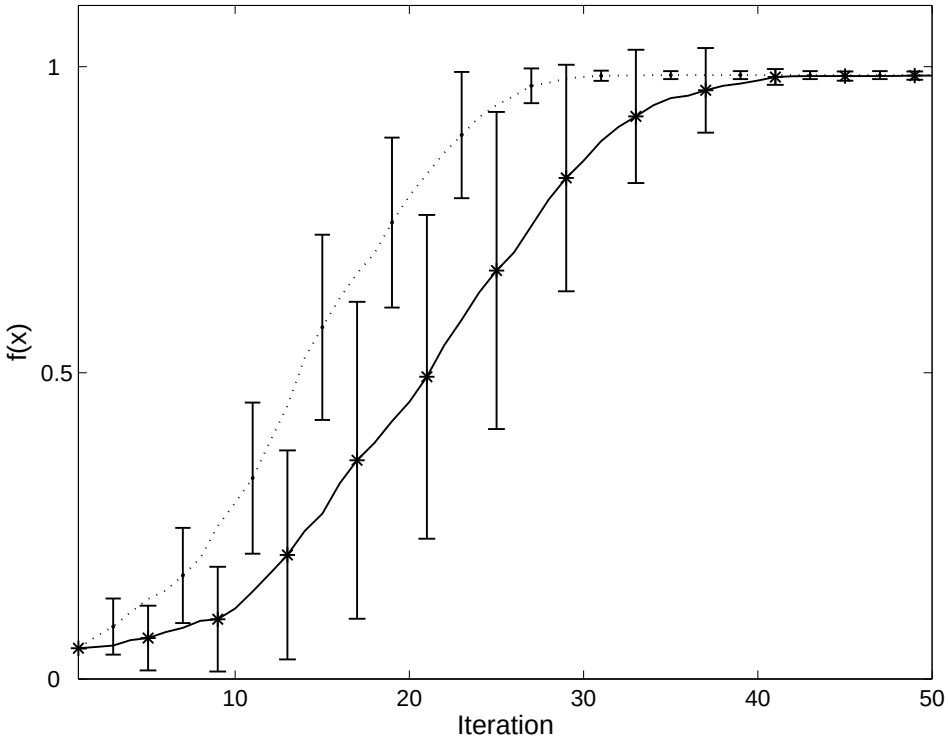

Figure 9.3: Example of Bounded GPO maximising a $6D$ elliptical Gaussian, with $\epsilon = 0.25$ (dotted) and $\epsilon = 2.5$ (solid). Shown is the mean and 1σ error bars of the best value attained at each iteration over 50 runs. Only every 4^{th} error bar is shown for clarity.

(ϵ_{min}) limits to restrict ϵ to a certain range. Note that this method is not suitable for global optimisation because the search bound ultimately decreases as the algorithm approaches a local optimum. This method is used for many of the local optimisation examples to follow as it seems to give good performance, and eliminates the need to *a priori* set a good value of ϵ - the algorithm sets this value as required.

9.6 Stopping Criteria

A general rule for when to stop is hard to define. Jones [27] suggests stopping when the maximum expected improvement falls below some threshold, however, this is not sufficient in general. If the particular data at some iteration of GPO resulted in the GP model being 'deceived' then the algorithm might pre-

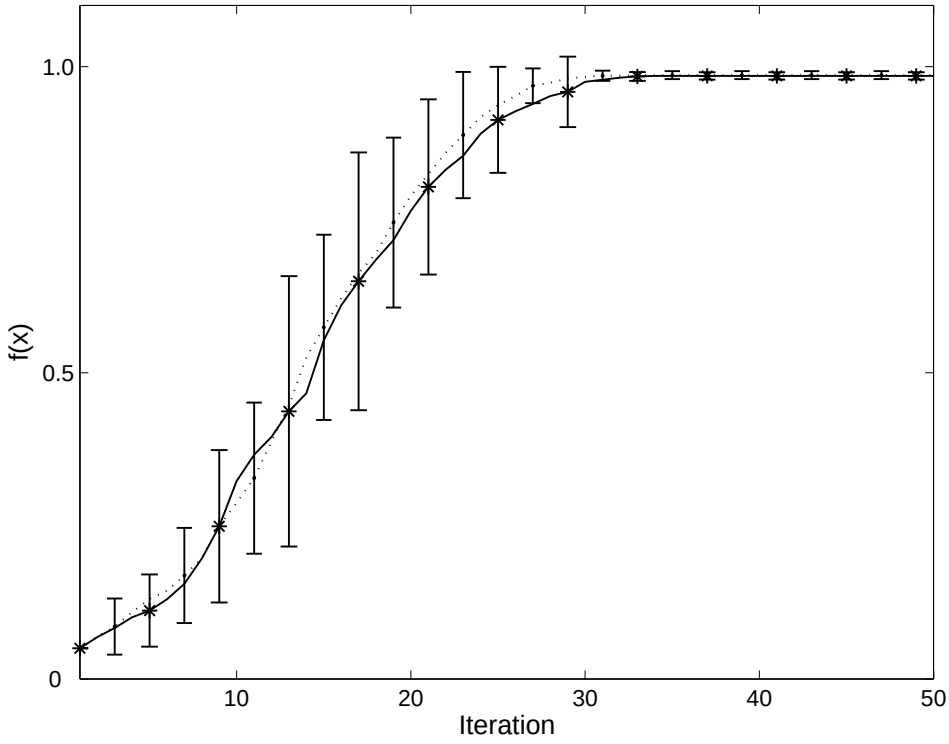

Figure 9.4: Example of Bounded GPO maximising a $6D$ elliptical Gaussian, with $\epsilon = 0.25$ (dotted) and $\epsilon = 0.25$ for the first 8 samples, and $\epsilon = 2.5$ otherwise (solid). Shown is the mean and 1σ error bars of the best value attained at each iteration over 50 runs. Only every 4^{th} error bar is shown for clarity.

---

maturely and erroneously terminate. An example of this is shown at iteration 3 of figure 9.2. Here, 3 data points exist that by chance are fitted very well by a very smooth GP model with low noise. As a result, the predictive variance and hence the expected improvement across the whole model becomes very small (although it is rescaled in the figure). At this point, if we put full trust in this model, then we might conclude that it is unlikely that we will make any further improvement. Clearly, such a conclusion is wrong. The problem here comes about because we use and commit to the maximum likelihood hyperparameters. Perhaps this problem would be alleviated if we were to take samples from posterior distribution over hyperparameters.

Instead of a general stopping rule, it is suggested here that one devise stopping rules based on problem knowledge or the particular application. For example, we might have a optimisation target or there might be some limit on the total number of samples that can be taken. Both of these cases implic-

itly define stopping conditions.

9.7 Problems with standard GPO

9.7.1 The Effect of Noise

Many optimisation problems are complicated by the fact that evaluation of the objective function is noisy. Either the underlying problem to be solved has a stochastic component, or the problem system is deterministic but our observations are noisy. In any case, we obviously want our underlying model to deal gracefully with noise. The most basic way to do this is to add a $\mathrm{cov}(\mathbf{x}_m, \mathbf{x}_n) = \delta_{mn}\sigma^2$ term to a Gaussian process covariance function. Such a model should be able to infer and account for additive, stationary, Gaussian white noise. In many cases, the noise will be more complicated than this. For example, the noise might be non-stationary (heteroscedastic) with a variance that changes across input space, or the noise might be non-Gaussian. Whether or not the simple stationary Gaussian noise model can adequately account for such noise depends on the degree of heteroscedasticity and non-Gaussianity. In this section, we demonstrate the effect of non-stationary Gaussian noise on the progress of GPO, applied to a simple $3D$ hyperelliptical Gaussian.

The objective function is $f(\mathbf{x}) = (1 + \alpha)\exp(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\,\mathbf{x})$, where $\boldsymbol{\Sigma} = \mathrm{diag}\left(\frac{1}{0.3^2}\frac{1}{0.65^2}\frac{1}{1.0^2}\right)$, and α is a Gaussian random variable with variance $\sigma^2 = 0.1$ for the noisy problem, and $\sigma^2 = 0$ for the noiseless control. The GPO covariance function is given by (7.1) and Gaussian prior distributions are placed over the hyperparameters. The optimisation starts from $\mathbf{x}_0 = \sqrt{\frac{1}{3}\log(400)}[0.3\ \ 0.65\ \ 1.0]^{\mathrm{T}}$ such that $f(\mathbf{x}_0) = 0.05$, and iterates until $\exp(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\,\mathbf{x}) \geq 0.975$ (until a sample is taken within the contour defined by $f(\mathbf{x}) = 0.975$ when $\alpha = 0$). At each iteration, the point of maximum expected improvement within a distance of 0.25 from the current best sample was used as the next sample point. The optimisation was repeated 50 times for the noisy and control problem. The control problem took an average of 15.6 ± 2.3 samples to solve, and the noisy problem took 18.0 ± 4.8 samples (mean $\pm$ 1 standard deviation). So in this simple example, GPO takes 15% more samples to solve the noisy problem but is $4.4 = \left(\frac{4.8}{2.3}\right)^2$ times more vari-

able.

Note that if the experiment was repeated with stationary noise, $f(\mathbf{x}) = \exp(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\,\mathbf{x}) + \alpha$ with $\alpha \sim \mathcal{N}(0, 0.1^2)$ then the optimisation initially proceeds randomly, as the initial signal level $f(\mathbf{x}_0) = 0.05 + \alpha_0$ is swamped by the noise.

9.7.2 Correlated Variables

Consider the optimisation of an hyperelliptical objective function, for example a Gaussian with covariance matrix $\boldsymbol{\Sigma} = \mathbf{V}^{\mathrm{T}}\boldsymbol{\Lambda}\mathbf{V}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues, and $\mathbf{V}$ is the orthonormal matrix of eigenvectors. We will examine two cases here. In the simplest case, the Gaussian is axis aligned such that $\mathbf{V} = \mathbf{I}$, so $\boldsymbol{\Sigma}$ is a diagonal matrix. The more general case is where the principal axes of the Gaussian are not aligned with the axes corresponding to the decision variables for the objective function. In other words, the output (dependent) variables are correlated and $\boldsymbol{\Sigma}$ is some arbitrary positive definite matrix.

In this section, we will use Gaussian Process Optimisation with bounded search (initial $\epsilon = 0.15$ and $\epsilon_{inc} = 1.5, \epsilon_{max} = 1, \epsilon_{min} = 0.1$) to find a point $\mathbf{x}^*$ such that $f(\mathbf{x}^*) > 0.975$, where $f(\mathbf{x}) = \exp(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\,\mathbf{x})$ and $\mathbf{x} \in \mathbb{R}^D$. Two cases are considered: the case when $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}$, and the case when $\boldsymbol{\Sigma} = \mathbf{V}^{\mathrm{T}}\boldsymbol{\Lambda}\mathbf{V}$, where $\mathbf{V}$ is defined by equations (7.5) and (7.6) with a Givens angle of $\rho_{ij} = \frac{\pi}{4}$. In both cases $\boldsymbol{\Lambda} = \mathrm{diag}(\frac{1}{\lambda_1^2} \ldots \frac{1}{\lambda_D^2})$, where $\lambda_1 \ldots \lambda_D$ are uniformly spaced from 0.1 to 1.0. The optimisations start from $\mathbf{x}_0 = \sqrt{\frac{1}{D}\log(400)}[\lambda_1 \ldots \lambda_D]^{\mathrm{T}}$ when $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}$, and $\mathbf{x}_0 = \sqrt{\frac{1}{D}\log(400)}\mathbf{V}^{\mathrm{T}}[\lambda_1 \ldots \lambda_D]^{\mathrm{T}}$ when $\boldsymbol{\Sigma} = \mathbf{V}^{\mathrm{T}}\boldsymbol{\Lambda}\mathbf{V}$. This means that $f(\mathbf{x}_0) = 0.05$ in both cases.

We will use a squared-exponential covariance function with a length scale $\exp(r_d)$ associated with each axis:

$$\mathrm{cov}(\mathbf{x}_i, \mathbf{x}_j) = \exp(v)\exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_i^{(d)} - x_j^{(d)})^2}{\exp(2r_d)}\right) + \delta_{ij}\exp(2\beta) \qquad (9.12)$$

Figure 7.1(left), (page 93), showed a sample GP generated from this covariance function. Notice how the features of this sample are axis aligned. We see no particular pattern as we move in the direction that bisects the axes.

Compare this to the second GP sample shown in figure 7.1. Clearly, this function has features that are not aligned to the variable axes. Here, we examine whether an axis-aligned covariance function is suitable for modelling and optimising objective functions with correlated output variables.

GPO was run 10 times for both problems (aligned and rotated) across problem dimensionalities $D = 2, 3, 4, 5, 6$. Figure 9.5 clearly shows that GPO performance (with covariance function (9.12)) is worse when the objective function contains correlated variables. Overall, optimising an axis-aligned hyperelliptical Gaussian with an axis-aligned Gaussian process model is at least 2 times easier than optimising the same objective when it has been significantly rotated relative to the decision variable axes.

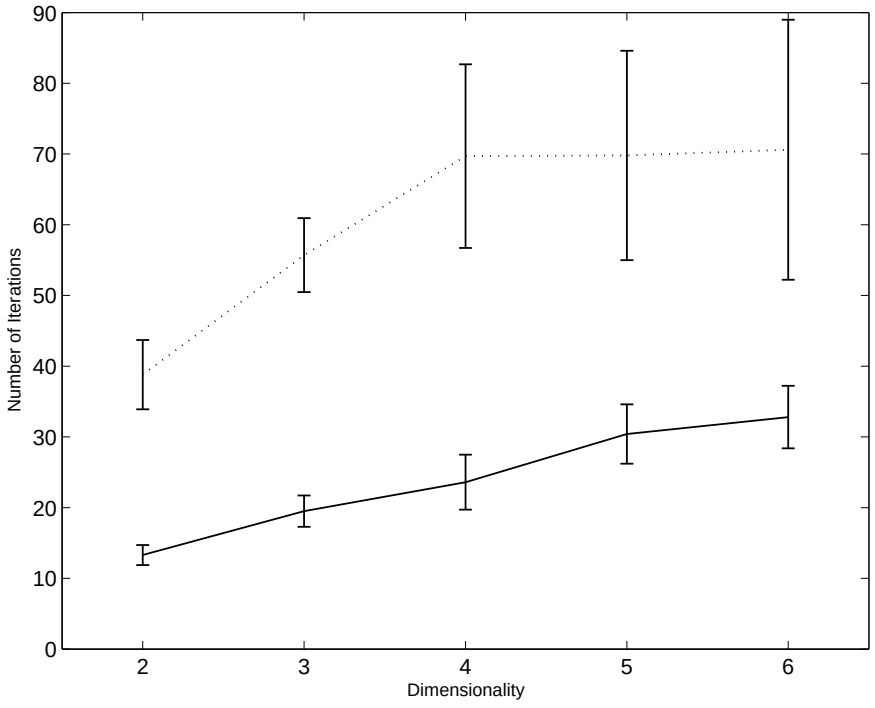

Figure 9.5: Results of running GPO with an axis-aligned covariance function on an axis-aligned (solid) and rotated (dotted) objective function. For each dimensionality, the mean number of iterations required to find a solution is shown along with the 1σ error bars.

9.7.3 Computational Complexity

Directly building a GP model of n observations has a computational cost of $O(n^3)$, corresponding to inversion of the covariance matrix. The degree to which this is a problem depends on the optimisation application. If samples are very expensive to obtain, and optimisation is offline, then this cost may be acceptable. However, if the optimisation runs on-line, in real time, and requires thousands of samples, then standard GPO may be too expensive. Section 10.4 discusses one method to potentially offset the problem of computational cost.

9.7.4 Non-stationarity

The covariance function employed by most practitioners is stationary. That is, $\mathrm{cov}(\mathbf{x}_i, \mathbf{x}_i + \boldsymbol{\delta}) = \mathrm{cov}(\mathbf{x}_j, \mathbf{x}_j + \boldsymbol{\delta}),\ \forall i, j$. This presents no problem if the objective function has characteristics that do nat vary over the input space. Informally, if the statistics that describe the behaviour of the objective function are independent of $\mathbf{x}$, then the objective function can be modelled and optimised with a stationary covariance function. If, however, the objective function has characteristics that vary significantly over the input space, then a stationary Gaussian process model may be inadequate.

Some objective functions have an unbounded range - either unbounded above, below or both. Many such problems also possess statistics that vary across input space. A simple example is the quadratic $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{x}$, which has a minimum at zero but is unbounded above. The quadratic has a rate of change that increases as one moves away from the origin, and therefore, the *local character* of the quadratic changes depending on where it is observed. Such a function cannot be adequately globally modelled with a stationary covariance function.

9.8 Summary

In this chapter, we have described and demonstrated a basic algorithm that uses Gaussian process inference for optimisation. The work presented here

differs from that of Jones [27] and Büche *et al.* [8] in the following ways.

(1) We use a Gaussian process regression model, rather than Jones' Kriging model for the response surface.

(2) We use the expected improvement to select the next iterate rather than the merit functions presented by Büche *et al.*

(3) We place prior distributions over the model parameters (i.e. the covariance function hyperparameters).

(4) We identify a number of problems with the GPO algorithm, in particular, the detrimental effect of correlated decision variables.

In the next chapter, we introduce enhancements to the algorithm that go some way toward dealing with the problems outlined in section 9.7.

Chapter 10

Enhanced GPO

10.1 Introduction

This chapter describes some enhancements to the basic GPO algorithm. These enhancements are introduced to help solve optimisation problems with correlated variables, and problems that may take many iterations to solve. We demonstrate these enhancements and show how GPO can be used to efficiently solve the double pole balancing task. Finally, we introduce and discuss the use of Bayesian neural networks for optimisation.

10.2 Rotated Covariance Functions

Section 9.7.2 suggested that GP models with axis aligned covariance functions perform poorly if the data to be modelled has features that are rotated relative to the coordinate axes. One way to overcome this is to use a parameterisation that allows for rotated covariance functions. There are a number of possible parameterisations, as discussed in chapter 7. Remember that the aim is to parameterise $\mathbf{\Sigma} = \mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}$, such that $\mathbf{V}$ is not forced to equal the identity matrix $\mathbf{I}$, and that any values of the parameters must result in a positive definite $\mathbf{\Sigma}$.

We repeat the optimisations of section 9.7.2, but use a GP model with a covariance function defined by a full covariance matrix, repeated here for con-

venience:

$$\mathrm{cov}(\mathbf{x}_m, \mathbf{x}_n) = \exp(v) \exp\left(-\tfrac{1}{2}(\mathbf{x}_m - \mathbf{x}_n)^{\mathrm{T}} \boldsymbol{\Sigma} (\mathbf{x}_m - \mathbf{x}_n)\right) + \delta_{mn} \exp(2\beta) \quad (10.1)$$

where $\boldsymbol{\Sigma}$ is parameterised with Givens angles, as described in section 7.3 (page 95).

The results are shown in figure 10.1 and show that using a rotated covariance function can speed the optimisation of a rotated objective function, when compared to optimisation using an axis-aligned covariance function. On the other hand, use of the rotated covariance function on the axis-aligned objective function leads to slower optimisation, especially as the problem dimensionality increases.

These results suggest that the most efficient optimisation is achieved by using a model that is well matched to the underlying objective function. A simple objective function (e.g. axis-aligned) is optimised faster with a simple (axis-aligned) model. A complex objective function (e.g. rotated) is optimised faster with a complex (rotated) model. Therefore, if the goal is to minimise the number of samples required for optimisation across a number of varying objective functions, then we cannot use a single model and hope to get good results. These results suggest that if we wish to use a single model to guide optimisation, then the best performance will be attained if that model is tuned to the properties of the objective function.

The situation here is consistent with the No Free Lunch Theorem [89, 90], which states, "*All algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost surfaces*". A specialised algorithm, tuned to a particular objective function will perform well when searching for the optimum of that function. However, that same algorithm will perform relatively poorly on some other problems that a general purpose algorithm might solve quite easily. In the current context, the GP model with the fully parameterised covariance matrix creates a specialised algorithm that is tuned to perform well on optimising rotated hyperelliptical Gaussian functions. The cost of such specialisation is a poorer performance on an axis-aligned function.

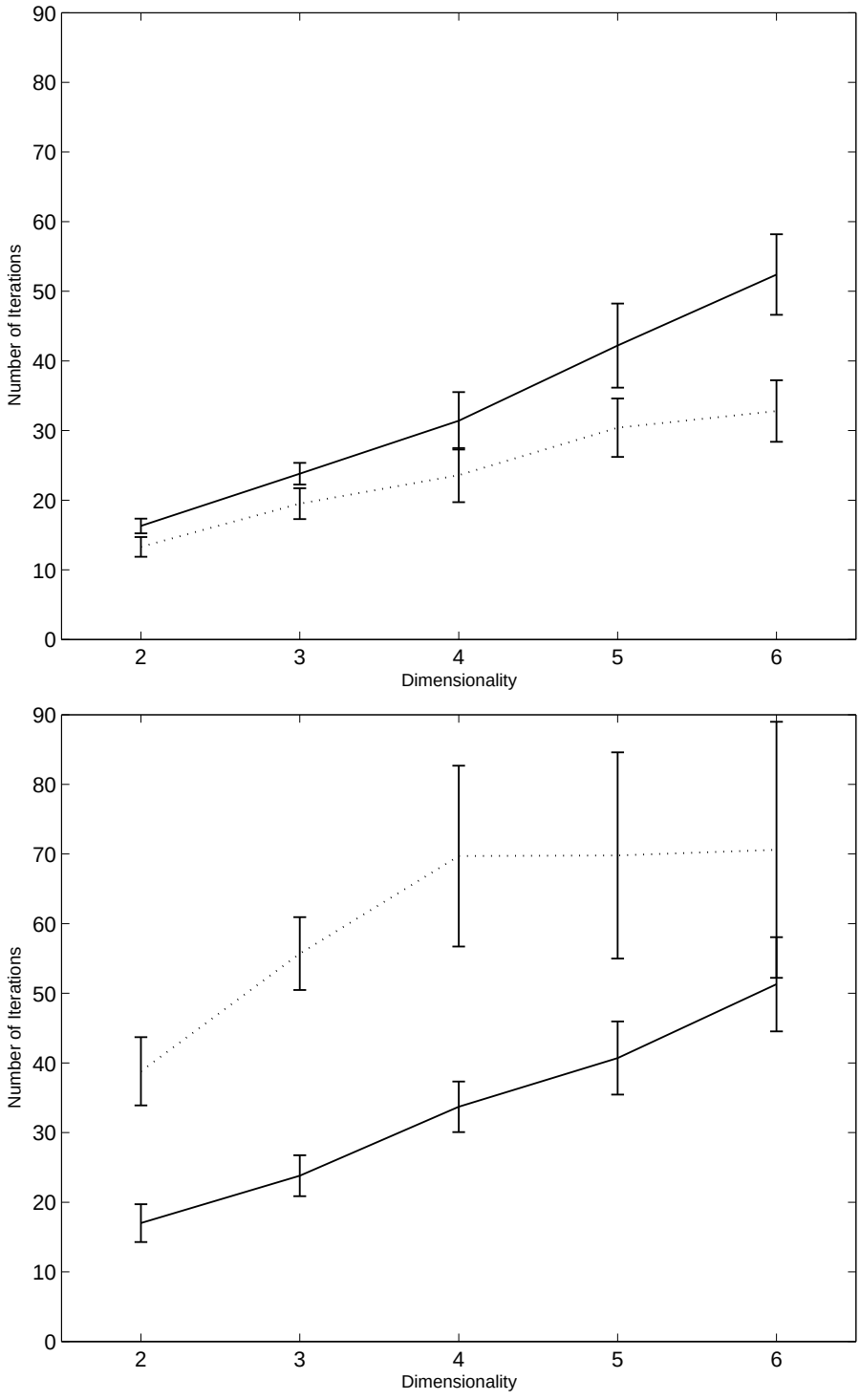

Figure 10.1: Results of running GPO, on an axis-aligned (top) and rotated (bottom) objective function, with an axis-aligned covariance function (dotted line) and a rotated covariance function (solid line). For each dimensionality, the mean number of iterations required to find a solution is shown along with the 1σ error bars.

10.3 Bayesian GPO

Section 10.2 showed how a more complex GP model may speed optimisation. However, we also saw how more complicated models can impede optimisation, particularly when the underlying objective function can be modelled sufficiently with a simpler model. This is reminiscent of the problems of overfitting. That is, a model that is too complex can fit data very well, but has a poor generalisation performance. For optimisation purposes with GPO, we certainly want to fit data well to maintain a reliable memory of what is good. However, if we overfit we risk making bad choices about where to go next. We want our models to generalise well and to make good predictions in unexplored space. The following sections explore the potential improvements that can be made to GPO by reducing overfitting and improving generalisation.

For regression problems, a principled way of reducing overfitting is to use Bayesian methods rather than the maximum likelihood or maximum *a posteriori* probability parameters for a single model. In the full Bayesian treatment, we don't commit to a single best model, and we attempt to retain as much information as we can about the posterior distribution over hyperparameters. That is, there are two distinct levels of inference. The first is where we make predictions by integrating over the posterior density, rather than using a single maximum *a posteriori* set of hyperparameters. The second level is where we deal with uncertainty over models. Here, we have the option of weighting various model types based on the evidence for those models and hence the posterior probability of each model. In this way, simple models that fit the data just as well as complex models are favoured.

10.3.1 Bayesian Expected Improvement

As discussed in section 8.1, for a single GP model $\mathcal{H}$, defined by a single parameterised covariance function, the predictive distribution for a new input vector $\mathbf{x}_*$ is:

$$p(y_*|\mathbf{x}_*, \mathcal{D}, \mathcal{H}) = \int p(y_*|\mathbf{x}_*, \boldsymbol{\theta}, \mathcal{D}, \mathcal{H}) p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{H}) d\boldsymbol{\theta} \tag{10.2}$$

$$\approx \frac{1}{R} \sum_{r=1}^{R} p(y_*|\mathbf{x}_*, \boldsymbol{\theta}_r, \mathcal{D}, \mathcal{H}) \tag{10.3}$$

where $\boldsymbol{\theta}_r$ are samples drawn from the posterior distribution with density $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{H})$, for example, by using MCMC methods.

Given a set of such samples we can calculate an approximation to the 'Bayesian expected improvement', EI_{Bayes}:

$$EI_{Bayes}(\mathbf{x}) = \frac{1}{R}\sum_{r=1}^{R} s_r(\mathbf{x})\left[u_r\,\Phi(u_r) + \phi(u_r)\right] \tag{10.4}$$

where $u_r = \frac{y_r(\mathbf{X}) - f_{\text{best}}}{s_r(\mathbf{X})}$ (for maximisation), $y_r(\mathbf{x})$ is the model prediction and $s_r^2(\mathbf{x})$ is the predictive variance at test point $\mathbf{x}$ for the Gaussian process corresponding to the r^{th} hyperparameter sample, $\boldsymbol{\theta}_r$.

Figure 10.2 shows an example of EI_{Bayes} calculated from 50 Metropolis samples and compares this to the expected improvement calculated from the MAP hyperparameters, EI_{MAP}. Note that the mean prediction for both the MAP and Bayesian methods are very similar, but that EI_{Bayes} and EI_{MAP} vary quite substantially. The EI peaks are almost at the same point which means that the next sample taken in both the Bayes and MAP cases would be very similar. However, the Bayesian method seems to have more "optimistic" expectations. That is, EI_{Bayes} is always greater than EI_{MAP}, and expects significant improvement in regions that EI_{MAP} considers less worthwhile.

If the posterior density over hyperparameters is approximated well by $\boldsymbol{\theta}_{MAP}$ then Bayesian GPO is not expected to significantly outperform MAP GPO. This can happen when we have large data sets, which can tend to make the posterior density unimodal, and concentrated around $\boldsymbol{\theta}_{MAP}$. In what follows, rather than calculating Bayesian expected improvement in the way above, we will instead focus our efforts on using Bayes' theorem to compare and weight different models to be used in the GPO algorithm.

10.3.2 Model Comparison

To recap, section 10.2 demonstrated a problem with GPO when optimising objective functions of differing complexity. A simple model produces good results on a simple objective function, but poor results on a complex objective function. A complex model, however, performs well on the complex problem, but is worse than the simple model optimising the simple problem.

We now reintroduce the notion of model comparison as discussed in section

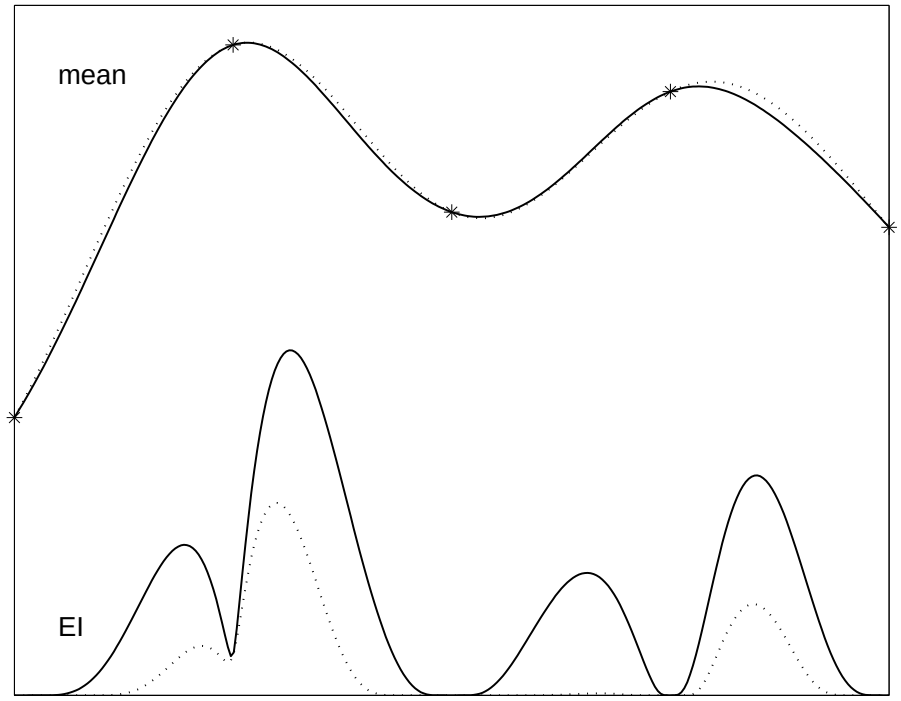

Figure 10.2: Expected Improvement for MAP (dotted) and Bayesian (solid) versions of GPO. The mean predictions from each method are at the top, while the rescaled EI is shown at the bottom.

8.2.1 (page 103). In that section, we saw how one could evaluate the model evidence, either to rank different models, or to weight their predictions appropriately. Roughly, the evidence for a simple model should be higher than that of a complex model in cases where they both fit the data sufficiently. Given that, consider incorporating the evaluation of model evidence into our optimisation algorithm. In particular, consider an algorithm consisting of two models, one simple, the other complex. So when a new sample is taken, instead of building a single model, we build two models. We decide on the next sample point by using predictions from both models, but weight the predictions according to the probability of each model.

To demonstrate the idea we repeat the optimisations from section 10.2. That is, we test the algorithm on two problems. The first is 'simple', and consists of an axis-aligned hyperelliptical Gaussian. The second is 'complex', consisting of a rotated hyperelliptical Gaussian. The algorithm uses two GP models to guide its search. The first has an axis-aligned covariance function, and the second a rotated covariance function with a fully parameterised covariance matrix.

At each iteration, sequential annealed importance sampling (section 8.2.5 on page 112) is used to evaluate the evidence for each model, and the posterior probability of each model is calculated. The next sample point is found by maximising the sum of the weighted expected improvement from each model, where each model is weighted according to its posterior probability. For M models we would maximise the following:

$$EI(\mathbf{x})_{comp} = \sum_{m=1}^{M} p(\mathcal{H}_m|\mathcal{D})EI(\mathbf{x})_m \tag{10.5}$$

where $EI(\mathbf{x})_m$ is the expected improvement from model m at $\mathbf{x}$.

The results are shown in figure 10.3, where it is clear that the algorithm using model comparison performs as well as both the simple and complex algorithms on the objective functions that they are specialised to optimise. In fact, it seems as though the new algorithm performs slightly better on both problems. Overall, at least in the case of optimising hyperelliptical Gaussians, we have overcome the problem of having to choose *a priori* which model to use.

Note that this performance is achieved even though the algorithm takes a number of iterations to form a useful belief about the form of the objective function. Obviously, it makes no sense to use just 1 or 2 samples as evidence for a decision either way. How many samples might be required to form an accurate belief? We can answer this question qualitatively with the results in figure 10.4. This figure shows the average posterior probability of the axis-aligned model (which is related to the posterior probability of the rotated model by $p(\text{aligned}|\mathcal{D}) = 1 - p(\text{rotated}|\mathcal{D})$), over 10 runs, for various dimensionalities. On average, when the objective function is axis-aligned, the algorithm shows a high posterior probability of the aligned model. On the other hand, the rotated model is far more plausible when the objective function is rotated. Note however, that it takes a significant number of samples to form these beliefs, and the number of samples required increases with problem dimensionality.

The example above shows the potential advantages to be gained by evaluating the evidence for multiple models at each iteration, and using the posterior probability to weight the importance of each model when searching for subsequent sample points. The cost of this improvement is increased computation, in particular, the cost of evaluating the evidence. For example, with an AMD Athlon XP 2400+ running MATLAB, it took 7 days to repeat the 6 dimensional

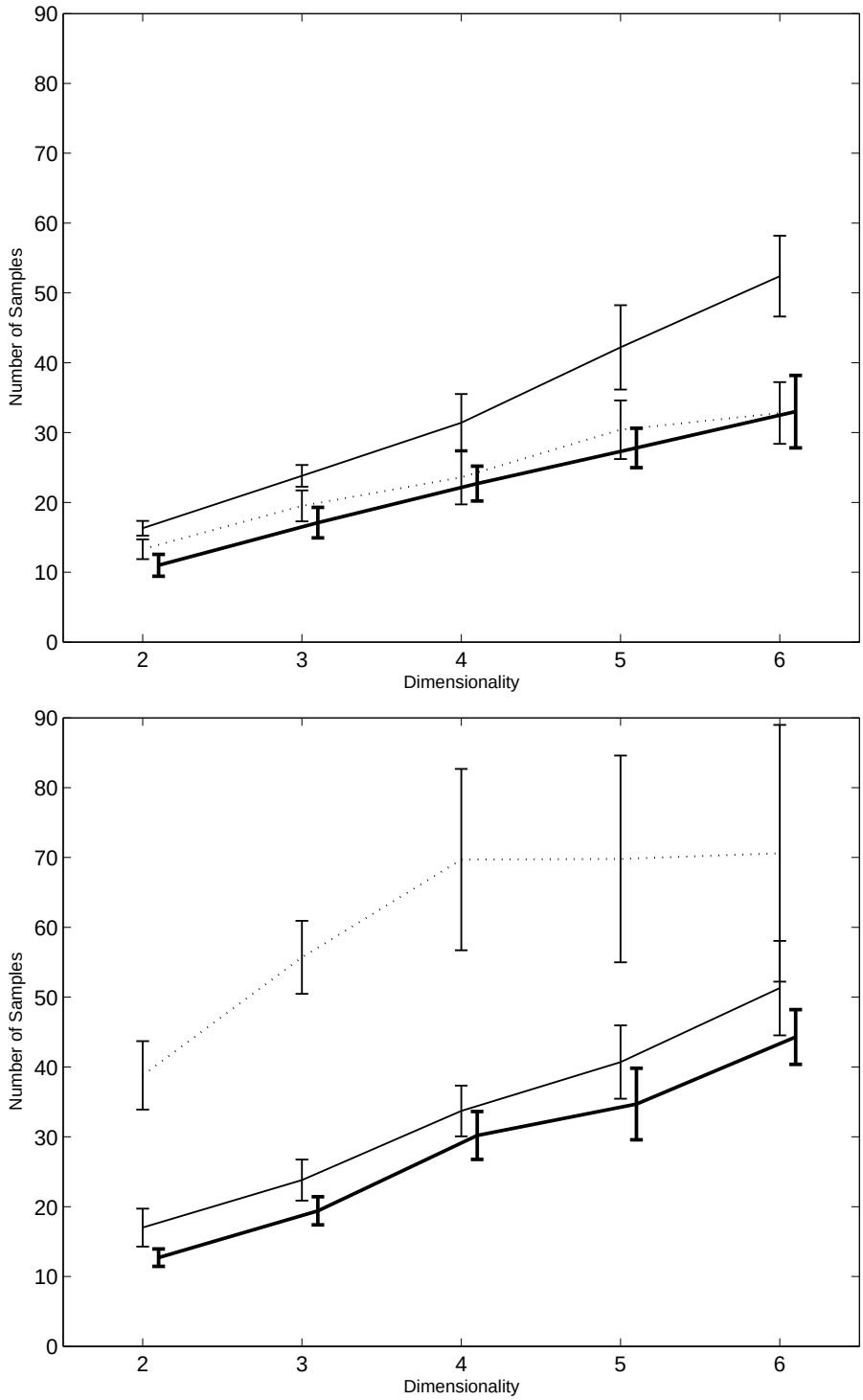

Figure 10.3: Results of running GPO with model comparison at each iteration (heavy solid line), compared with GPO with an axis-aligned (dotted line) and a rotated covariance function (solid line). All algorithms were run 10 times on an axis-aligned objective function (top), and rotated objective function (bottom). For each dimensionality, the mean number of samples required to find a solution is shown along with the 1σ error bars. The GPO with model comparison plot (heavy solid line) is offset slightly to the right for clarity.

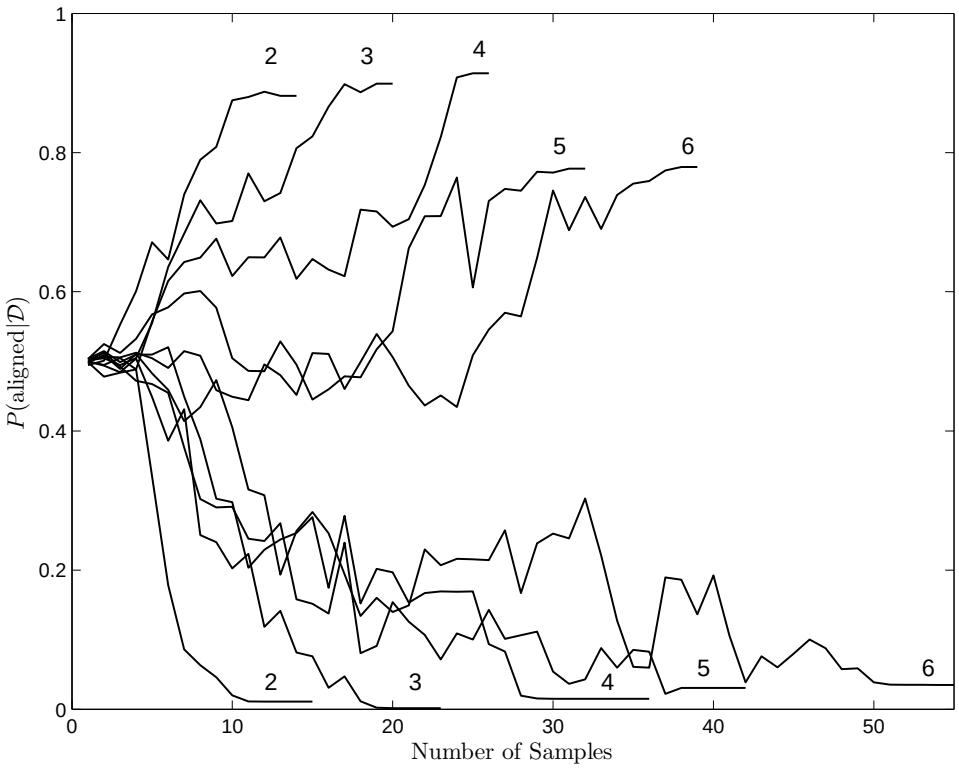

Figure 10.4: Average posterior probability of the axis-aligned model at each iteration (number of samples taken) across 10 runs for various problem dimensionalities. The curves at the top show the probability when the objective function is axis-aligned. The lower curves correspond to a rotated objective function. The numbers associated with each curve show the problem dimensionality.

---

optimisations (both axis aligned and rotated) using sequential annealed importance sampling with $w_{max}^{rat} = 2.0$ and 200 annealing runs. To speed the method up, the evidence for each model could be evaluated separately, and hence in parallel. Furthermore, the Markov chains for annealed importance sampling could also be run in parallel. So, given enough parallel resources, GPO with model comparison could be sped up significantly.

According to the No Free Lunch Theorem as discussed in section 10.2, the GPO with model comparison (MGPO) algorithm must perform exactly the same as the axis-aligned and the rotated algorithms, when averaged over all possible cost surfaces. So, even though the MGPO has better performance on hyperelliptical Gaussians, there must exist some problems where better performance would be observed from the axis-aligned or rotated algorithms. Consider now defining a battery of models, where each model is specialised

for some particular class of problems, and provides a measure of expected improvement at any point of interest. By having a big enough battery, sufficient parallel computational resources, and good prior knowledge about the types of problems we wish to solve, then we can begin to challenge the somewhat gloomy limitations of the NFLT. We cannot defeat NFLT, but at least we can search for an algorithm that performs well on problems of interest. We may not be concerned if our algorithm is absolutely hopeless at finding the optimum of a random landscape.

10.4 Reduced Rank Gaussian Processes for Optimisation

Basic GP regression scales as $\mathcal{O}(n^3)$ due to the cost of inverting the covariance matrix. The inverse, $\mathbf{C}^{-1}$, is required both for training (in calculating the log-likelihood), and for making predictions (mean and variance). Calculating the inverse exactly is acceptable for problems with relatively small n, but as n grows large (e.g. $n > 10000$) exact inverses become far too expensive. In this section we consider speeding up basic GPO by using an approximation that reduce the $\mathcal{O}(n^3)$ cost.

Gaussian processes optimisation with expected improvement requires a model that provides a predictive mean and variance at any input point. Therefore, any approximate model we use to speed up optimisation needs to provide a predictive mean and variance. One such model is a reduced rank Gaussian process (RRGP) model [55], as discussed in chapter 5.

To reiterate, an RRGP model of n data points $\mathbf{X}_n$ is formed by considering a set of $m \leq n$ support points $\mathbf{X}_m$. In this thesis, the support points form a subset of $\mathbf{X}_n$ although this is not necessary in general. To simplify, let the support points be the first m points in $\mathbf{X}_n$. That is, $\mathbf{X}_n = [\mathbf{X}_m | \mathbf{x}_{m+1} \dots \mathbf{x}_n]$. Given a training input set $\mathbf{X}_n$, and covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$, we form the matrices $\mathbf{K}_{nm}$ and $\mathbf{K}_{mm}$ where $\mathbf{K}_{ab}$ is an $a \times b$ matrix, with the $(i,j)^{th}$ entry defined by $k(\mathbf{x}_i, \mathbf{x}_j)$. At any test point $\mathbf{x}_*$ we find the predictive mean $m(\mathbf{x}_*)$ and variance $v(\mathbf{x}_*)$ as follows:

$$m(\mathbf{x}_*) = \mathbf{q}(\mathbf{x}_*)^{\mathrm{T}} \boldsymbol{\mu}_* \tag{10.6}$$

$$v(\mathbf{x}_*) = \sigma^2 + \mathbf{q}(\mathbf{x}_*)^{\mathrm{T}} \boldsymbol{\Sigma}_* \mathbf{q}(\mathbf{x}_*) \tag{10.7}$$

where $\mathbf{q}(\mathbf{x}_*)^{\mathrm{T}} = \left[\mathbf{k}(\mathbf{x}_*)^{\mathrm{T}} \mid k_{**}\right]$.

The scalar $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$, and $\mathbf{k}(\mathbf{x}_*) = [k(\mathbf{x}_*, \mathbf{x}_1) \ldots k(\mathbf{x}_*, \mathbf{x}_m)]$ is the $m \times 1$ vector of covariances between $\mathbf{x}_*$ and the m support points.

The $(m+1) \times 1$ vector $\boldsymbol{\mu}_*$ is the mean of the posterior distribution over the augmented weights, and is equal to

$$\boldsymbol{\mu}_* = \sigma^{-2} \boldsymbol{\Sigma}_* \left[\mathbf{K}_{nm} \mid \mathbf{k}_*\right]^{\mathrm{T}} \mathbf{y} \tag{10.8}$$

where $\mathbf{k}_* = [k(\mathbf{x}_*, \mathbf{x}_1) \ldots k(\mathbf{x}_*, \mathbf{x}_n)]$ is the $n \times 1$ vector of covariances between $\mathbf{x}_*$ and the n training inputs.

The $(m+1) \times (m+1)$ matrix $\boldsymbol{\Sigma}_*$ is the covariance matrix of the posterior distribution over the augmented weights, and is equal to

$$\boldsymbol{\Sigma}_* = \begin{bmatrix} \boldsymbol{\Sigma}^*_{11} & \boldsymbol{\Sigma}^*_{12} \\ \boldsymbol{\Sigma}^*_{21} & \boldsymbol{\Sigma}^*_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Sigma}^{-1} & \mathbf{k}(\mathbf{x}_*) + \sigma^{-2}\mathbf{K}_{nm}^{\mathrm{T}}\mathbf{k}_* \\ \mathbf{k}(\mathbf{x}_*)^{\mathrm{T}} + \sigma^{-2}\mathbf{k}_*^{\mathrm{T}}\mathbf{K}_{nm} & k_{**} + \sigma^{-2}\mathbf{k}_*^{\mathrm{T}}\mathbf{k}_* \end{bmatrix}^{-1} \tag{10.9}$$

where the $m \times m$ matrix $\boldsymbol{\Sigma}^{-1} = \sigma^{-2}\mathbf{K}_{nm}^{\mathrm{T}}\mathbf{K}_{nm} + \mathbf{K}_{mm}$

The computational cost of prediction is an initial $\mathcal{O}(nm^2)$ to compute $\boldsymbol{\Sigma}$. Then, each test point has a cost of $\mathcal{O}(nm)$, due to the most expensive computation $\mathbf{K}_{nm}^{\mathrm{T}}\mathbf{k}_*$. Once $\boldsymbol{\Sigma}$ has been calculated, it is best to calculate $\boldsymbol{\Sigma}_*$ by inversion by partitioning. Firstly, let $\mathbf{r} = \mathbf{k}(\mathbf{x}_*) + \sigma^{-2}\mathbf{K}_{nm}^{\mathrm{T}}\mathbf{k}_*$.
Then $\boldsymbol{\Sigma}^*_{11} = \boldsymbol{\Sigma} + \boldsymbol{\Sigma}\mathbf{r}\mathbf{r}^{\mathrm{T}}\boldsymbol{\Sigma}/\rho$, $\boldsymbol{\Sigma}^*_{12} = -\boldsymbol{\Sigma}\mathbf{r}/\rho = \boldsymbol{\Sigma}^{*\mathrm{T}}_{21}$, and $\boldsymbol{\Sigma}^*_{22} = 1/\rho$ where $\rho = k_{**} + \sigma^{-2}\mathbf{k}_*^{\mathrm{T}}\mathbf{k}_* - \mathbf{r}^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{r}$.

10.4.1 Reduced Rank GP Training

To construct a reduced rank GP model, we require a covariance function, usually parameterised with a set of hyperparameters, $\boldsymbol{\theta}$ and σ^2, and a set of support inputs. For training, we wish to use Bayes' theorem to infer the most likely or most probable hyperparameters and support subset.

Recall that the support subset is equal to the first m training inputs. This means a different support set simply corresponds to reordering of the training inputs. One such reordering is described as follows. Let $\mathbf{z}$ be a vector of indices that reorders the original data $\mathbf{X}$ and $\mathbf{y}$ to form the training

data $\mathbf{X}_n$ and $\mathbf{y}_n$. That is, given $\mathbf{z} = [z_1 \dots z_n]^{\mathrm{T}}$, we reorder the original data $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_n], \mathbf{y} = [y_1 \dots y_n]^{\mathrm{T}}$ to form the training data $\mathbf{X}_n = [\mathbf{x}_{z_1} \dots \mathbf{x}_{z_n}], \mathbf{y}_n = [y_{z_1} \dots y_{z_n}]^{\mathrm{T}}$. Note that only the first m elements of $\mathbf{z}$ are relevant, and their ordering is arbitrary – these are the elements that define the support subset. The remainder of the training set can be ordered arbitrarily.

The marginal likelihood of the hyperparameters given a support set is $p(\mathbf{y}_n|\mathbf{X}_n, \mathbf{z}, \boldsymbol{\theta}, \sigma^2) = \mathcal{N}(0, \mathbf{Q})$ where $\mathbf{Q} = \sigma^2\mathbf{I}_n + \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^{\mathrm{T}}$. The log of this is [55]:

$$\mathcal{L} = -\tfrac{n}{2}\log(2\pi) - \tfrac{1}{2}\log|\mathbf{Q}| - \tfrac{1}{2}\mathbf{y}_n^{\mathrm{T}}\mathbf{Q}^{-1}\mathbf{y}_n \tag{10.10}$$

where we can use the matrix inversion lemma to efficiently calculate

$$\mathbf{Q}^{-1} = \sigma^{-2}\mathbf{I}_n - \sigma^{-4}\mathbf{K}_{nm}(\mathbf{K}_{mm} + \sigma^{-2}\mathbf{K}_{nm}^{\mathrm{T}}\mathbf{K}_{nm})^{-1}\mathbf{K}_{nm}^{\mathrm{T}} \tag{10.11}$$

which requires inversion of an $m \times m$ matrix, rather than an $n \times n$ matrix.

Given a support set, we can maximise $\mathcal{L}$ to find the maximum likelihood hyperparameters. If we wish, we incorporate prior beliefs about the hyperparameters and a log prior term to equation (10.10). We can then find the maximum *a posteriori* (MAP) hyperparameters.

To learn the support set, ideally one would like to find the support set that maximises $\mathcal{L}$. Unfortunately, the number of possible support sets is combinatorial, ${}^nC_m$ [55], which is obviously prohibitive for large n (e.g. ${}^{32}C_5 = 201376$). Instead, we examine just a fixed number of the possible subsets, and use the best for RRGP predictions.

10.4.2 Reduced Rank GP Optimisation

Given a support set and hyperparameters, the RRGP model can be queried for a predictive mean and variance at any input point. This prediction can then be used to find the expected improvement $EI_{rrgp}(\mathbf{x})$, and its maximum, resulting in reduced rank GP optimisation (RRGPO).

Note that RRGPO is the same as basic GPO, except that we use a reduced rank approximation to the full GP. Furthermore, the derivatives of 10.6 and 10.7 can be found, enabling the evaluation of $\dfrac{\partial EI_{rrgp}(\mathbf{x})}{\partial x_i}$. However, this derivative is

complicated, and is not evaluated in the example to follow. Instead, $EI_{rrgp}(\mathbf{x})$ is maximised by random search.

In this example, RRGPO is used to maximise a noiseless $36D$ hyperelliptical Gaussian $f(\mathbf{x}) = \exp(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\,\mathbf{x})$, where $\boldsymbol{\Sigma} = \mathrm{diag}(\frac{1}{\lambda_1^2} \ldots \frac{1}{\lambda_{36}^2})$, and $\lambda_1 \ldots \lambda_{36}$ are uniformly spaced from 0.3 to 1.0. The optimsation starts from $\mathbf{x}_0 = \frac{\sqrt{\log 400}}{6}[\lambda_1 \ldots \lambda_{36}]^{\mathrm{T}}$, (such that $f(\mathbf{x}_0) = 0.05$), and proceeds until $f(\mathbf{x}_*) \geq 0.975$.

At each iteration of the optimisation, a RRGP is built by finding hyperparameters and a support set that maximises the joint posterior density $p(\boldsymbol{\theta}, \sigma^2, \mathbf{z}|\mathcal{D})$. At iteration n, if $n \leq 36$, then a full GP model is used ($m = n$), and the support set is equivalent to the training inputs. In this case, we simply maximise the posterior density over hyperparameters. For $n > m$, we use a reduced rank of $m = 36$. In this case, the support set is not equivalent to the training input set. We examined a random selection of 10 support subsets, and the support set that resulted in the maximum posterior probability (after optimising the hyperparameters) was chosen as the support set for predictions. This is suboptimal, but as mentioned above, it is too expensive to examine all possible support sets.

Results are shown in figure 10.5, along with an $18D$ example with $m = 36$. The $18D$ problem was solved in 69 iterations, and the $36D$ problem took 147 iterations. Note that this performance was attained even though a maximum of 36 support points were included in the support set for training and prediction, showing that GPO using a reduced rank approximation is feasible, and useful at least for this noiseless toy example.

10.5 Double Pole Balancing with GPO

This section describes the double pole balancing task and shows how it can be solved efficiently using GPO.

10.5.1 The Double Pole Balancing Task

The double pole balancing task consists of two upright poles (or inverted pendulums), attached by hinges to a cart. The goal is to keep the two poles bal-

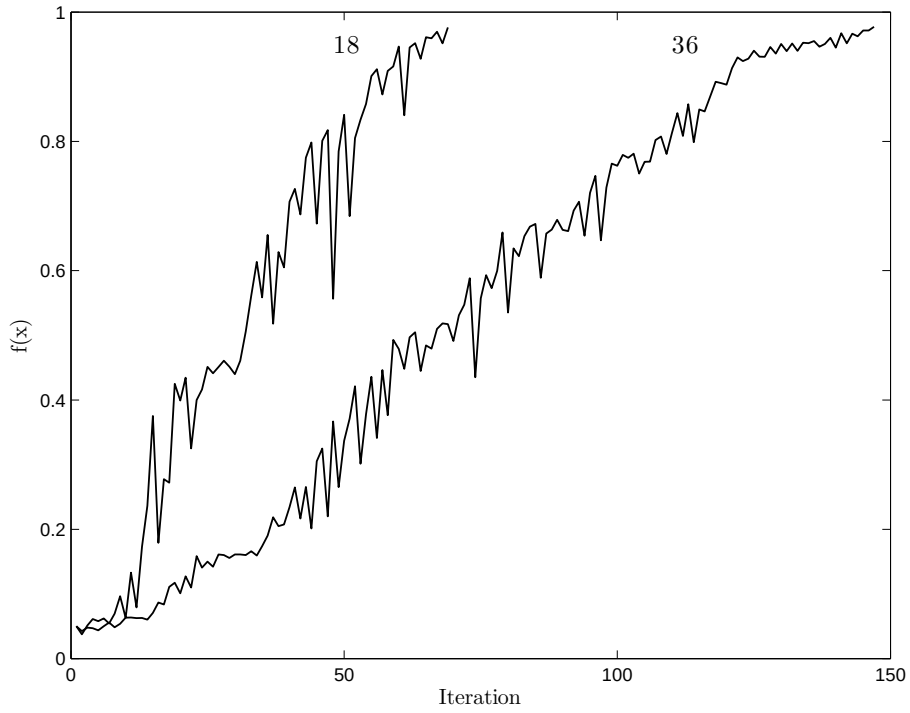

Figure 10.5: Reduced Rank Gaussian Processes Optimisation of a 18 and 36 dimensional hyperelliptical Gaussian. Both optimisations had a reduced rank of $m = 36$.

---

anced by applying a $[-10, 10]N$ force to the cart. Balanced poles are defined as within $\pm 36\,^\circ$ from vertical, and the cart is limited to a track which is $4.8m$ long. The controller is supplied with inputs from sensors measuring the cart's position and velocity $x, \dot{x}$ and the angle and angular velocity of each pole with respect to the cart $\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2$. The poles have different lengths and masses (pole 1 is $0.1m$ and $0.01kg$; pole 2 is $1.0m$ and $0.1kg$) and the system is noiseless with initial state vector $\mathbf{s} = [x\ \dot{x}\ \theta_1\ \dot{\theta}_1\ \theta_2\ \dot{\theta}_2]^{\mathrm{T}} = [0\ 0\ 0\ 0\ \frac{\pi}{180}\ 0]^{\mathrm{T}}$, where angles are measured in rad from vertical, and angular velocities are measured in rad/s. The centre of the track is defined as $x = 0$, and is the position of the cart at the beginning of the task. Note that this task is Markovian as the full system state vector $\mathbf{s}$ is available to the controller and is the same as the "double pole balancing with velocity information" problem as presented by Stanley and Miikkulainen [76, 75, 74].

If the goal is to keep the poles balanced for as long as possible, one solution is to wiggle the poles back and forth about a central position. To prevent this, Gruau [22] defined a fitness function that penalises such solutions. This Gruau fitness is equal to the weighted sum of two fitness components, $f_{\text{gruau}} =$

$0.1f_1 + 0.9f_2$, [76, 74]. The two components are defined over 1000 time steps (10 seconds of simulated time):

$$f_1 = t/1000 \tag{10.12a}$$

$$f_2 = \begin{cases} 0 & \text{if } t < 100, \\ \frac{0.75}{\sum_{i=t-100}^{t}(|x^i|+|\dot{x}^i|+|\theta_1|+|\dot{\theta}_1|)} & \text{otherwise.} \end{cases} \tag{10.12b}$$

where t is the number of time steps both poles remained balanced during the $10s$ trial. f_{gruau} can be maximised by keeping both poles balanced, and by maintaining the cart steady at the centre of the track during the final $1s$ of the trial. Effectively, to maximise f_{gruau} the controller must balance the poles without 'wiggling'.

As the denominator of (10.12b) approaches zero, f_2 approaches infinity. For this reason, f_{gruau} was non-linearly rescaled into the range $[0, 1]$ to give:

$$f_{\text{gpo}} \;=\; \tanh\left(\frac{f_{\text{gruau}}}{2}\right) \tag{10.13}$$

Controllers were considered successful solutions when $f_{\text{gpo}} \geq \tanh\left(\frac{5}{2}\right)$.

10.5.2 Feedforward Neural Network Controllers

The double pole balancing task described above is a non-linear, unstable control problem. However, because the poles have different lengths and masses, the system is controllable. In addition, the task is Markovian. Overall, full knowledge of the system state is sufficient to balance the poles, and this can be achieved with a mapping from $\mathbf{s}$ to u, our control force. In other words, there exists at least one mapping $f_{\text{solution}}(\cdot) : \mathbf{s} \mapsto u$ that is capable of balancing the poles. A successful controller must functionally approximate such a mapping.

In this section, controllers are implemented by feedforward neural networks with a single hidden layer. The control force from a network with H units is:

$$u = 10 \tanh\left(\mathbf{w}_o^{\mathrm{T}} \tanh\left(\mathbf{W}_i^{\mathrm{T}}\mathbf{s} + \mathbf{b}\right)\right) \tag{10.14}$$

where $\mathbf{w}_o$ is an $H \times 1$ vector of output weights, $\mathbf{b}$ is an $H \times 1$ vector of biases, $\mathbf{W}_i$ is a $6 \times H$ matrix of input weights, $\mathbf{s}$ is the 6×1 state vector. The controller output force is limited to $[-10, 10]N$.

10.5.3 Optimisation and Incremental Network Growth

The optimisation started with a single unit in the network, $H = 1$. Initially, therefore, there were 8 parameters that need optimising. GPO, with an axis aligned covariance function as in equation (7.1), and data rotation (section 7.5) prior to training was used to optimise these weights until either there had been no improvement in the best fitness for 64 consecutive samples, or 250 samples had been taken. When either of these conditions were met, the current optimised parameters were frozen, and a new unit with zeroed weights was added to the network. The cycle repeated until a solution was found, or 5 units and their weights had been optimised. Note that the initial weights for the first iteration were zero, meaning that the algorithm started from the same place every time it was run. Further note that at all stages of the optimisation, only 8 parameters were being optimised.

10.5.4 Optimisation Results

The optimisation described above was run 100 times, as shown in figure 10.6. 96 of these runs found a successful controller solution with $f_{\text{gruau}} \geq \tanh\left(\frac{5}{2}\right)$. The median number of evaluations required to find a successful controller was 151, and the mean was 194.

Table 10.1 shows that the majority (78%) of successful controllers used only 1 unit in their solution (i.e. 6 input weights, 1 bias and 1 output weight).

Number of Units	% of Controllers
1	75
2	12
3	4
4	2
5	3

Table 10.1: Number of units per successful controller on the double pole balancing task.

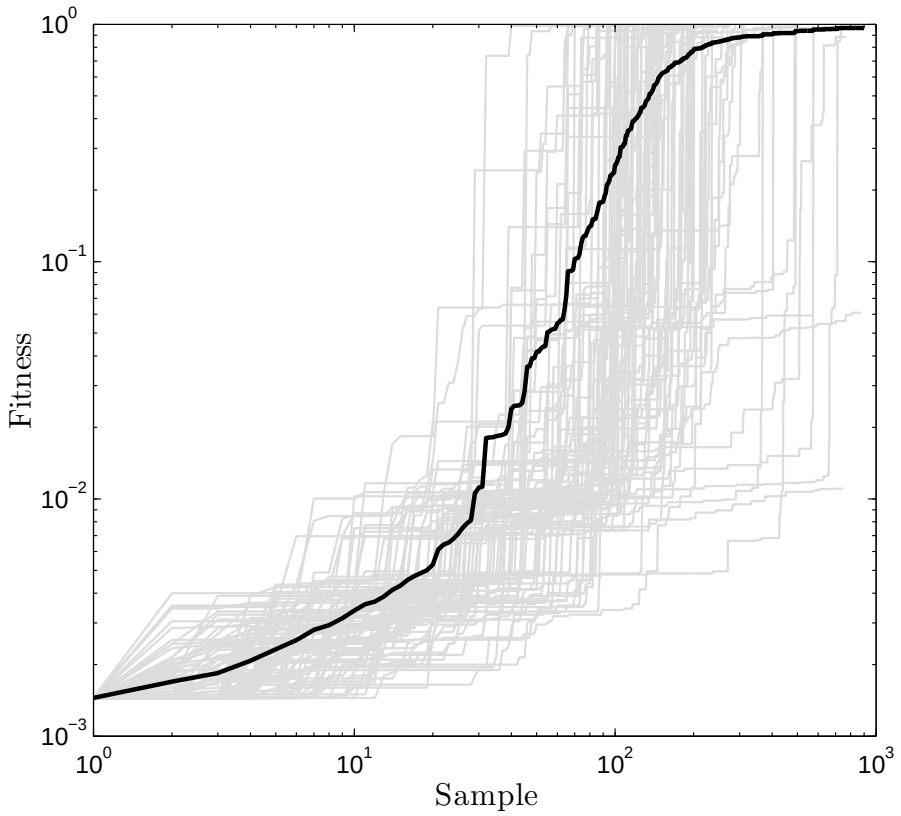

Figure 10.6: Double Pole Balancing with Gruau fitness, optimised using GPO. The figure shows 100 separate optimisations (grey) and the average (black).

10.5.5 Comparison with NEAT

Stanley and Miikkulainen [76, 75, 74] introduced "Neuroevolution of Augmenting Topologies" (NEAT), and applied it to a number of pole balancing tasks, including the double pole balancing with velocity information task, which is the same as the double pole balancing problem as presented above. The NEAT method is a genetic algorithm with mutation and crossover operations specially crafted to enhance the evolution of neural network controllers. The details of NEAT are not important here, we simply use the NEAT results for comparison with GPO.

The NEAT method produced impressive results in solving the double pole balancing task with velocity information. On average, NEAT required on average 3578 network evaluations before finding a controller solution, which compared favourably with other results from literature. GPO produced suc-

cessful controllers in 96 out of 100 trials, and did so with a mean of 194 evaluations. This is a significant improvement over NEAT.

10.6 Bayesian Neural Networks for Optimisation

In this section we examine the use of neural network models for optimisation purposes. That is, instead of using a GP model to build a response surface, we use a Bayesian neural network (BNN). A BNN model, after MacKay [36, 39, 38] and Neal [48, 46] and reviewed by Lampinen [28], provides not only a prediction at any given input point, but also a measure of model uncertainty at that point. We can therefore use a BNN model to calculate the expected improvement at any input point, and use the point of maximum expected improvement as the next sample point.

One advantage in using a neural network model over a GP model is that neural networks can naturally model non-stationary objective functions. For a GP model, it is quite hard to define a covariance function that is capable of dealing with significant non-stationarity. As an example, consider a model that consists of a sum of two single-hidden-layer neural-networks. The first network has small weights and makes non-zero predictions in a limited region, $\mathcal{X}_1$. The second network has large weights and makes non-zero predictions in a second, disjoint region $\mathcal{X}_2$. The sum of the networks will make predictions that vary across the input space, depending on whether the test point resides in $\mathcal{X}_1$ or $\mathcal{X}_2$. In other words, this model can represent non-stationarity.

Another advantage in using neural network models is that the cost of evaluating the log-likelihood scales as $\mathcal{O}(n)$. Compare this to a basic GP model, where the complexity is $\mathcal{O}(n^3)$. This means we could use Bayesian neural network models for optimisation problems requiring many thousands, perhaps millions of evaluations. With basic GPO, we might be limited to a 1000 or so evaluations.

Consider a feedforward network with a single hidden layer, and a single output unit. We assume additive Gaussian noise with variance σ^2 on the observed targets and wish to model data $\mathcal{D} = \{(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n)\}$. The network output as a function of input is $f(\mathbf{x}\,; \mathbf{w})$ where $\mathbf{w}$ are the networks parameters, or weights. Consider the case where we have prior knowledge of σ^2, and

derive the posterior density of weights given the data, $p(\mathbf{w}|\mathcal{D}, \sigma^2)$ as follows.

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) = \mathcal{N}\big(f(\mathbf{x}_i\,; \mathbf{w}), \sigma^2)\big) \tag{10.15}$$

$$p(\mathcal{D}|\mathbf{w}, \sigma^2) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) \tag{10.16}$$

$$p(\mathbf{w}|\mathcal{D}, \sigma^2) = \frac{p(\mathcal{D}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\sigma^2)}{p(\mathcal{D})} \tag{10.17}$$

We would like to make a prediction at some point $\mathbf{x}_*$ by integrating over the weights $\mathbf{w}$. However, such an integral is intractable, so we resort to approximate predictions.

$$p(y_*|\mathbf{x}_*, \mathcal{D}, \sigma^2) = \int p(y_*|\mathbf{x}_*, \mathbf{w}, \sigma^2)p(\mathbf{w}|\mathcal{D}, \sigma^2)d\mathbf{w} \tag{10.18}$$

$$\approx \frac{1}{m}\sum_{i=1}^{m} p(y_*|\mathbf{x}_*, \mathbf{w}_i, \sigma^2) \tag{10.19}$$

$$\tag{10.20}$$

where $\mathbf{w}_i$ is the i^{th} sample from the posterior distribution with density $p(\mathbf{w}|\mathcal{D}, \sigma^2)$ typically acquired from a numerical method. For example, Neal [48] uses Hamiltonian Monte Carlo (HMC) for this purpose, whereas MacKay [36, 39, 38] uses the Laplace approximation to approximate $p(\mathbf{w}, \sigma^2|\mathcal{D})$ by a Gaussian centred on the most probable parameters.

In the simplest case, let us set a Gaussian prior distribution over $\mathbf{w}$. For a given data set, we can then use HMC to generate m weight samples to make predictions and estimate the expected improvement over the current maximum, f_{max}, at any input point, $\mathbf{x}$.

$$EI_{BNN}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} s_i(\mathbf{x})\left[u_i\,\Phi(u_i) + \phi(u_i)\right] \tag{10.21}$$

where $\Phi(\cdot)$ and $\phi()$ are defined in equation 9.4. $s_i^2(\mathbf{x})$ is the output variance of the i^{th} sample network $\widehat{f}_i(\mathbf{x})$, and $u_i = \frac{\widehat{f}_i(\mathbf{X}) - f_{\text{max}}}{s_i(\mathbf{X})}$.

As an example, consider figure 10.7. The dashed line shows an objective function, sampled at the points shown as dots. The sampled data is modelled with a Bayesian neural network with 16 hidden units. σ was set to 0.01 and Gaussian prior distributions were placed over the weights, such that

$p(w_{ij}) = \mathcal{N}(0, 3^2), p(u_i) = \mathcal{N}(0, 1^2), p(b_j) = \mathcal{N}(0, 3^2), p(c) = \mathcal{N}(0, 1^2)$. The network output is defined by

$$f(x) = c + \sum_{i=1}^{16} u_i \tanh\left(w_i x + b_i\right) \tag{10.22}$$

The weights were sampled using HMC with $\tau = 20, \epsilon = 0.0015$ (see [34] pages 387-390). Ten Markov chains were simulated, and each was run in for 200 iterations after which 5 samples were taking every 15 iterations to give a total of 50 samples. These samples were used to calculate EI_{BNN} which is plotted as the solid black line. The network output for each sample $\widehat{f}_m(x)$ is shown by the grey lines.

Note that the network samples produce outputs that are more variable away from data points. This means the Bayesian neural network is more uncertain in its prediction at these points. The result is that some areas have regions of high expected improvement. In this simple case, the peaks of the expected improvement function are good guides as to where to sample next to make an improvement.

Consider now, using BNN's and EI_{BNN} for optimisation purposes. We optimise the $1D$ function shown as the dashed line in figure 10.8. At each iteration, we use HMC to take samples from the posterior distribution over weights. We use the same HMC settings as above, but only simulate 5 Markov chains, and take 10 samples from each. The 50 samples are used to find the point of maximum expected improvement within the left and right limits of x.

For more complex or higher dimensional optimisation problems the basic BNN optimisation algorithm described above needs some improvement. Firstly, if we do not have prior knowledge of the noise variance, then this would need to be inferred. We could add a parameter representing σ, place a prior distribution over this parameter, and take samples from the new posterior distribution. Alternatively, we could use the method presented by Neal [48] and integrate out this parameter analytically. This is done by setting a Gamma prior distribution over $\tau = \sigma^{-2}$, and integrating to find

$$p(\mathcal{D}|\mathbf{w}) = \int_0^\infty p(\tau) p(\mathcal{D}|\mathbf{w}, \tau) d\tau \tag{10.23}$$

where the integral is analytically tractable.

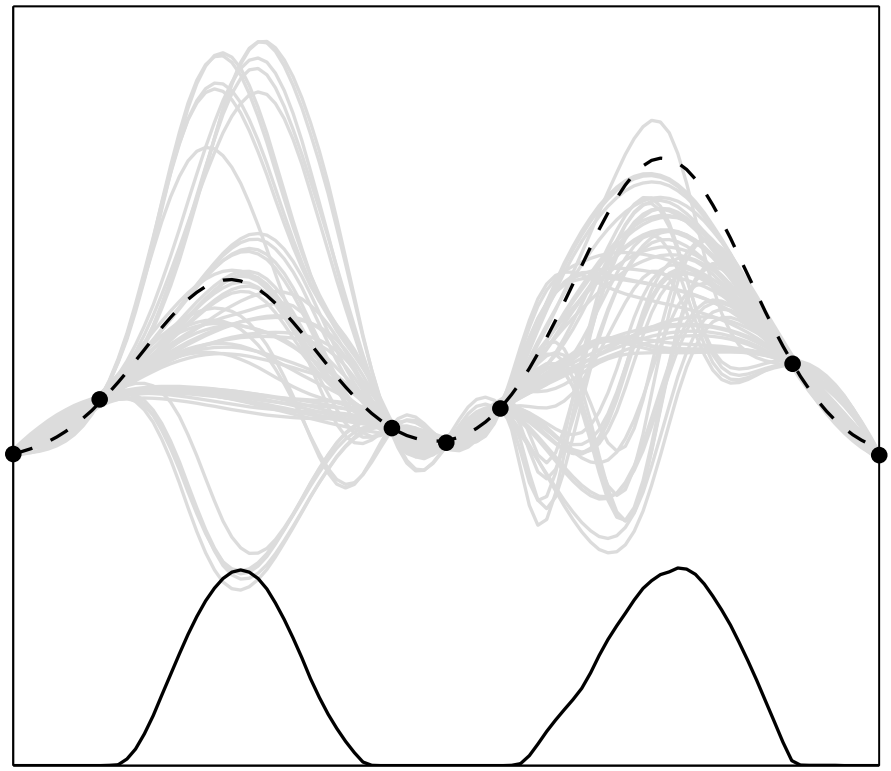

Figure 10.7: Bayesian Neural Network predictions, shown as samples from the posterior distribution over networks (grey lines). The objective function is shown as the dashed line, and has been sampled at the dots to form a data set. The expected improvement calculated from these samples is shown rescaled by the solid black line at the bottom of the window.

Secondly, in many cases we do not have good prior knowledge about the behaviour, or form, of the objective function. If the objective function is relatively simple, (e.g. smooth with a single optimum) then the neural network model might have just a few hidden units, and the weights might have small values. On the other hand, if the objective function is complicated with many local optima, then more hidden units and larger weights will be required to form an adequate neural network model. So without good prior knowledge, it is difficult to decide on how many hidden units are sufficient, and what prior distributions should be set over the weights. One solution to this problem is to examine a range of models and calculate the model evidence for each.

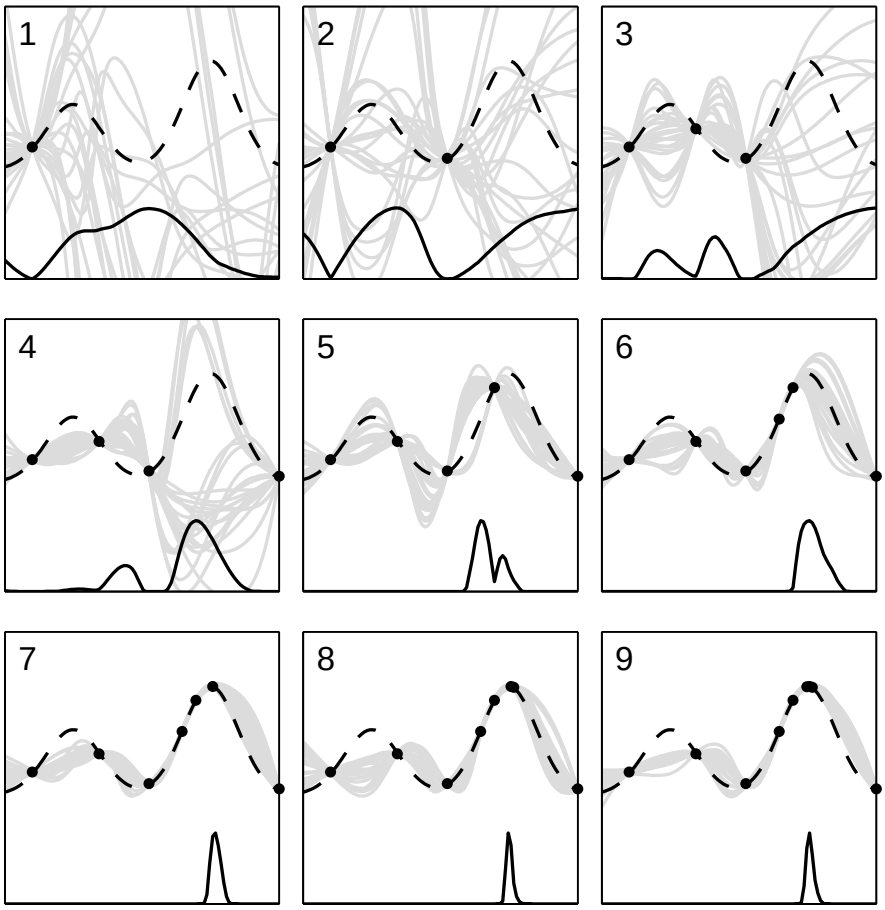

Figure 10.8: Bayesian Neural Network Optimisation applied to a $1D$ function for 9 iterations. The function to be optimised is shown as a dashed line. The sample points are shown as black dots. The predictive mean for all HMC samples from the posterior distribution over networks is shown by the grey lines. The expected improvement is rescaled and shown at the bottom of each window.

We then either use the best model, or weight the predictions and expected improvement by the posterior probability of each model. Alternatively, we might simply "flood" the model with an excessive number of hidden units, set a vague prior distribution over the weights, and let the Bayesian method and the data decide what is best. The latter sounds good in principle, but may be too computationally demanding to implement in practise.

Whether the Bayesian neural networks for optimisation method as described

here is efficient or not remains to be determined. The method is introduced simply as a possible alternative to GPO, and as a method that may be capable of optimising non-stationary objective functions.

10.7 Summary

This chapter has introduced and demonstrated some enhancements to the basic GPO algorithm. We have shown that this algorithm can be used to *efficiently* find a solution to the non-trivial problem of double pole balancing. Additionally, we have shown how the use of Bayesian model comparison methods can lead to improved GPO performance when it is not known *a priori* what the most suitable model is. Furthermore, we have introduced reduced rank GPO, which reduces the computational complexity of the optimisation while maintaining a reasonable performance. Finally, we introduced Bayesian neural networks for optimisation as a possibly alternative to GPO.

Chapter 11

GPs for Gradient and Integral Estimation

11.1 Introduction

Chapters 9 and 10 examined methods to efficiently perform optimisations using Gaussian process models. This chapter introduces and discusses methods to efficiently estimate gradients and integrals of noisy functions using Gaussian process models.

11.2 Gradient Estimation

Consider the situation where we can query a function $f^*(\mathbf{x})$ with any input $\mathbf{x}_i \in \mathbb{R}^D$, and receive back a noisy evaluation $y_i = f^*(\mathbf{x}_i) + \epsilon_i$ where ϵ_i is a zero mean Gaussian random variable with a variance σ^2. By taking a set of samples, $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ we wish to estimate the gradient $\mathbf{g}^*(\mathbf{x}_0) = \frac{\partial f^*(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}_0}$ at a point $\mathbf{x}_0$. If sampling from $f^*(\mathbf{x})$ incurs some cost, then we might wish to minimise the number of samples required to get an acceptable gradient estimate.

11.2.1 Derivative Processes

Consider a Gaussian process over $\mathbb{R}^D$, constructed as the output of a linear filter excited by Gaussian white noise (see section 2.1 page 17). The filter is defined by an impulse response $h(\mathbf{x}), \mathbf{x} \in \mathbb{R}^D$, which is convolved with a noise process $u(\mathbf{x})$ to produce the output Gaussian process $y(\mathbf{x}) = u(\mathbf{x}) * h(\mathbf{x})$. Now define $z_i(\mathbf{x})$ as the partial derivative of $y(\mathbf{x})$ with respect to x_i. Differentiation is a linear operation, so $z_i(\mathbf{x})$ is also a Gaussian process, which is clear from the following:

$$
\begin{aligned}
z_i(\mathbf{x}) &= \frac{\partial y(\mathbf{x})}{\partial x_i} && (11.1)\\
&= \frac{\partial}{\partial x_i}\left\{u(\mathbf{x}) * h(\mathbf{x})\right\} && (11.2)\\
&= \frac{\partial}{\partial x_i}\left\{\int_{-\infty}^{\infty} u(\mathbf{s})h(\mathbf{x}-\mathbf{s})d\mathbf{s}\right\} && (11.3)\\
&= \int_{-\infty}^{\infty} u(\mathbf{s})\left\{\frac{\partial}{\partial x_i}h(\mathbf{x}-\mathbf{s})\right\}d\mathbf{s} && (11.4)\\
&= u(\mathbf{x}) * g_i(\mathbf{x}) && (11.5)
\end{aligned}
$$

where $g_i(\mathbf{x}) = \dfrac{\partial h(\mathbf{x})}{\partial x_i}$ is the impulse response of the filter that produces $z_i(\mathbf{x})$ when excited by the noise process $u(\mathbf{x})$. In other words, we generate the partial derivative process $z_i(\mathbf{x})$ by exciting a linear filter $g_i(\mathbf{x})$ with Gaussian white noise $u(\mathbf{x})$.

By extension, we can generate higher-order and mixed partial-derivatives of the process $y(\mathbf{x})$ by convolving the same input noise sequence with partial derivatives of the impulse response

$$
\frac{\partial^{n+m} y(\mathbf{x})}{\partial x_i^n \partial x_j^m} = u(\mathbf{x}) * \frac{\partial^{n+m} h(\mathbf{x})}{\partial x_i^n \partial x_j^m} \tag{11.6}
$$

assuming that all the partial derivatives exist, which is so if the impulse response $h(\mathbf{x})$ is Gaussian.

The derivative processes, $z_1(\mathbf{x}) \ldots z_D(\mathbf{x})$, are derived from the same noise source as the original process $y(\mathbf{x})$. Therefore, the derivative processes and the original processes form a set of dependent Gaussian processes and we can use the results of chapter 3 for their analysis. In particular, we can find the covariance function between the process and first-derivative processes. To do so, define the covariance function $\mathrm{cov}_{ij}(\mathbf{a}, \mathbf{b})$ as the covariance between $z_i(\mathbf{a})$

and $z_j(\mathbf{b})$, with $i, j > 0$, and $\mathbf{a}, \mathbf{b} \in \mathcal{X}$ (the input space). Furthermore, let the auto-covariance function for the process $y(\mathbf{x})$ be $k(\mathbf{a}, \mathbf{b})$. Therefore,

$$\begin{aligned}
\mathrm{cov}_{ij}(\mathbf{a}, \mathbf{b}) &= \int_{\mathbb{R}^D} g_i(\mathbf{a} - \mathbf{r}) g_j(\mathbf{b} - \mathbf{r}) d^D\mathbf{r} && (11.7)\\
&= \int_{\mathbb{R}^D} \frac{\partial h(\mathbf{a} - \mathbf{r})}{\partial a_i} \frac{\partial h(\mathbf{b} - \mathbf{r})}{\partial b_j} d^D\mathbf{r} && (11.8)\\
&= \frac{\partial^2}{\partial a_i \partial b_j} \left\{ \int_{\mathbb{R}^D} h(\mathbf{a} - \mathbf{r}) h(\mathbf{b} - \mathbf{r}) d^D\mathbf{r} \right\} && (11.9)\\
&= \frac{\partial^2 k(\mathbf{a}, \mathbf{b})}{\partial a_i \partial b_j} && (11.10)
\end{aligned}$$

In a similar fashion, we can show that the covariance function between $y(\mathbf{a})$ and $z_j(\mathbf{b})$ is

$$c_j(\mathbf{a}, \mathbf{b}) = \frac{\partial k(\mathbf{a}, \mathbf{b})}{\partial b_j} \tag{11.11}$$

Equations (11.10) and (11.11) are the same as those presented by Solak *et al.* [73], albeit with different notations.

11.2.2 Gaussian Process Gradient Estimation

For input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, consider the squared-exponential covariance function as a function of input separation $\mathbf{s} = \mathbf{x}_i - \mathbf{x}_j$

$$k(\mathbf{s}) = \exp(v) \, \exp\left(-\tfrac{1}{2}\mathbf{s}^{\mathrm{T}}\mathbf{A}\mathbf{s}\right) + \delta_{ij} \exp(2\beta) \tag{11.12}$$

where $\mathbf{A}$ is a $D \times D$ symmetric, positive definite matrix.

Given data $\mathcal{D}$, and hyperparameters $\boldsymbol{\theta} = \{\mathbf{A}, v, \beta\}$ define $f(\mathbf{x})$ as the mean prediction of a Gaussian process model

$$f(\mathbf{x}) = \mathbf{k}^{\mathrm{T}}(\mathbf{x}) \, \mathbf{C}^{-1} \, \mathbf{y} \tag{11.13}$$

where $\mathbf{k}^{\mathrm{T}}(\mathbf{x}) = [k(\mathbf{x} - \mathbf{x}_1) \ldots k(\mathbf{x} - \mathbf{x}_n)]$ and $\mathbf{y} = [y_1 \ldots y_n]^{\mathrm{T}}$. $\mathbf{C}$ is the covariance matrix built from $\mathcal{D}$ and $k(\mathbf{s})$.

$f(\mathbf{x})$ can be differentiated to find the model gradient at $\mathbf{x}_0$

$$\begin{aligned}
\mathbf{g}(\mathbf{x}_0) &= \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}_0} && (11.14)\\
&= \mathbf{J}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{y} && (11.15)
\end{aligned}$$

where $\mathbf{J}^{\mathrm{T}}$ is the $D \times n$ transpose of the Jacobian

$$\mathbf{J}^{\mathrm{T}} = \left[\frac{\partial k(\mathbf{x}_0 - \mathbf{x}_1)}{\partial \mathbf{x}_0} \;\middle|\; \dots \;\middle|\; \frac{\partial k(\mathbf{x}_0 - \mathbf{x}_n)}{\partial \mathbf{x}_0} \right] \tag{11.16}$$

which has columns defined by differentiating equation (11.12)

$$\frac{\partial k(\mathbf{x}_0 - \mathbf{x}_i)}{\partial \mathbf{x}_0} = -\exp(v)\, \exp\left(-\tfrac{1}{2}(\mathbf{x}_0 - \mathbf{x}_i)^{\mathrm{T}} \mathbf{A} (\mathbf{x}_0 - \mathbf{x}_i)\right) \mathbf{A}(\mathbf{x}_0 - \mathbf{x}_i) \tag{11.17}$$

Differentiation is a linear operation, so $\mathbf{g}(\mathbf{x}_0)$ is a Gaussian process . In particular, $\mathbf{g} = \mathbf{g}(\mathbf{x}_0)$ is a vector with components $g_1 \dots g_D$, with Gaussian statistics defined by a $D \times D$ covariance matrix $\mathbf{B}$, where

$$\mathbf{B} = \begin{bmatrix} \mathrm{cov}(g_1, g_1) & \dots & \mathrm{cov}(g_1, g_D) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(g_D, g_1) & \dots & \mathrm{cov}(g_D, g_D) \end{bmatrix} \tag{11.18}$$

$$= \left. \frac{\partial^2 k(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i \, \partial \mathbf{x}_j} \right|_{\mathbf{x}_i, \mathbf{x}_j = \mathbf{x}_0} \tag{11.19}$$

$$= \exp(v)\mathbf{A} \tag{11.20}$$

The $D \times n$ matrix of the covariance between our n observations and D gradient components is

$$\begin{bmatrix} \mathrm{cov}(y_1, g_1) & \dots & \mathrm{cov}(y_n, g_1) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(y_1, g_D) & \dots & \mathrm{cov}(y_n, g_D) \end{bmatrix} = \left[\frac{\partial k(\mathbf{x}_0,\, \mathbf{x}_1)}{\partial \mathbf{x}_0} \;\middle|\; \dots \;\middle|\; \frac{\partial k(\mathbf{x}_0,\, \mathbf{x}_j)}{\partial \mathbf{x}_0} \right] \tag{11.21}$$

$$= \mathbf{J}^{\mathrm{T}} \tag{11.22}$$

The Gaussian process prior distribution over $\mathbf{y}$ and $\mathbf{g}$ is

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{g} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}\,, \begin{bmatrix} \mathbf{C} & \mathbf{J} \\ \mathbf{J}^{\mathrm{T}} & \mathbf{B} \end{bmatrix} \right) \tag{11.23}$$

Using standard formulae [61], we condition on $\mathcal{D}$ to find the distribution

$$\mathbf{g} \big| \mathcal{D} \sim \mathcal{N}\Big(\mathbf{J}^{\mathrm{T}} \mathbf{C}^{-1} \mathbf{y}\,,\; \mathbf{B} - \mathbf{J}^{\mathrm{T}} \mathbf{C}^{-1} \mathbf{J} \Big) \tag{11.24}$$

where the mean is equal to that in equation (11.15).

In summary, given data $\mathcal{D}$, we can estimate the gradient $\mathbf{g}$ at $\mathbf{x}_0$ and find the covariance matrix given by $\mathbf{B} - \mathbf{J}^{\mathrm{T}} \mathbf{C}^{-1} \mathbf{J}$.

11.2.3 Sample Minimisation

When we estimate $\mathbf{g}$ using a Gaussian process model, equation (11.24) tells us that there is uncertainty in this estimate. Intuitively, we desire a gradient estimate with low uncertainty. This section describes a method to sequentially select sample points so as to systematically reduce the uncertainty in $\mathbf{g}$.

One way to reduce the uncertainty in the gradient estimate is to take a new sample $\mathbf{x}_*$ placed so as to minimise the entropy of $p(\mathbf{g}|\mathcal{D}, \mathbf{x}_*)$. We call this entropy S_* and derive it as follows, noting that the entropy of a multivariate Gaussian probability density with covariance matrix $\mathbf{\Psi}$ is [67]:

$$S = \tfrac{D}{2}\log(2\pi e) + \tfrac{1}{2}\log(|\mathbf{\Psi}|) \tag{11.25}$$

The gradient $\mathbf{g}$, conditional on sampling at $\mathbf{x}_*$ is

$$\mathbf{g}\big|\mathcal{D}, \mathbf{x}_* \;\sim\; \mathcal{N}\Big(\mathbf{J}^{\mathrm{T}}\mathbf{C}^{-1}\,\mathbf{y}\;,\;\mathbf{F}\Big) \tag{11.26}$$

where $\mathbf{F}$ is derived as follows.

Firstly, define $\mathbf{\Sigma}$ as the covariance matrix for the multivariate Gaussian joint probability density function $p(y_1, \ldots, y_n, y_*, g_1, \ldots, g_D)$

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{C} & \mathbf{k} & \mathbf{J} \\ \mathbf{k}^{\mathrm{T}} & \kappa & \mathbf{b}^{\mathrm{T}} \\ \mathbf{J}^{\mathrm{T}} & \mathbf{b} & \mathbf{B} \end{bmatrix}$$

which is the same as the covariance matrix in equation (11.23) augmented by covariance entries for the test point $\mathbf{x}_*$, namely, $\mathbf{k} = \mathbf{k}(\mathbf{x}_*)$, $\kappa = \mathrm{cov}(y_*, y_*) = \exp(v) + \exp(2\beta)$, and

$$\begin{aligned} \mathbf{b} &= \begin{bmatrix} \mathrm{cov}(g_1, \mathbf{x}_*) \\ \vdots \\ \mathrm{cov}(g_D, \mathbf{x}_*) \end{bmatrix} \\ &= \frac{\partial k(\mathbf{x}_0, \mathbf{x}_*)}{\partial \mathbf{x}_0} \\ &= -\exp(v)\,\exp\left(-\tfrac{1}{2}(\mathbf{x}_0 - \mathbf{x}_*)^{\mathrm{T}}\mathbf{A}(\mathbf{x}_0 - \mathbf{x}_*)\right)\mathbf{A}(\mathbf{x}_0 - \mathbf{x}_*) \end{aligned}$$

Define a block partitioning of $\boldsymbol{\Sigma}$ as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$$

where

$$\boldsymbol{\Sigma}_{11} = \begin{bmatrix} \mathbf{C} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & \kappa \end{bmatrix} \qquad \boldsymbol{\Sigma}_{12} = \begin{bmatrix} \mathbf{J} \\ \mathbf{b}^{\mathrm{T}} \end{bmatrix} = \boldsymbol{\Sigma}_{21}^{\mathrm{T}} \qquad \boldsymbol{\Sigma}_{22} = \mathbf{B}$$

Given that the Schur complement [92] is $\boldsymbol{\Sigma}_{2/1} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}$, the covariance matrix $\mathbf{F}$ to fit into equation (11.26) is

$$\mathbf{F} = \mathbf{B} - \begin{bmatrix} \mathbf{J}^{\mathrm{T}} & \mathbf{b} \end{bmatrix} \begin{bmatrix} \mathbf{C} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & \kappa \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{J} \\ \mathbf{b}^{\mathrm{T}} \end{bmatrix}$$

The entropy given $\mathbf{x}_*$ is therefore

$$S_* = \tfrac{D}{2}\log(2\pi e) + \tfrac{1}{2}\log(|\mathbf{F}|) \tag{11.27}$$

We can now attempt to find a $\mathbf{x}_*$ that minimises S_*. Sampling at such a point is expected to maximally decrease the entropy of the gradient distribution. Hence we will maximally increase our "certainty" about the model gradient at $\mathbf{x}_0$. If the model is accurate, then we expect the gradient $\mathbf{g}$ to be a good approximation of the problem gradient $\mathbf{g}^*(\mathbf{x}_0)$.

A simple illustrative example is shown in figure 11.1. The top panel shows a Gaussian process model of 4 samples, taken from a function with additive Gaussian noise. The bottom panel shows the conditional entropy of the estimate of the gradient at $x = 0$, as a function of the next sample point x_*. The entropy has two major local minima, either side of $x_* = 0$, indicating the best places to take the next sample. These minima are neither too close nor too far from the point at which the gradient is to be estimated. This makes sense, as a sample that is too close cannot offer much information about the gradient because it is easily swamped by noise. A point that is too far away 'loses sight' of the local trend, and cannot contribute either. The most valuable point for contributing to the gradient estimate lies somewhere in between the two extremes, and ultimately depends on the model prior distributions and training data.

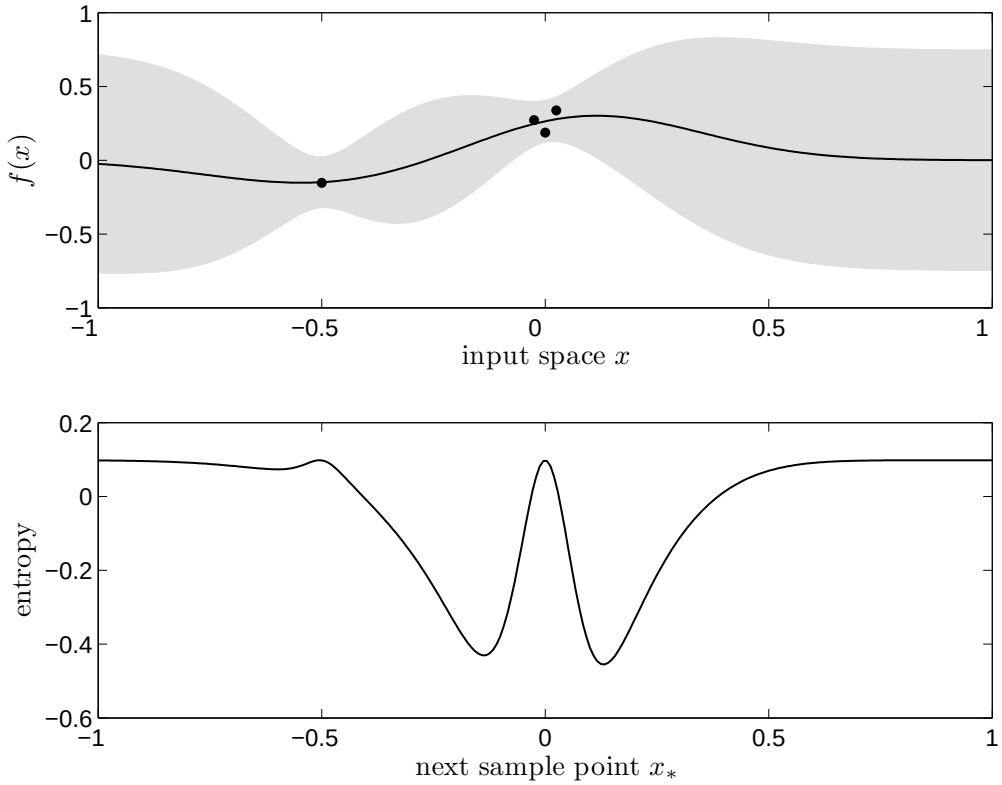

Figure 11.1: (Top) Gaussian process model (solid line) of 4 samples (dots) from a noisy function. The 95% confidence interval is shaded. (Bottom) Conditional entropy of the model's estimate of the gradient at $x = 0$, as a function of the next sample point.

11.2.4 Gradient of Entropy

Note that we can differentiate (11.27) with respect to the j^{th} component of $\mathbf{x}_*$ to aid in the minimisation.

$$\frac{\partial S_*}{\partial \mathbf{x}_*^j} = \tfrac{1}{2} \operatorname{Tr}\left(\mathbf{F}^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{x}_*^j}\right) \tag{11.28}$$

where

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}_*^j} = -\frac{\partial \boldsymbol{\Sigma}_{21}}{\partial \mathbf{x}_*^j}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\frac{\partial \boldsymbol{\Sigma}_{11}}{\partial \mathbf{x}_*^j}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\frac{\partial \boldsymbol{\Sigma}_{21}}{\partial \mathbf{x}_*^j}$$

To further reduce this, observe that

$$\frac{\partial \boldsymbol{\Sigma}_{21}}{\partial \mathbf{x}_*^j} = \begin{bmatrix} \dfrac{\partial \mathbf{J}^{\mathrm{T}}}{\partial \mathbf{x}_*^j} & \dfrac{\partial \mathbf{b}}{\partial \mathbf{x}_*^j} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{D\times n} & \dfrac{\partial \mathbf{b}}{\partial \mathbf{x}_*^j} \end{bmatrix}$$

where $\frac{\partial \mathbf{b}}{\partial \mathbf{x}_*^j}$ is the j^{th} column of $\frac{\partial \mathbf{b}}{\partial \mathbf{x}_*}$

$$\frac{\partial \mathbf{b}}{\partial \mathbf{x}_*} = \left[\frac{\partial \mathbf{b}}{\partial \mathbf{x}_*^1} \;\middle|\; \cdots \;\middle|\; \frac{\partial \mathbf{b}}{\partial \mathbf{x}_*^D} \right]$$

$$= \exp(v) \exp\left(-\tfrac{1}{2}(\mathbf{x}_0 - \mathbf{x}_*)^{\mathrm{T}} \mathbf{A} (\mathbf{x}_0 - \mathbf{x}_*)\right) \times \left[\mathbf{A} - \mathbf{A}(\mathbf{x}_0 - \mathbf{x}_*)(\mathbf{x}_0 - \mathbf{x}_*)^{\mathrm{T}} \mathbf{A}\right]$$

Finally,

$$\frac{\partial \mathbf{\Sigma}_{11}}{\partial \mathbf{x}_*^j} = \begin{bmatrix} \mathbf{0}_{n \times n} & \frac{\partial \mathbf{k}}{\partial \mathbf{x}_*^j} \\ \frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}_*^j} & 0 \end{bmatrix}$$

where

$$\frac{\partial \mathbf{k}}{\partial \mathbf{x}_*^j} = j^{th} \text{ column of } \mathbf{J}$$

11.2.5 Gradient Estimation Algorithm

We can use algorithm 11.1 to estimate the gradient $\mathbf{g}^*(\mathbf{x}_0)$, using n samples from $f^*(\mathbf{x})$ by modelling $f^*(\mathbf{x})$ with a Gaussian process model parameterised by $\boldsymbol{\theta}$. We specify a prior density $p(\boldsymbol{\theta})$ which represents our prior beliefs about $\boldsymbol{\theta}$. For example, the covariance function in equation (11.12) has hyperparameters $\boldsymbol{\theta} = \{v, \beta, \alpha_1, \ldots \alpha_D\}$ with $\mathbf{A} = \mathrm{diag}(\alpha_1 \ldots \alpha_D)$. In this case we might specify a Gaussian prior distribution for $\boldsymbol{\theta}$.

Figure 11.2 shows the results of estimating the gradient at a point for a 6 dimensional hyperelliptical Gaussian surface, using the gradient estimation algorithm (Algorithm 11.1). Samples of the surface were corrupted by Gaussian noise with $\sigma = 0.02$, and the surface range was bounded to $[0, 1]$.

11.3 GPs for Integral Estimation

We have seen how Gaussian process models can be used to efficiently estimate the gradient of a noisy function. Here, we briefly examine how we can do a similar thing with definite integral estimation.

Algorithm 11.1: Estimate gradient of $f^*(\mathbf{x})$ at $\mathbf{x}_0$ using Gaussian process model over n samples.

Input: $f^*(\cdot)$, $\mathbf{x}_0$, n, $p(\boldsymbol{\theta})$

$y_0 \leftarrow f^*(\mathbf{x}_0)$;
for $i = 1$ *to* n **do**
  $\mathcal{D} \leftarrow \{(\mathbf{x}_0, y_0) \dots (\mathbf{x}_{i-1}, y_{i-1})\}$;
  $\boldsymbol{\theta}_{max} \leftarrow \arg\max_{\theta} \log p(\boldsymbol{\theta}|\mathcal{D})$;
  $\mathbf{x}_i \leftarrow \arg\min_{x_*} S_*(\mathbf{x}_*|\mathcal{D}, \boldsymbol{\theta}_{max})$;
  $y_i \leftarrow f^*(\mathbf{x}_i)$;
$\mathbf{y} \leftarrow \{y_1 \dots y_n\}$;
$\mathbf{g} \leftarrow \mathbf{J}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{y}$;
return $\mathbf{g}$

11.3.1 GPs for Definite Integration over Rectangles

Consider a Gaussian process $F(x)$ generated as the output of a filter with impulse response $H(x)$, when stimulated with Gaussian white noise $w(x)$ as in section 2.1. Call $F(x)$ the antiderivative process, and take the derivative to find the process $f(x)$, as follows:

$$f(x) = \frac{dF(x)}{dx} = \frac{d}{dx}\{w(x) * H(x)\} \tag{11.29}$$

$$= w(x) * H'(x) \tag{11.30}$$

$$= w(x) * h(x) \tag{11.31}$$

where $H'(x)$ is the derivative of the impulse response $H(x)$.

Consider an interval $[x_l, x_r]$ over which we wish to integrate $f(x)$. We know the antiderivative of $f(x)$ is $F(x)$ so by the fundamental theorem of calculus the integral is

$$I = \int_{x_l}^{x_r} f(x)dx = F(x_r) - F(x_l) \tag{11.32}$$

F is a Gaussian process, so the integral I is a Gaussian random variable of which we wish to find the mean and variance.

$F(x)$ and $f(x)$ are generated from same the noise process $w(x)$, so form a set of two dependent Gaussian processes. Further, we assume Gaussian noise is

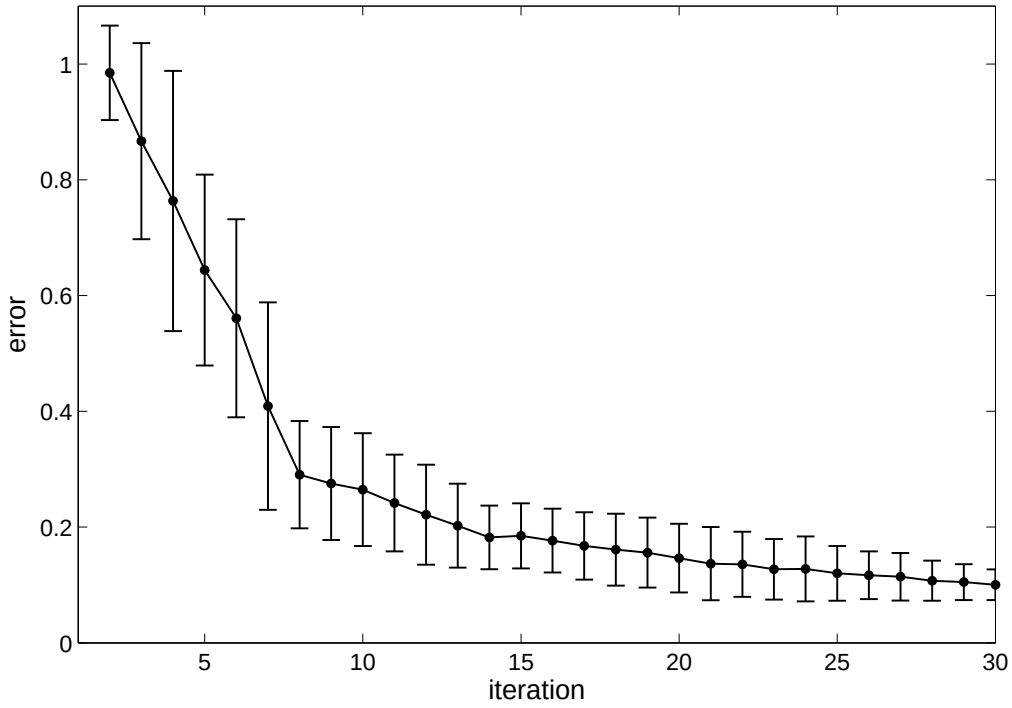

Figure 11.2: Six dimensional gradient estimation from a Gaussian process model. 30 samples were taken and the figure shows the error at each iteration, which is the ratio of the magnitude of the error vector to the magnitude of the actual gradient vector. 20 runs were averaged to produce the solid line and 1σ error bars.

---

added to the processes $F(x)$ and $f(x)$ with variances σ_F^2 and σ_f^2 respectively. Using the result in appendix A.1 the covariance functions for inputs x_i and x_j, separated by $s = x_j - x_i$ are

$$\text{cov}(F(x_i), F(x_j)) = H(s) * H(-s) + \sigma_F^2 \delta_{ij} \tag{11.33}$$

$$\text{cov}(F(x_i), f(x_j)) = H(s) * h(-s) \tag{11.34}$$

$$\text{cov}(f(x_i), f(x_j)) = h(s) * h(-s) + \sigma_f^2 \delta_{ij} \tag{11.35}$$

This means that if one observes $f(x)$ at n points to create a vector of observations $\mathbf{y}$ (where $y_i \sim \mathcal{N}(f(x_i), \sigma_f^2)$), then one can construct a covariance matrix $\mathbf{\Sigma}_*$ for the joint Gaussian distribution $p(\mathbf{y}, y_*, F(x_l), F(x_r))$. The scalars $F(x_l)$ and $F(x_r)$ are the (unobserved) values of the antiderivative at the limits x_l and x_r. The (yet unobserved) value of the process at a test point x_* is y_*. The

covariance matrix is:

$$\mathbf{\Sigma}_* = \begin{bmatrix} \mathbf{C} & \mathbf{k} & \mathbf{a} \\ \mathbf{k}^{\mathrm{T}} & \kappa & \mathbf{b}^{\mathrm{T}} \\ \mathbf{a}^{\mathrm{T}} & \mathbf{b} & \mathbf{B} \end{bmatrix} \tag{11.36}$$

where the $(i,j)^{th}$ element of $\mathbf{C}$ is $C_{ij} = \mathrm{cov}(f(x_i), f(x_j))$. The vector $\mathbf{k}$ has as its j^{th} element $k_j = \mathrm{cov}(f(x_j), f(x_*))$. The scalar $\kappa = \mathrm{cov}(f(x_*), f(x_*))$. The j^{th} row of the $n \times 2$ matrix $\mathbf{a}$ is $[\mathrm{cov}(f(x_j), F(x_l)) \quad \mathrm{cov}(f(x_j), F(x_r))]$. The vector $\mathbf{b} = [\mathrm{cov}(F(x_l), f(x_*)) \quad \mathrm{cov}(F(x_r), f(x_*)]^{\mathrm{T}}$, and the matrix

$$\mathbf{B} = \begin{bmatrix} \mathrm{cov}(F(x_l), F(x_l)) & \mathrm{cov}(F(x_l), F(x_r)) \\ \mathrm{cov}(F(x_r), F(x_l)) & \mathrm{cov}\left(F(x_r), F(x_r)\right) \end{bmatrix} \tag{11.37}$$

Using standard formulae for conditioning on a multivariate Gaussian [61], the predictive distribution for $F(x_l)$ and $F(x_r)$ conditioned on $\mathbf{y}$ and x_* is Gaussian

$$\begin{bmatrix} F(x_l) \\ F(x_r) \end{bmatrix} \Bigg| \, \mathbf{y}, x_* \quad \sim \quad \mathcal{N}\left(\mathbf{m}(x_*)\,,\, \mathbf{V}(x_*)\right) \tag{11.38}$$

where $\mathbf{m}(x_*)$ is the mean and $\mathbf{V}(x_*)$ is the covariance matrix

$$\mathbf{V}(x_*) = \mathbf{B} - \begin{bmatrix} \mathbf{a}^{\mathrm{T}} & \mathbf{b} \end{bmatrix} \begin{bmatrix} \mathbf{C} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & \kappa \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{a} \\ \mathbf{b}^{\mathrm{T}} \end{bmatrix} \tag{11.39}$$

which is dependent on the test input x_*, but not the test output observation y_*. That is, we can calculate $\mathbf{V}(x_*)$ before observing y_*.

The integral $I = F(x_r) - F(x_l)$, and has a conditional distribution

$$I|\mathbf{y}, x_* \sim \mathcal{N}\big(\mathbf{z}^{\mathrm{T}}\mathbf{m}(x_*), \mathbf{z}^{\mathrm{T}}\mathbf{V}(x_*)\mathbf{z}\big) \tag{11.40}$$

where $\mathbf{z} = [-1 \quad 1]^{\mathrm{T}}$. The entropy of this can be minimised by minimising the variance $\mathbf{z}^{\mathrm{T}}\mathbf{V}(x_*)\mathbf{z}$. So, to minimise the entropy of the integral estimate I, we take the next sample point at the x_* that minimises $\mathbf{z}^{\mathrm{T}}\mathbf{V}(x_*)\mathbf{z}$. Once the next sample has been taken (y_* is observed) we can calculate the predictive mean

$$m(\mathbf{x}_*) = \begin{bmatrix} \mathbf{a}^{\mathrm{T}} & \mathbf{b} \end{bmatrix} \begin{bmatrix} \mathbf{C} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & \kappa \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \tag{11.41}$$

The concept is illustrated by the simple example in figures 11.3 and 11.4. This example estimates the integral over the interval $[-2.5, 2.5]$ using a Gaussian process model, and selects samples at each iteration by finding the point of minimum entropy. The top panel of figure 11.3 shows the situation after 3 samples, where the model is rather vague over the interval. The bottom panel shows the entropy of the estimate given the next sample point, which has a minimum near the left-most integration limit. The situation after 10 samples is shown at the top of figure 11.4 where the model is more confident over the integration interval. In the bottom panel, it can be seen that the minimum entropy point still lies within the integration interval, even though there is large model uncertainty outside the interval. Additionally, at no stage does the algorithm select sample points too far out of the integration interval. Intuitively, this is to be expected as points outside the interval do not contribute to the value of the integral.

There are two problems with estimating the integral in this way. Firstly, $H(x)$, which is the antiderivative of $h(x)$, must correspond to a stable linear filter's impulse response. This is a problem if $h(x)$ is Gaussian, meaning $H(x)$ is the error-function, which is not a finite energy impulse response. Instead, we must set $H(x)$ as the Gaussian, which forces the process we are interested in to have a covariance function that may not be desirable. For example, if $H(x) = \exp(-\frac{1}{2}x^2)$ then the covariance function for $f(x)$ is

$$\mathrm{cov}(\tau) = -\frac{1}{4}\sqrt{\pi}(\tau^2 - 2)\exp(-\frac{1}{4}\tau^2) \tag{11.42}$$

which may or may not be useful depending on the underlying function to be modelled.

The second problem is that the estimation of the integral gets more complicated as the dimensionality of the input space grows. Consider estimating the integral over a rectangular region when the input space has dimensionality $D = 2$. The process in question is $f(\mathbf{x})$ where $\mathbf{x} = [x_1 \ x_2]^{\mathrm{T}}$. Define the processes $F(x_1, x_2)$ and $G(x_1, x_2)$, which are related to $f(\mathbf{x})$ as follows:

$$G(x_1, x_2) = \frac{\partial F(x_1, x_2)}{\partial x_2} \tag{11.43}$$

$$f(\mathbf{x}) = \frac{\partial G(x_1, x_2)}{\partial x_1} \tag{11.44}$$

$$f(\mathbf{x}) = \frac{\partial^2 F(x_1, x_2)}{\partial x_1 \partial x_2} \tag{11.45}$$

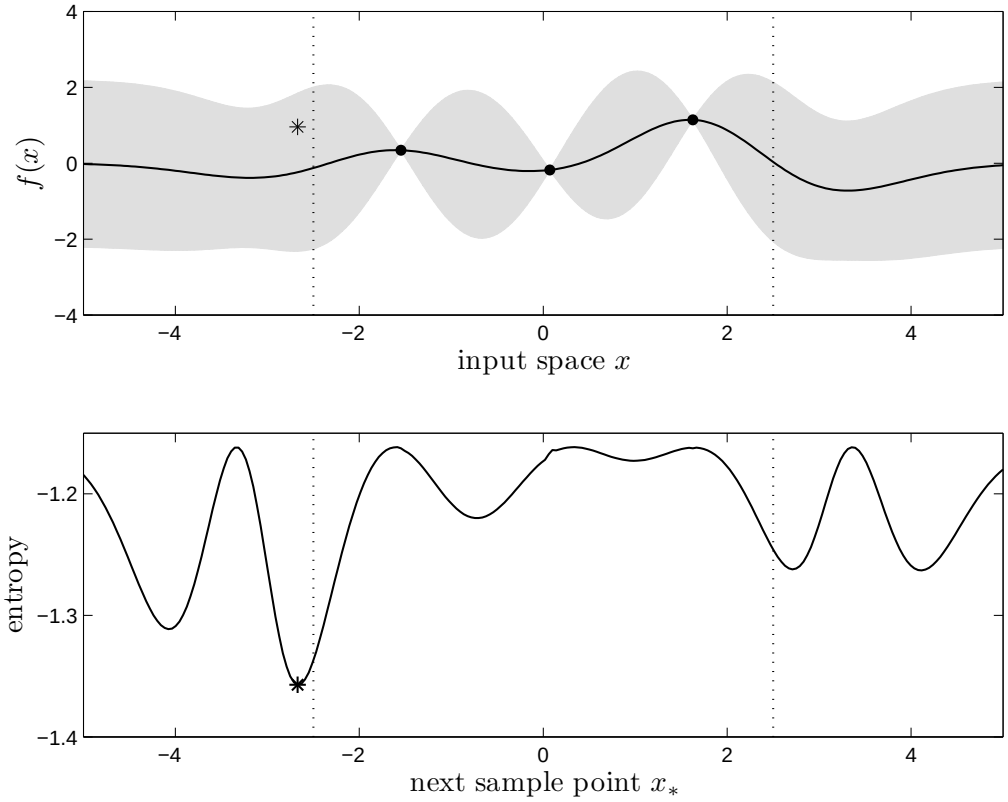

Figure 11.3: (Top) Gaussian process model of 3 training points (dots) showing mean prediction (solid line) with 95% confidence intervals. (Bottom) Conditional entropy of the model's estimate of the integral over $[-2.5, 2.5]$ (dotted line) as a function of the next sample point. The point of minimum entropy determines the next sample point (star) which will be sampled and added to the training set.

The impulse response for the filter that generates $f(\mathbf{x})$ is $\dfrac{\partial^2 H(\mathbf{x})}{\partial x_1 \partial x_2}$, where $H(\mathbf{x})$ is the impulse response for the filter generating the antiderivative process $F(\mathbf{x})$.

$f(\mathbf{x})$ can be integrated over a rectangular region $\mathcal{R}$ with limits $a_i \leq x_i \leq b_i$ using iterated integration

$$\int_{a_2}^{b_2}\int_{a_1}^{b_1} f(\mathbf{x})dx_1dx_2 = \int_{a_2}^{b_2} [G(b_1, x_2) - G(a_1, x_2)]\, dx_2 \tag{11.46}$$

$$= F(b_1, b_2) - F(b_1, a_2) - F(a_1, b_2) + F(a_1, a_2) \tag{11.47}$$

which means $\mathbf{V}(\mathbf{x}_*)$ is a 4×4 matrix. In general, $\mathbf{V}(\mathbf{x}_*)$, will be a $2^D \times 2^D$ matrix, which presents a problem if for example $D = 16$. However, this method may be quite useful for low dimensional integrals especially if we note that

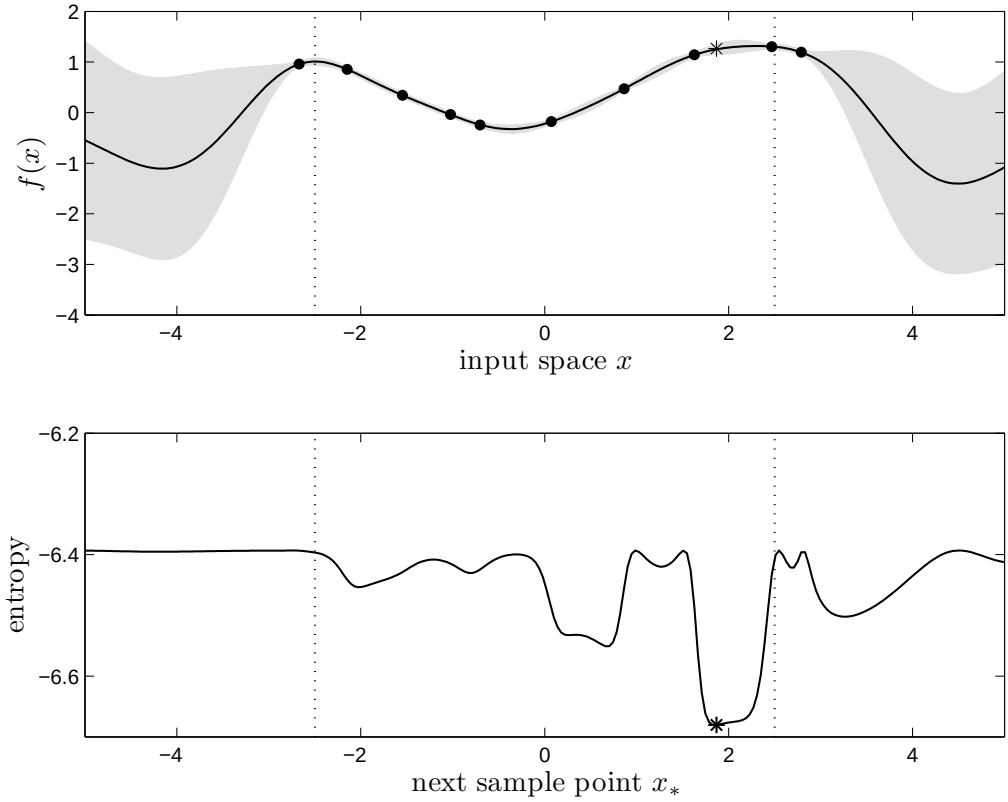

Figure 11.4: (Top) Gaussian process model of 10 training points (dots) showing mean prediction (solid line) with 95% confidence intervals. (Bottom) Conditional entropy of the model's estimate of the integral over $[-2.5, 2.5]$ (dotted line) as a function of the next sample point. The point of minimum entropy determines the next sample point (star) which will be sampled and added to the training set.

---

in equation (11.39), $\mathbf{a}$ and $\mathbf{B}$ remain fixed for varying $\mathbf{x}_*$ and can be precomputed. The vector $\mathbf{b}$ has 2^D elements and each of these change as $\mathbf{x}_*$ varies. The vector $\mathbf{k}$ varies but it only has N elements, regardless of D. For example, if $D = 8$ then $\mathbf{B}$ is a 256×256 matrix, and $\mathbf{a}$ is a $256 \times N$ matrix, which can easily be precomputed. We then find $\mathbf{x}_*$ to minimise the entropy, which requires recalculating $\mathbf{k} \in \mathbb{R}^N, \kappa \in \mathbb{R}$ and $\mathbf{b} \in \mathbb{R}^{256}$ many times.

It remains to be determined if this method would actually prove useful and estimate integrals efficiently. It is presented here simply to introduce the concept and provide another example of Gaussian process regression being used to solve other machine learning problems.

11.4 Summary

This chapter has introduced mechanisms for estimating gradients and (definite) integrals of noisy functions using Gaussian process models. Both methods use an iterative method, where samples are taken at input points that minimise the uncertainty in the resulting estimate. The methods are presented with no claims as to their efficiency, but simply as examples of how Gaussian process regression can be applied to other areas of machine learning, in this case, active learning.

Chapter 12

Conclusions

In general, this thesis has introduced some extensions to the Gaussian processes for regression methodology, and shown how those extensions can be applied to other areas in machine learning.

Chapter 2 introduced an alternative view of Gaussian processes and their specification. Instead of parameterising a covariance function in such a way as to ensure that it is positive definite, it was shown that a Gaussian process can be constructed by parameterising a stable, linear filter. Doing so automatically implies a positive definite covariance function that can be found by performing a convolution integral. The limitation is that the filter must be stable, hence defined by an impulse response that is absolutely integrable. One caveat is that it is not possible to perform the convolution integral for *all* stable filters. This chapter also extended the linear filtering framework to define discrete time and discrete space Gaussian processes, generated by exciting digital filters with noise.

Chapter 3 extended the linear filtering method, introducing and demonstrating a method to build sets of dependent Gaussian processes. It was shown that a set of dependent Gaussian processes is generated by stimulating an analog or digital multiple-input multiple-output filter with Gaussian white noise sources. Doing so automatically implies auto and cross covariance functions that result in positive definite covariance models. This overcomes a previous difficulty with multiple output Gaussian process models, where it has been difficult to define valid cross-covariance functions.

Chapter 4, discussed the problem of system identification. Although system

identification is a well studied problem, this chapter is the first demonstration of how dependent Gaussian processes can be used for this purpose.

Chapter 5 reviewed the reduced rank approximation for lowering the computational cost of Gaussian process regression. This chapter also provided an extension, using the same underlying theory, enabling reduced-rank non-stationary Gaussian processes to be constructed.

In order to reduce the computational cost of dependent Gaussian process modelling, chapter 6 extended the reduced rank method to the case of dependent Gaussian processes by introducing reduced-rank dependent Gaussian processes. Furthermore, this work was extended to define non-stationary reduced-rank dependent Gaussian processes.

Chapter 7 reviewed some existing methods for parameterising arbitrary positive definite covariance matrices, hence providing a mechanism to build rotated and stretched covariance functions.

Chapter 8 extended *annealed importance sampling* by introducing the *sequential annealed importance sampling* method for calculating Bayesian evidence in an on-line fashion. This method allows one to update an evidence estimate with the arrival of new data, and was demonstrated on some toy problems. Furthermore, a new heuristic was described allowing the automatic construction of an annealing schedule for use in these methods.

Chapter 9 reviewed an algorithm, Gaussian process optimisation, for the efficient solution of continuous optimisation problems. At each iteration, the algorithm builds a Gaussian process model of all the training data. The model is used to select the next iterate by finding the input that has the greatest expected improvement over the best solution examined thus far. Problems with this algorithm were discussed, in particular, the problem of optimisation objective functions that have their main features rotated relative to the axes of the covariance function.

Chapter 10 extended the basic Gaussian process optimisation algorithm by using more complicated covariance function parameterisations, capable of modelling features not aligned to the coordinate axes. It was shown that these more complex models could optimise objective functions with rotated features more quickly than the basic, simple algorithm. However, the complex algorithm performed less well than the simple algorithm on an axis-aligned

objective function. Next, the algorithm was extended to use multiple underlying Gaussian process models, with the optimisation being guided by an evidence weighted combination of these. The results showed that using the evidence created a more general purpose algorithm, capable of efficiently solving both the simple *and* complex problems. Finally, a method was described that reduces the computational cost of Gaussian process optimisation by using reduced rank Gaussian processes.

Finally, chapter 11 introduced two more applications of Gaussian process regression. Firstly, it was shown how Gaussian process models can be used to efficiently estimate the gradient of a noisy function. Secondly, a method was introduced to efficiently estimate definite integrals using an underlying Gaussian process model.

Overall, this thesis has presented many new ideas, and introduced some new algorithms based on these. In many cases, the new algorithms would be better described as "pseudo-algorithms", or "algorithm skeletons", as this thesis has not fully developed or rigorously tested them against existing best methods. Instead, where we have introduced a new method, we have simply presented the background theory and then "proved the concept" with a simple demonstration on a toy problem. Therefore, one criticism of this work is that it fails to formally compare the new algorithms against current benchmarks. This is something that must be done, if these algorithms are to develop into trusted methods. However, we have left this work for the future, focusing here instead on the exploration of new ideas.

Appendix A

Dependent GP Covariance Function Derivations

A.1 Auto-Covariance and Cross-Covariance Functions

Consider M independent, stationary, Gaussian white noise processes, $x_1(\mathbf{s}) \dots x_M(\mathbf{s})$, $\mathbf{s} \in \mathbb{R}^D$, producing N-outputs, $y_1(\mathbf{s}) \dots y_N(\mathbf{s})$, with the n^{th} defined as follows:

$$y_n(\mathbf{s}) = u_n(\mathbf{s}) + w_n(\mathbf{s})$$

where $w_n(\mathbf{s})$ is stationary Gaussian white noise, and $u_n(\mathbf{s})$ is defined by a sum of convolutions:

$$\begin{aligned} u_n(\mathbf{s}) &= \sum_{m=1}^{M} h_{mn}(\mathbf{s}) * x_m(\mathbf{s}) \\ &= \sum_{m=1}^{M} \int_{\mathbb{R}^D} h_{mn}(\boldsymbol{\alpha}) x_m(\mathbf{s} - \boldsymbol{\alpha}) d^D\boldsymbol{\alpha} \end{aligned}$$

where h_{mn} is the kernel connecting latent input m to output n.

The function $\mathrm{cov}_{ij}^{y}(\mathbf{s}_a, \mathbf{s}_b)$ defines the auto $(i = j)$ and cross covariance $(i \neq j)$ between $y_i(\mathbf{s}_a)$ and $y_j(\mathbf{s}_b)$, and is derived as follows:

$$\mathrm{cov}_{ij}^{y}(\mathbf{s}_a, \mathbf{s}_b) = \mathrm{cov}_{ij}^{u}(\mathbf{s}_a, \mathbf{s}_b) + \sigma_i^2 \delta_{ij}\, \delta_{ab}$$

where σ_i^2 is the variance of $w_i(\mathbf{s})$, and

$$\begin{aligned}
\mathrm{cov}_{ij}^{u}(\mathbf{s}_a, \mathbf{s}_b) &= E\left\{u_i(\mathbf{s}_a)u_j(\mathbf{s}_b)\right\} \qquad (u_i(\mathbf{s}), u_j(\mathbf{s}) \text{ are zero mean processes}) \\
&= E\left\{\sum_{m=1}^{M}\int_{\mathbb{R}^D} h_{mi}(\boldsymbol{\alpha})x_m(\mathbf{s}_a-\boldsymbol{\alpha})d^D\boldsymbol{\alpha}\sum_{n=1}^{M}\int_{\mathbb{R}^D} h_{nj}(\boldsymbol{\beta})x_n(\mathbf{s}_b-\boldsymbol{\beta})d^D\boldsymbol{\beta}\right\} \\
&= \sum_{m=1}^{M}\sum_{n=1}^{M}\int_{\mathbb{R}^D}\int_{\mathbb{R}^D} h_{mi}(\boldsymbol{\alpha})h_{nj}(\boldsymbol{\beta})E\left\{x_m(\mathbf{s}_a-\boldsymbol{\alpha})x_n(\mathbf{s}_b-\boldsymbol{\beta})d^D\boldsymbol{\alpha}\, d^D\boldsymbol{\beta}\right\}
\end{aligned}$$

where we have changed the order of expectation and integration because $\int_{\mathbb{R}^D}|h_{mn}(\mathbf{s})|^2 d^D s < \infty \quad \forall m,n,$ i.e. $h_{mn}(\mathbf{s})$ are finite energy kernels (corresponding to stable linear filters).

Now, $x_m(\mathbf{s}_1)$ and $x_m(\mathbf{s}_2)$ are Gaussian random variables that are independent unless $m = n$ and $\mathbf{s}_1 = \mathbf{s}_2$, so

$$\begin{aligned}
\mathrm{cov}_{ij}^{u}(\mathbf{s}_a, \mathbf{s}_b) &= \sum_{m=1}^{M}\int_{\mathbb{R}^D}\int_{\mathbb{R}^D} h_{mi}(\boldsymbol{\alpha})h_{nj}(\boldsymbol{\beta})\delta\big(\boldsymbol{\alpha}-[\mathbf{s}_a-\mathbf{s}_b+\boldsymbol{\beta}]\big)d^D\boldsymbol{\alpha}\, d^D\boldsymbol{\beta} \\
&= \sum_{m=1}^{M}\int_{\mathbb{R}^D} h_{mj}(\boldsymbol{\beta})h_{mi}(\boldsymbol{\beta}+(\mathbf{s}_a-\mathbf{s}_b))d^D\boldsymbol{\beta}
\end{aligned}$$

which is the sum of kernel correlations.

If the kernels are stationary, then we can define a stationary $\mathrm{cov}_{ij}^{u}(\cdot)$ in terms of a separation vector $\mathbf{d}_s = \mathbf{s}_a - \mathbf{s}_b$.

$$\mathrm{cov}_{ij}^{u}(\mathbf{d}_s) = \sum_{m=1}^{M}\int_{\mathbb{R}^D} h_{mj}(\boldsymbol{\beta})h_{mi}(\boldsymbol{\beta}+\mathbf{d}_s)d^D\boldsymbol{\beta} \tag{A.1}$$

A.2 Covariance functions for Gaussian Kernels

Equation A.1 defines the auto and cross covariance for the outputs $y_1(\mathbf{s}) \ldots y_n(\mathbf{s})$. Here, we set the kernels to parameterised Gaussians and perform the integral.

Let

$$h_{mn}(\mathbf{s}) = v_{mn}\exp\left(-\frac{1}{2}(\mathbf{s}-\boldsymbol{\mu}_{mn})^{\mathrm{T}}\mathbf{A}_{mn}(\mathbf{s}-\boldsymbol{\mu}_{mn})\right)$$

where $v_{mn} \in \mathbb{R}$, $\mathbf{s}, \boldsymbol{\mu}_{mn} \in \mathbb{R}^D$, and $\mathbf{A}_{mn}$ is a $D \times D$ positive definite matrix.

Now let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^D$ and $\mathbf{A}, \mathbf{B}$ be $D \times D$ positive definite matrices. Define

$$\begin{aligned} f(\mathbf{s}, \mathbf{a}, \mathbf{b}, \mathbf{A}, \mathbf{B}) &= \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}(\mathbf{s}-\mathbf{a})^{\mathrm{T}}\mathbf{A}(\mathbf{s}-\mathbf{a})\right) \exp\left(-\frac{1}{2}(\mathbf{s}-\mathbf{b})^{\mathrm{T}}\mathbf{B}(\mathbf{s}-\mathbf{b})\right) d^D\mathbf{s} \\ &= \exp\left(-\frac{1}{2}c\right) \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}(\mathbf{s}-\boldsymbol{\mu})^{\mathrm{T}}\mathbf{G}(\mathbf{s}-\boldsymbol{\mu})\right) d^D\mathbf{s} \\ &= \exp\left(-\frac{1}{2}c\right) \frac{(2\pi)^{\frac{D}{2}}}{\sqrt{|\mathbf{G}|}} \end{aligned}$$

where $\mathbf{G} = \mathbf{A} + \mathbf{B}$, $\boldsymbol{\mu} = \mathbf{G}^{-1}(\mathbf{A}\mathbf{a} + \mathbf{B}\mathbf{b})$, and

$$\begin{aligned} c &= \mathbf{a}^{\mathrm{T}}\mathbf{A}\mathbf{a} + \mathbf{b}^{\mathrm{T}}\mathbf{B}\mathbf{b} - \boldsymbol{\mu}^{\mathrm{T}}\mathbf{G}\boldsymbol{\mu} \\ &= (\mathbf{b}-\boldsymbol{\epsilon})^{\mathrm{T}}\boldsymbol{\Sigma}(\mathbf{b}-\boldsymbol{\epsilon}) + g \end{aligned}$$

where $\boldsymbol{\Sigma} = \mathbf{A} - \mathbf{A}\mathbf{G}^{-1}\mathbf{A} = \mathbf{A}\mathbf{G}^{-1}\mathbf{B}$ and $\boldsymbol{\epsilon} = \boldsymbol{\Sigma}^{-1}\mathbf{B}\mathbf{G}^{-1}\mathbf{A}\mathbf{a} = \mathbf{a}$ and $g = -\boldsymbol{\epsilon}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{\epsilon} + \mathbf{a}^{\mathrm{T}}\mathbf{A}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{A}\mathbf{G}^{-1}\mathbf{A}\mathbf{a} = 0$ so,

$$f(\mathbf{s}, \mathbf{a}, \mathbf{b}, \mathbf{A}, \mathbf{B}) = \frac{(2\pi)^{\frac{D}{2}}}{\sqrt{|\mathbf{A}+\mathbf{B}|}} \exp\left(-\frac{1}{2}(\mathbf{b}-\mathbf{a})^{\mathrm{T}}\boldsymbol{\Sigma}\,(\mathbf{b}-\mathbf{a})\right)$$

We can now write

$$\begin{aligned} \mathrm{cov}_{ij}^{u}(\mathbf{d}_s) &= \sum_{m=1}^{M} \int_{\mathbb{R}^D} \left\{ v_{mj} \exp\left(-\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\mu}_{mj})^{\mathrm{T}}\mathbf{A}_{mj}(\boldsymbol{\beta}-\boldsymbol{\mu}_{mj})\right) \right. \\ &\quad \left. \times\ v_{mi} \exp\left(-\frac{1}{2}(\boldsymbol{\beta}+\mathbf{d}_s-\boldsymbol{\mu}_{mi})^{\mathrm{T}}\mathbf{A}_{mi}(\boldsymbol{\beta}+\mathbf{d}_s-\boldsymbol{\mu}_{mi})\right) \right\} d^D\boldsymbol{\beta} \\ &= \sum_{m=1}^{M} v_{mi}v_{mj} f(\boldsymbol{\beta}, \boldsymbol{\mu}_{mj}, \mathbf{d}_s - \boldsymbol{\mu}_{mi}, \mathbf{A}_{mj}, \mathbf{A}_{mi}) \\ &= \sum_{m=1}^{M} \frac{(2\pi)^{\frac{D}{2}} v_{mi}v_{mj}}{\sqrt{|\mathbf{A}_{mj}+\mathbf{A}_{mi}|}} \exp\left(-\frac{1}{2}(\mathbf{d}_s - [\boldsymbol{\mu}_{mi}-\boldsymbol{\mu}_{mj}])^{\mathrm{T}}\boldsymbol{\Sigma}\,(\mathbf{d}_s - [\boldsymbol{\mu}_{mi}-\boldsymbol{\mu}_{mj}])\right) \end{aligned}$$

where $\boldsymbol{\Sigma} = \mathbf{A}_{mi}(\mathbf{A}_{mi}+\mathbf{A}_{mj})^{-1}\mathbf{A}_{mj}$.

Bibliography

[1] ABRAHAMSEN, P. A review of Gaussian random fields and correlation functions. Tech. Rep. 917, Norwegian Computing Center, Box 114, Blindern, N-0314 Oslo, Norway, 1997.

[2] BAHER, H. *Analog Digital Signal Processing*. John Wiley Sons, 1990.

[3] BARBER, D., AND WILLIAMS, C. K. I. Gaussian processes for Bayesian classification via hybrid monte carlo. In *Advances in Neural Information Processing Systems* (1997), M. C. Mozer, M. I. Jordan, and T. Petsche, Eds., vol. 9, The MIT Press.

[4] BISHOP, C. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[5] BOX, G., JENKINS, G. M., AND REINSEL, G. *Time Series Analysis: Forecasting Control*, 3 ed. Prentice Hall, 1994.

[6] BOYLE, P., AND FREAN, M. Dependent Gaussian processes. In *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. MIT Press, Cambridge, MA, 2005, pp. 217–224.

[7] BOYLE, P., AND FREAN, M. Multiple output Gaussian process regression. Tech. Rep. CS-TR-05-2, Victoria University of Wellington, 2005.

[8] BUCHE, D., SCHRAUDOLPH, N., AND KOUMOUTSAKOS, P. Accelerating evolutionary algorithms with gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics 35*, 2 (2005), 183–194.

[9] BUNTINE, W., AND WEIGEND, A. Bayesian backpropagation. *Complex Systems 5* (1991), 603–643.

[10] CALDER, C. Efficient posterior inference and prediction of space-time processes using dynamic process convolutions. *Joint Proceedings of the Sixth International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences and the Fifteenth Annual Conference of TIES, The International Environmetrics Society* (2004).

[11] CHAN, R. H., AND NG, M. K. Conjugate gradient methods for Toeplitz systems. *SIAM Review 38*, 3 (1996), 427–482.

[12] CRESSIE, N. *Statistics for Spatial Data*. Wiley, 1993.

[13] CSATÓ, L. *Gaussian Processes - Iterative Sparse Approximation*. PhD thesis, Aston University, 2002.

[14] CSATÓ, L., AND OPPER, M. Sparse representation for Gaussian process models. In *Advances in Neural Information Processing Systems, NIPS* (2001), vol. 13, pp. 444–450.

[15] CSATÓ, L., AND OPPER., M. Sparse on-line Gaussian processes. *Neural Computation 14* (2002), 641–668.

[16] FRANKLIN, G., POWELL, J., AND EMAMI-NAEINI, A. *Feedback Control of Dynamic Systems*. Addison-Wesley, 1998.

[17] GIBBS, M. *Bayesian Gaussian Processes for Classification and Regression*. PhD thesis, University of Cambridge, Cambridge, U.K., 1997.

[18] GIBBS, M., AND MACKAY, D. J. Efficient implementation of Gaussian processes. `http://www.inference.phy.cam.ac.uk/mackay/abstracts/gpros.html`, 1996.

[19] GIBBS, M. N., AND MACKAY, D. J. Variational Gaussian process classifiers. *IEEE Trans. on Neural Networks 11*, 6 (2000), 1458–1464.

[20] GOLDBERG, P. W. WILLIAMS, C. K. I., AND BISHOP, C. M. Regression with input-dependent noise: A Gaussian process treatment. In *Advances in Neural Information Processing Systems, NIPS* (1998), M. J. Jordan, M. I. Kearns and S. A. Solla, Eds., vol. 10.

[21] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix Computations*, 2 ed. The Johns Hopkins University Press, 1989.

[22] GRUAU, F., WHITLEY, D., AND PYEATT, L. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, USA, 28–31 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., MIT Press, pp. 81–89.

[23] HAYKIN, S. *Communication Systems, 3rd ed.* Wiley, 1994.

[24] HIGDON, D. Space and space-time modelling using process convolutions. In *Quantitative methods for current environmental issues* (2002), C. Anderson, V. Barnett, P. Chatwin, and A. El-Shaarawi, Eds., Springer Verlag, pp. 37–56.

[25] IFEACHOR, E. C., AND JERVIS, B. W. *Digital Signal Processing: A Practical Approach*. Addison-Wesley, 1993.

[26] JEFFREYS, H. *Theory of Probability*. Oxford: Clarendon Press, 1961.

[27] JONES, D. R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization 21* (2001), 345–383.

[28] LAMPINEN, J., AND VEHTARI, A. Bayesian neural netwroks - review and case studies. *Neural Networks 14*, 3 (2001), 7–24.

[29] LEE, H., HIGDON, D., CALDER, C., AND HOLLOMAN, C. Efficient models for correlated data via convolutions of intrinsic processes. *Statistical Modelling 5*, 1 (2005), 53–74.

[30] LJUNG, L. From data to model: A guided tour of system identification. Tech. Rep. Lith-ISY-R-1652, Department of Electrical Engineering, Linköping University, 1994.

[31] LJUNG, L., AND SÖDERSTRÖM, T. L. *Theory and Practice of Recursive Identification*. MIT Press, 1983.

[32] LÜTKEPOHL, H. *New Introduction to Multiple Time Series Analysis*. Springer, 2005.

[33] MACKAY, D. J. Gaussian processes: A replacement for supervised neural networks? *In NIPS97 Tutorial*, 1997.

[34] MACKAY, D. J. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[35] MACKAY, D. J. C. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, 1992.

[36] MACKAY, D. J. C. A practical Bayesian framework for backprop networks. *Neural Computation* (1992).

[37] MACKAY, D. J. C. Bayesian methods for backpropagation networks. In *Models of Neural Networks III*, E. Domany, J. L. van Hemmen, and K. Schulten, Eds. Springer-Verlag, New York, 1994, ch. 6, pp. 211–254.

[38] MACKAY, D. J. C. Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems 6* (1995), 469–505.

[39] MACKAY, D. J. C. Bayesian non-linear modelling for the 1993 energy prediction competition. In *Maximum Entropy and Bayesian Methods, Santa Barbara 1993* (Dordrecht, 1996), G. Heidbreder, Ed., Kluwer, pp. 221–234.

[40] MACKAY, D. J. C. Introduction to Gaussian processes. In *Neural Networks and Machine Learning*, C. M. Bishop, Ed., NATO ASI Series. Kluwer, 1998, pp. 133–166.

[41] MACKAY, D. J. C. Introduction to Monte Carlo methods. In *Learning in Graphical Models*, M. I. Jordan, Ed., NATO Science Series. Kluwer Academic Press, 1998, pp. 175–204.

[42] MATHERON, G. Principles of geostatistics. *Economic Geology 58* (1963), 1246–1266.

[43] MITCHELL, T. *Machine Learning*. McGraw Hill, 1997.

[44] MYERS, R. H., AND MONTGOMERY, D. C. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, 2 ed. Wiley-Interscience, 2002.

[45] NEAL, R. Probabilistic inference using markov chain monte carlo methods. Tech. Rep. CRG-TR-93-1, Dept. of Computer Science, Univ. of Toronto, 1993.

[46] NEAL, R. *Bayesian Learning for Neural Networks, Lecture Notes in Statistics, No 118*. Springer-Verlag, 1996.

[47] NEAL, R. Monte carlo implementation of Gaussian process models for Bayesian regression and classification. Tech. Rep. CRG-TR-97-2, Dept. of Computer Science, Univ. of Toronto, 1997.

[48] NEAL, R. M. Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Tech. Rep. CRG-TR-92-1, Dept. of Computer Science, Univ. of Toronto, 1992.

[49] NEAL, R. M. Annealed importance sampling. Tech. Rep. 9805, Department of Statistics, University of Toronto, 1998.

[50] O'HAGAN, A. Curve fitting and optimal design for prediction (with discussion). *J. Roy. Statist. Soc. Ser. B 40* (1978), 1–42.

[51] OSGOOD, B. *Lecture Notes for EE 261, The Fourier Transform and its Applications*. Electrical Engineering Department, Stanford University, 2005, ch. 8.

[52] PACIOREK, C. *Nonstationary Gaussian processes for regression and spatial modelling*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A., 2003.

[53] PACIOREK, C., AND SCHERVISH, M. Nonstationary covariance functions for Gaussian process regression. In *Advances in Neural Information Processing Systems, NIPS 16* (2004), pp. 273–280.

[54] PRESS, W., FLANNERY, B., TEUKOLSKY, S., AND VETTERLING, W. *Numerical recipes in C: The art of scientific computing.* Cambridge University Press., 1989.

[55] QUIÑONERO-CANDELA, J., AND RASMUSSEN, C. E. Analysis of some methods for reduced rank Gaussian process regression. In *Switching and Learning in Feedback Systems* (Heidelberg, Germany, January 2005), R. Murray-Smith and R. Shorten, Eds., vol. 3355 of *Lecture Notes in Computer Science*, Springer, pp. 98–127.

[56] QUIÑONERO-CANDELA, J., AND RASMUSSEN, C. E. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research 6*, 12 (2005), 1935–1959.

[57] RASMUSSEN, C. E. *Evaluation of Gaussian Processes and other methods for Non-Linear Regression*. PhD thesis, Graduate Department of Computer Science, University of Toronto, 1996.

[58] RASMUSSEN, C. E. Gaussian processes to speed up hybrid monte carlo for expensive bayesian integrals. In *Bayesian Statistics* (2003), J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. M. Smith, and M. West, Eds., vol. 7, Oxford University Press, pp. 651–659.

[59] RASMUSSEN, C. E., AND GHAHRAMANI, Z. Infinite mixtures of Gaussian process experts. In *Advances in Neural Information Processing Systems, NIPS* (2002), T. G. Diettrich, S. Becker, and Z. Ghahramani, Eds., vol. 14, The MIT Press.

[60] RASMUSSEN, C. E., AND KUSS, M. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems, NIPS* (2002), S. Thrun, L. Saul, and B. Schlkopf, Eds., vol. 16, The MIT Press.

[61] RASMUSSEN, C. E., AND WILLIAMS, C. K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[62] RUMELHART, D., HINTON, G., AND WILLIAMS, R. Learning representations by back-propagating errors. *Nature 323* (1986), 533–536.

[63] SEEGER, M. *Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations*. PhD thesis, University of Edinburgh, 2003.

[64] SEEGER, M. Gaussian processes for machine learning. Tech. rep., Department of EECS, University of California at Berkeley, 2004.

[65] SEEGER, M. Gaussian processes for machine learning. *International Journal of Neural Systems 14*, 2 (2004), 1–38.

[66] SEEGER, M., AND WILLIAMS, C. Fast forward selection to speed up sparse Gaussian process regression, 2003. In Workshop on AI and Statistics 9.

[67] SHANNON, C. Amathematical theory of communication. *Bell System Tech. J. 27* (1948).

[68] SHEWCHUK, J. R. An introduction to the conjugate gradient method without the agonizing pain. Tech. Rep. CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

[69] SILVERMAN, B. W. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *J.R.Statist.Soc.B 47* (1985), 1–52.

[70] SMITH, J. O. Introduction to digital filters, september 2005 draft. `http://ccrma.stanford.edu/~jos/filters/Time_Domain_Filter_Estimation.html`, May 2005.

[71] SMOLA, A. J., AND BARTLETT, P. L. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems, NIPS* (2001), vol. 13, pp. 619–625.

[72] SNELSON, E., C. E. R., AND GHAHRAMANI, Z. Warped Gaussian processes. In *Advances in Neural Information Processing Systems, NIPS* (2004), L. S. Thrun, S. and B. Schlkopf, Eds., vol. 16, pp. 337–344.

[73] SOLAK, E., MURRAY-SMITH, R., LEITHEAD, W. E., LEITH, D., AND RASMUSSEN, C. E. Derivative observations in Gaussian process models of dynamic systems. In *Advances in Neural Information Processing Systems, NIPS* (2003), vol. 15, The MIT Press, pp. 1033–1040.

[74] STANLEY, K. O. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2004.

[75] STANLEY, K. O., AND MIIKKULAINEN, R. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)* (2002), Morgan Kaufmann.

[76] STANLEY, K. O., AND MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*, 2 (2002), 99–127.

[77] STORKEY, A. Truncated covariance matrices and toeplitz methods in Gaussian processes. In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN'99), London, UK* (1999), pp. 55–60.

[78] STORKEY, A. J. *Efficient Covariance Matrix Methods for Bayesian Gaussian Processes and Hopfield Neural Networks*. PhD thesis, University of London, 1999.

[79] TRENCH, W. F. An algorithm for the inversion of finite toeplitz matrices. *SIAM J. Appl. Math. 12* (1964).

[80] TRESP, V. A Bayesian committee machine. *Neural Computation 12*, 11 (2000), 2719–2741.

[81] TRESP, V. Mixtures of Gaussian processes. In *Advances in Neural Information Processing Systems, NIPS* (2001), T. K. Leen, T. G. Dietterich, and T. V., Eds., vol. 13, The MIT press.

[82] VIVARELLI, F., AND WILLIAMS, C. K. Discovering hidden features with Gaussian processes regression. *Advances in Neural Information Processing Systems 11* (1999).

[83] WANG, J., FLEET, D., AND HERTZMANN, A. Gaussian process dynamical models. In *Advances in Neural Information Processing Systems, NIPS* (2006), Y. Weiss, B. Schlkopf, and J. Platt, Eds., vol. 18, The MIT Press, pp. 1443–1450.

[84] WATSON, G. A. An algorithm for the inversion of block matrices of toeplitz form. *J. ACM 20*, 3 (1973), 409–415.

[85] WEISSTEIN, E. W. Moore-Penrose matrix inverse. `http://mathworld.wolfram.com/Moore-PenroseMatrixInverse.html`.

[86] WILLIAMS, C. K., AND RASMUSSEN, C. E. Gaussian processes for regression. In *Advances in Neural Information Processing Systems* (1996), D. Touretzsky, M. Mozer, and M. Hasselmo, Eds., vol. 8.

[87] WILLIAMS, C. K. I., RASMUSSEN, C. E., SCHWAIGHOFER, A., AND TRESP, V. Observations on the Nyström method for Gaussian process prediction. Tech. rep., University of Edinburgh, 2002.

[88] WILLIAMS, C. K. I., AND SEEGER, M. Using the Nyström method to speedup kernel machines. *Advances in Neural Information Processing Systems 13* (2001).

[89] WOLPERT, D. H., AND MACREADY, W. G. No free lunch theorems for search. Tech. Rep. SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, NM, 1995.

[90] WOLPERT, D. H., AND MACREADY, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 1 (April 1997), 67–82.

[91] ZEIDLER, E., Ed. *Oxford Users'Guide to Mathematics*. Oxford University Press, 2004.

[92] ZHANG, F. *The Schur Complement and Its Applications*. Springer-Verlag, 2005.