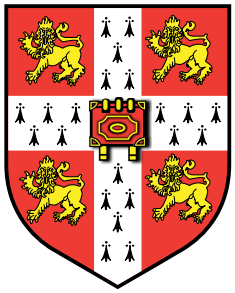
Scalable Inference for Structured Gaussian Process Models



Yunus Saatçi

St. Edmund's College

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

December 15, 2011

Preface

This thesis contributes to the field of Bayesian machine learning. Familiarity with most of the material in Bishop [2007], MacKay [2003] and Hastie et al. [2009] would thus be convenient for the reader. Sections which may be skipped by the expert reader without disrupting the flow of the text have been clearly marked with a "fast-forward" ([▷▷]) symbol. It will be made clear in the text which parts represent core contributions made by the author.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text. This disseration does not exceed sixty-five thousand words in length. No parts of the dissertation have been submitted for any other qualification.

Abstract

The generic inference and learning algorithm for Gaussian Process (GP) regression has $\mathcal{O}(N^3)$ runtime and $\mathcal{O}(N^2)$ memory complexity, where N is the number of observations in the dataset. Given the computational resources available to a present-day workstation, this implies that GP regression simply *cannot be run* on large datasets. The need to use non-Gaussian likelihood functions for tasks such as classification adds even more to the computational burden involved.

The majority of algorithms designed to improve the scaling of GPs are founded on the idea of approximating the true covariance matrix, which is usually of rank N, with a matrix of rank P, where $P \ll N$. Typically, the true training set is replaced with a smaller, representative (pseudo-) training set such that a specific measure of information loss is minimized. These algorithms typically attain $\mathcal{O}(P^2N)$ runtime and $\mathcal{O}(PN)$ space complexity. They are also general in the sense that they are designed to work with *any* covariance function. In essence, they trade off accuracy with computational complexity. The central contribution of this thesis is to improve scaling instead by exploiting any structure that is present in the covariance matrices generated by *particular* covariance functions. Instead of settling for a kernel-independent accuracy/complexity trade off, as is done in much the literature, we often obtain accuracies close to, or exactly equal to the full GP model at a fraction of the computational cost.

We define a *structured* GP as any GP model that is endowed with a kernel which produces structured covariance matrices. A trivial example of a structured GP is one with the linear regression kernel. In this case, given inputs living in $\mathbb{R}^D$, the covariance matrices generated have rank D – this results in significant computational gains in the usual case where $D \ll N$. Another case arises when a stationary kernel is evaluated on equispaced, scalar inputs. This results in *Toeplitz* covariance matrices and all necessary computations can be carried out exactly in $\mathcal{O}(N \log N)$.

This thesis studies four more types of structured GP. First, we comprehensively review the case of kernels corresponding to *Gauss-Markov* processes evaluated on scalar inputs. Using state-space models we show how

(generalised) regression (including hyperparameter learning) can be performed in $\mathcal{O}(N \log N)$ runtime and $\mathcal{O}(N)$ space. Secondly, we study the case where we introduce block structure into the covariance matrix of a GP time-series model by assuming a particular form of nonstationarity a priori. Third, we extend the efficiency of scalar Gauss-Markov processes to higher-dimensional input spaces by assuming *additivity*. We illustrate the connections between the classical backfitting algorithm and approximate Bayesian inference techniques including Gibbs sampling and variational Bayes. We also show that it is possible to relax the rather strong assumption of additivity without sacrificing $\mathcal{O}(N \log N)$ complexity, by means of a projection-pursuit style GP regression model. Finally, we study the properties of a GP model with a tensor product kernel evaluated on a multivariate *grid* of inputs locations. We show that for an *arbitrary* (regular or irregular) grid the resulting covariance matrices are *Kronecker* and full GP regression can be implemented in $\mathcal{O}(N)$ time and memory usage.

We illustrate the power of these methods on several real-world regression datasets which satisfy the assumptions inherent in the structured GP employed. In many cases we obtain performance comparable to the generic GP algorithm. We also analyse the performance degradation when these assumptions are not met, and in several cases show that it is comparable to that observed for sparse GP methods. We provide similar results for regression tasks with non-Gaussian likelihoods, an extension rarely addressed by sparse GP techniques.

I would like to dedicate this thesis to my family and my close friends. I would not be here writing this very sentence if it were not for them...

Acknowledgements

I would like to thank my supervisor Carl Edward Rasmussen for being a friend, co-founder and supervisor to me all at once. He gave me considerable freedom in pursuing my own research direction, yet when it came to the *detail* he would always provide extremely valuable advice. One of the most important notions I will take away from his supervision will be that "the devil is in the detail". I hope this thesis actually demonstrates that I have incorporated this notion into my research.

I am extremely grateful to have Zoubin Ghahramani as the head of my lab, and my advisor. His breadth of knowledge in machine learning sets a very high bar for aspiring students. His ability to explain seemingly complex ideas in a simple, "fluff-free" way, and his never-ending desire to unify seemingly-disparate ideas under a common framework is simply invaluable to confused students such as myself. This, combined with a friendly, humble and enthusiastic personality, makes Zoubin an absolute pleasure to work with and to learn from.

I am also highly grateful to have worked in the presence of the PhD students and postdocs of the Computational & Biological Learning Lab (and neighbouring labs!). Nothing contributes more to research than people bouncing ideas off each other! In particular, I would like to thank John P. Cunningham, Ryan Turner, Marc Deisenroth, Shakir Mohamed, Ferenc Huszar, Alex Davies, Andrew Wilson, Jurgen van Gael, David Knowles and David Duvenaud for influencing my train of thought which has culminated in this thesis.

I would also like to thank the Engineering and Physical Sciences Research Council (EPSRC) and Ceptron HK Limited for funding my research. I also thank my beloved St. Edmund's College for funding my travels and the final few months of my PhD, and providing some amazing accommodation and an amazing bar filled with amazing people!

> It is remarkable that a science which began with the consideration of games of chance should have become the most important object of human knowledge. [...] The most important questions of life are indeed, for the most part, really only problems of probability. [...] Probability theory is nothing but common sense reduced to calculation.

— Pierre-Simon, marquis de Laplace, *Théorie Analytique des Probabilités, 1812.*

Contents

List of Figures

List of Algorithms

List of Tables

Notation

We write a scalar as x, a vector as $\mathbf{x}$, a matrix as $\mathbf{X}$. The ith element of a vector is in the typeface of a scalar x_i. The ith row and jth column of $\mathbf{X}$ is $X(i,j)$ or $X_{i,j}$. The ith row of $\mathbf{X}$ is $\mathbf{X}_{i,:}$ or $\mathbf{x}_i$. The ith column of $\mathbf{X}$ is $\mathbf{X}_{:,i}$ or $\mathbf{x}_i$. We represent an inclusive range between a and b as $a : b$. If an Equation refers to another Equation in square brackets ("[·]"), this means that the referred Equation has been used to derive the referring Equation. By standard convention, even though Gaussian Process hyperparameters form a vector, we represent them with the typeface of a scalar, θ.

Symbols used

$\mathbb{R}$	The real numbers.
$\mathbb{R}^+$	Positive real numbers.
$\mathbb{C}$	The complex numbers.
$\mathbb{Q}$	The rational numbers.
$\mathbb{Z}$	The integers.
$\mathbb{Z}^+$	Positive integers.
$x*$	The complex conjugate of x.
$\mathbf{X} \otimes \mathbf{Y}$	The Kronecker product of $\mathbf{X}$ and $\mathbf{Y}$.
$\mathbf{1}$	A vector of ones.
$\mathbf{0}$	A vector of zeros or a matrix of zero, depending on context.
$\mathbf{I}_D$	The identity matrix of size D.
$\text{vec}(\mathbf{X})$	The vectorization of a matrix $\mathbf{X}$.
$\text{KL}(p\|\|q)$	The Kullback-Leibler (KL) divergence between distributions p and q.
$H(p)$	The entropy of distribution p.
$\mathbb{E}(X)$	Expectation of a random variable X.
$\mathbb{V}(X)$	Variance of a random variable X.
$\text{Median}(X)$	Median of a random variable X.
$\mathbb{E}_{p(\cdot)}(X)$	Expectation of a random variable X with respect to p.
$\mathbb{V}_{p(\cdot)}(X)$	Variance of a random variable X with respect to p.
$\text{Cov}(\mathbf{X})$	Covariance of a vector random variable $\mathbf{X}$.
$\xrightarrow{p}$	Convergence in probability.
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	A Gaussian distribution with specified mean $\boldsymbol{\mu}$ and (co-)variance $\boldsymbol{\Sigma}$. Random variable symbol is omitted.
$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	A Gaussian distribution with specified mean $\boldsymbol{\mu}$ and (co-)variance $\boldsymbol{\Sigma}$. Random variable symbol *not* omitted.
Student-$\text{t}_\nu(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	A multivariate Student's t distribution with mean $\boldsymbol{\mu}$, covariance $\boldsymbol{\Sigma}$, and ν degrees of freedom.

$\Gamma(\alpha, \beta)$	A gamma distribution with shape α and inverse scale β.
Poisson(λ)	A Poisson distribution with mean λ.
$\mathcal{GP}(\mu, k)$	A Gaussian process (GP) with mean function $\mu(\cdot)$ and kernel $k(\cdot, \cdot)$.
$x \equiv y$	x defined as y.
$\mathcal{O}(\cdot)$	The big-O asymptotic complexity of an algorithm.
$\leftarrow$	An assignment operation in an algorithm.
ne$(\cdot)$	Returns neighbours of a variable or factor in a factor graph.
w.r.t.	Shortening of the phrase "with respect to".
IID	Shortening of the phrase "independent, identically distributed".
p.d.f.	Shortening of the phrase "probability density function".
c.d.f.	Shortening of the phrase "cumulative distribution function".

Chapter 1

Introduction & Motivation

1.1 Gaussian Process Regression [▷▷]

Regression is the task of returning predictions of a continuous output variable at *any* input location, given a training set of input-output pairs. The inputs can be any type of object which is hoped to provide some predictability of the response, although more often that not, they are a set of real-valued features living in $\mathbb{R}^D$, and will be assumed to be so in this thesis.

As a direct consequence of the definition of regression, it is clear that inference must revolve around a *function* mapping inputs to outputs, because only by inferring a function can we predict the response at any input location. In the case of Bayesian inference, this implies that one needs to define a prior distribution over *functions*. Learning will occur as a result of updating the prior in light of the training set $\{\mathbf{X}, \mathbf{y}\}$, where $\mathbf{X} \equiv \{\mathbf{x}_n\}_{n=1}^N, \mathbf{x}_n \in \mathbb{R}^{\mathbb{D}}$ are the training inputs and $\mathbf{y} \equiv \{y_n\}_{n=1}^N, y_i \in \mathbb{R}$ are the training targets, to obtain a posterior distribution over functions. The question then is, how can one place a distribution over an infinite dimensional object such as a function?

Bayesian Parametric Regression

Conceptually, the simplest way is to parameterise the function with a finite set of parameters $\mathbf{w}$ and place a prior over them. This implicitly results in a prior over functions. In other words, we assume a parametric form for the unknown function,

$f(\mathbf{x}; \mathbf{w})$. Clearly, placing a prior $p(\mathbf{w}|\theta)$ induces a distribution over functions, where the type of functions supported depends on the relationship of $\mathbf{w}$ to the function value. For example, if one expects a linear input-output relationship to exist in the data then a good choice would be $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$. The functions supported by the prior, and thus the posterior, will then only include functions linear in the input. A set of functions drawn from such a "linear function" prior is illustrated in Figure 1.1.

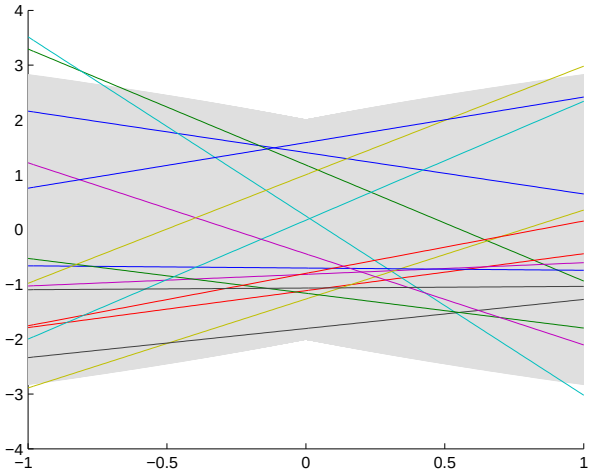


Figure 1.1: 50 functions over $\{-1, 1\}$ drawn from the prior induced by $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$. Notice how the marginal standard deviation increases with distance from origin – this is a direct result of the prior supporting linear functions only. The greyscale highlights a distance of 2 standard deviations from 0.

Representing functions with a finite number of parameters allows function learning to take place via the learning of $\mathbf{w}$:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \theta) = \frac{p(\mathbf{y}|\mathbf{w}, \mathbf{X}, \theta)p(\mathbf{w}|\theta)}{Z(\theta)}, \tag{1.1}$$

where

$$Z(\theta) = p(\mathbf{y}|\mathbf{X}, \theta) = \int p(\mathbf{y}|\mathbf{w}, \mathbf{X}, \theta)p(\mathbf{w}|\theta)d\mathbf{w}, \tag{1.2}$$

is the normalizing constant, also known as the *marginal likelihood* of the model specified by *hyperparameters* θ, because it computes precisely the likelihood of observing the given dataset given the modelling assumptions encapsulated in θ. As a result, it also offers the opportunity to perform *model adaptation* via optimization with

respect to θ. Of course, if the problem at hand requires averaging over models with different hyperparameters to improve predictions (e.g. if the dataset is small), it is possible to add another layer to the hierarchy of Bayesian inference.

The posterior in Equation 1.1 can be used directly to compute both the expected value of the underlying function $f(\mathbf{x}_\star; \mathbf{w})$ and its variance at input location $\mathbf{x}_\star$, through evaluating

$$\mu_\star = \mathbb{E}_{p(\mathbf{w}|\mathbf{y},\mathbf{X},\theta)}\left(f(\mathbf{x}_\star; \mathbf{w})\right), \tag{1.3}$$

$$\sigma_\star^2 = \mathbb{E}_{p(\mathbf{w}|\mathbf{y},\mathbf{X},\theta)}\left(f(\mathbf{x}_\star; \mathbf{w})^2\right) - \mu_\star^2. \tag{1.4}$$

Thus, we can predict the function value and, perhaps equally importantly, report how certain we are about our prediction at *any* input location, as is required for the task of regression.

An example of Equations 1.1 through 1.4 in action is given by Bayesian linear regression, for which:

- $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$.
- $p(\mathbf{w}|\theta) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \mathbf{\Sigma})$.
- $p(\mathbf{y}|\mathbf{w}, \mathbf{X}, \theta) = \prod_{i=1}^{N} \mathcal{N}(y_i; \mathbf{w}^\top \mathbf{x}, \sigma_n^2)$.

The prior covariance $\mathbf{\Sigma}$ is usually a diagonal matrix with D independent parameters. The *likelihood* term $p(\mathbf{y}|\mathbf{w}, \mathbf{X})$ encodes the assumption that the observations are the true function values corrupted with zero mean IID Gaussian noise with variance σ_n^2. The hyperparameters include the parameters in Σ and the noise variance. Because both the prior and the likelihood are Gaussian it follows that the posterior $p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \theta)$ is also Gaussian and $Z(\theta)$ can be calculated analytically:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \theta) = \mathcal{N}(\boldsymbol{\mu}_N, \mathbf{\Sigma}_N), \tag{1.5}$$

$$\log Z(\theta) = -\frac{1}{2}\left(\mathbf{y}^\top \mathbf{\Sigma}_E^{-1} \mathbf{y} + \log(\det(\mathbf{\Sigma}_E)) + N \log(2\pi)\right), \tag{1.6}$$

where

$$\boldsymbol{\mu}_N = \frac{1}{\sigma_n^2}\boldsymbol{\Sigma}_N \mathbf{X}^\top \mathbf{y}, \tag{1.7}$$

$$\boldsymbol{\Sigma}_N = \left(\boldsymbol{\Sigma}^{-1} + \frac{1}{\sigma_n^2}\mathbf{X}^\top \mathbf{X}\right)^{-1}, \tag{1.8}$$

$$\boldsymbol{\Sigma}_E = \mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^\top + \sigma_n^2 \mathbf{I}_N. \tag{1.9}$$

The predictive distribution at $\mathbf{x}_\star$ also becomes a Gaussian and is fully specified by $\mu_\star$ and $\sigma_\star^2$

$$\mu_\star = \boldsymbol{\mu}_N^\top \mathbf{x}_\star, \tag{1.10}$$

$$\sigma_\star^2 = \mathbf{x}_\star{}^\top \boldsymbol{\Sigma}_N \mathbf{x}_\star. \tag{1.11}$$

For a list of standard Gaussian identities, see Appendix A. The posterior distribution over functions implied by Equation 1.5 given a synthetic dataset is shown in Figure 1.2. Note how the predictive mean in Equation 1.10 is linear in the training targets, and how the predictive variance in Equation 1.11 does not depend on the targets at all – these are hallmark properties of analytically tractable regression models. In addition, it can be seen from Equation 1.6 that $Z(\theta)$ is a complicated non-linear function of the hyperparameters, indicating that attempts to integrate them out will have to resort to approximate inference techniques such as MCMC (for a recent example, see Murray and Adams [2010]). Hyperparameter optimization is an easier and well-justified alternative as long as there is enough data. As shown in O'Hagan and Forster [1994] and Jaynes [2003], the hyperparameter posterior will tend to a Gaussian distribution (i.e., a *unimodal* one) in the infinite data limit, given certain conditions which are usually satisfied by the standard GP setup. Furthermore, asymptotic Gaussianity applies because:

$$p(\theta|\mathbf{y}) \propto \prod_{i=1}^{N} p(y_i|\theta, y_1, \ldots y_{i-1}) p(\theta), \tag{1.12}$$

and it is usually the case that

$$\frac{1}{N}\sum_{i=1}^{N}\frac{\partial^2 \log p(y_i|\theta, y_1, \ldots y_{i-1})}{\partial\theta^2} \xrightarrow{p} k, \text{ as } N \to \infty, \; k \neq 0. \tag{1.13}$$

The condition in Equation 1.13 can be interpreted as requiring that the observations do not get less and less "informative" about θ. Suppressing dependence on input locations and integrating out the latent function, Equation 1.12 can be viewed as describing the GP hyperparameter posterior, given that the likelihood terms correspond to the GP marginal likelihood evaluated using the chain rule of probability. For GP models, it is usually the case that every observation is informative about the hyperparameters, except in cases where, for example, we have coincident inputs.

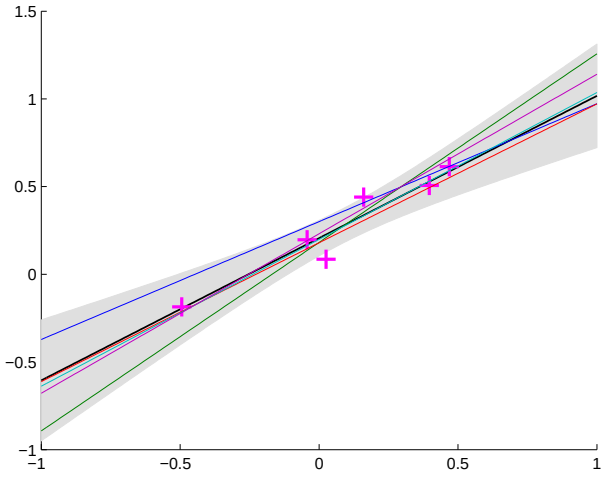


Figure 1.2: The bold line shows the posterior mean, and the greyscale errorbars show 2 times the marginal posterior standard deviation, given the data (shown as magenta points). The coloured lines represent 5 samples from the posterior.

The flexibility of the linear regression model can be increased by mapping the inputs $\mathbf{x}$ into some possibly higher-dimensional feature space through some feature mapping ϕ, although linearity in the *parameters* $\mathbf{w}$ must be retained to ensure analytic tractability. Furthermore, the expressions for the posterior, marginal likelihood and predictive distributions remain mainly unchanged – all that is required is to replace every occurrence of an input location $\mathbf{x}_i$ with its feature mapping $\phi(\mathbf{x}_i) \equiv \boldsymbol{\phi}_i$. If the underlying function $f(\mathbf{x}; \mathbf{w})$ is assumed to be nonlinear in $\mathbf{w}$, then Equations 1.1 through 1.4 become analytically intractable and one must resort to approximate

inference methods.

Bayesian Nonparametric Regression

While conceptually simple, the parametric way of placing a distribution over functions suffers from several shortcomings. Firstly, the parameters in $\mathbf{w}$ are simply coefficients, so it is difficult in practice to specify a prior over them which correctly captures our intuitive beliefs about *functional* characteristics such as smoothness and amplitude. For example, let's imagine we are employing the linear model outlined above in some feature space. It is hard to map a prior over the coefficient of say a polynomial feature $\phi(\mathbf{x}) = x_i^a x_j^b x_k^c$ for $a, b, c \in \mathbb{Z}^+$ to properties such as smoothness and differentiability. As a result, both the prior and the posterior over $\mathbf{w}$ can get difficult to interpret. Secondly, when attempting to model complex input-output relationships the number of parameters can get large. Of course, we avoid the risk of overfitting by integrating over the parameter vector, however, a large parameter vector can result an uncomfortably large number of hyperparameters. The hyperparameter vector may grow even further as a result of using hand-crafted features with their own internal parameters. Hence, overfitting can return to haunt us during hyperparameter optimization.

It is natural to therefore ask whether there is an *direct* way of placing a prior over the underlying function $\mathbf{f}$. The answer lies in the use of *stochastic processes*, which are distributions over functions, *by definition*[1]. For regression the simplest stochastic process which serves the purpose is the *Gaussian process* (GP). The method for describing a distribution over functions (which are infinite dimensional) is accomplished through specifying every finite dimensional marginal density implied by that distribution. Conceptually the distribution over the entire function exists "in the limit". This leads us to the following definition of the GP:

Definition 1. *The probability distribution of* $\mathbf{f}$ *is a Gaussian process if any finite selection of input locations* $\mathbf{x}_1, \ldots, \mathbf{x}_N$ *gives rise to a multivariate Gaussian density over the associated targets, i.e.,*

$$p(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)) = \mathcal{N}(\mathbf{m}_N, \mathbf{K}_N), \tag{1.14}$$

[1]Stochastic processes were originally conceived to be distributions over functions of time, however, it is possible to extend the idea to multidimensional input spaces.

where $\mathbf{m}_N$ is the mean vector of length N and $\mathbf{K}_N$ is the N-by-N covariance matrix. The mean and covariance of every finite marginal is computed using the *mean function* and the *covariance function* of the GP respectively. As is the case for the Gaussian distribution, the mean and covariance functions are sufficient to fully specify the GP. The mean function $\mu(\mathbf{x})$ gives the average value of the function at input $\mathbf{x}$, and is often equal to zero everywhere because it usually is the case that, *a priori*, function values are equally likely to be positive or negative. The covariance function $k(\mathbf{x}, \mathbf{x}') \equiv \mathrm{Cov}(f(\mathbf{x}), f(\mathbf{x}'))$ specifies the covariance between the function values at two input locations. For the prior over functions to be proper, the covariance function has to be a *positive definite* function satisfying the property that $\iint h(\mathbf{x})k(\mathbf{x}, \mathbf{x}')h(\mathbf{x}')d\mathbf{x}d\mathbf{x}' > 0$ for any h except $h(\mathbf{x}) = 0$. For any finite marginal of the GP as defined above we have that, for $i, j = 1, \ldots, N$:

$$m_N(i) = \mu(\mathbf{x}_i), \tag{1.15}$$

$$K_N(i, j) = k(\mathbf{x}_i, \mathbf{x}_j). \tag{1.16}$$

This is a consequence of the *marginalization property* of the Gaussian distribution, namely that if:

$$p\left(\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_\mathbf{a}\\ \boldsymbol{\mu}_\mathbf{b}\end{bmatrix}, \begin{bmatrix}\mathbf{K}_{\mathbf{a},\mathbf{a}} & \mathbf{K}_{\mathbf{a},\mathbf{b}}\\ \mathbf{K}_{\mathbf{b},\mathbf{a}} & \mathbf{K}_{\mathbf{b},\mathbf{b}}\end{bmatrix}\right), \tag{1.17}$$

then the means and covariances of marginals are simply the relevant subvectors and submatrices of the joint mean and covariance respectively, i.e., $p(\mathbf{a}) = \mathcal{N}(\mu_\mathbf{a}, \mathbf{K}_{\mathbf{a},\mathbf{a}})$ and $p(\mathbf{b}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{b}, \mathbf{K}_{\mathbf{b},\mathbf{b}})$. Indeed, this is exactly what is happening in Equations 1.15 and 1.16 – we are simply reading out the relevant subvector and submatrix of the infinitely long mean function and the infinitely large covariance function. Of course, we will never need access to such infinitely large objects in practice because we only need to query the GP at training and test inputs, which form a finite set. For a rigorous introduction to the theory of GPs, see Doob [1953].

The definition of a GP allows function learning to take place directly through distributions over function values and not through surrogate parameter vectors, namely

$$p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta) = \frac{p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta)p(\mathbf{f}|\mathbf{X}, \theta)}{Z(\theta)}, \tag{1.18}$$

and

$$Z(\theta) = p(\mathbf{y}|\mathbf{X}, \theta) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) p(\mathbf{f}|\mathbf{X}, \theta) d\mathbf{f}. \tag{1.19}$$

Conditioning the GP prior over $\mathbf{f}$ on the input locations $\mathbf{X}$ restricts attention to the finite marginal of the GP evaluated at $\mathbf{X}$. Therefore, $p(\mathbf{f}|\mathbf{X}, \theta) = \mathcal{N}(\mathbf{m}_N, \mathbf{K}_N)$, where θ now includes any parameters that control the mean and covariance function. In the following the mean function will always be assumed to equal the zero function, so we can further write $p(\mathbf{f}|\mathbf{X}, \theta) = \mathcal{N}(\mathbf{0}, \mathbf{K}_N)$. Note also that the integral used to compute the marginal likelihood in Equation 1.19 reduces to an N-dimensional integral. We assume the same noise model for the targets as we did for the parametric linear regression model – they are the true function values corrupted by IID Gaussian noise. Thus, the likelihood is simply $p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) = \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I}_N)$. As we once again have a Gaussian prior with a Gaussian likelihood we can evaluate the expression in Equations 1.18 and 1.19 analytically:

$$\begin{aligned} p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta) &= \mathcal{N}(\mathbf{m}_{N|\mathbf{y}}, \mathbf{K}_{N|\mathbf{y}}), & (1.20) \\ \log Z(\theta) &= -\frac{1}{2}\left(\mathbf{y}^\top \mathbf{K}_E^{-1} \mathbf{y} + \log(\det(\mathbf{K}_E)) + N \log(2\pi)\right). & (1.21) \end{aligned}$$

The expressions for the posterior mean and covariance and that for the marginal likelihood are quite similar to their parametric counterparts:

$$\begin{aligned} \mathbf{m}_{N|\mathbf{y}} &= \frac{1}{\sigma_n^2} \mathbf{K}_{N|\mathbf{y}} \mathbf{y}, & (1.22) \\ \mathbf{K}_{N|\mathbf{y}} &= \left(\mathbf{K}_N^{-1} + \frac{1}{\sigma_n^2} \mathbf{I}_N\right)^{-1}, & (1.23) \\ \mathbf{K}_E &= \mathbf{K}_N + \sigma_n^2 \mathbf{I}_N. & (1.24) \end{aligned}$$

Note that the posterior over $\mathbf{f}$ is also an N-dimensional Gaussian as it is also conditioned on the training inputs $\mathbf{X}$.

Given the posterior over the underlying function values, we can make predictions at M test inputs $\mathbf{X}_\star \equiv [\mathbf{x}_\star^{(1)}, \ldots, \mathbf{x}_\star^{(M)}]$ jointly. So far the equations derived have all assumed $M = 1$, however, with GPs we can predict jointly almost as easily as predicting individual test points so our predictive equations will be slightly more general than usual. Let $\mathbf{f}_\star$ be the random vector representing the function values at

$\mathbf{X}_\star$. Then,

$$p(\mathbf{f}_\star|\mathbf{X}_\star, \mathbf{X}, \mathbf{y}, \theta) = \int p(\mathbf{f}_\star|\mathbf{X}_\star, \mathbf{X}, \mathbf{f}, \theta) p(\mathbf{f}|\mathbf{X}, \mathbf{y}, \theta) d\mathbf{f}, \tag{1.25}$$

where the term $p(\mathbf{f}_\star|\mathbf{X}_\star, \mathbf{X}, \mathbf{f}, \theta)$ can be derived directly from the definition of a GP. Furthermore, since

$$p\left(\begin{bmatrix}\mathbf{f} \\ \mathbf{f}_\star\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{0} \\ \mathbf{0}\end{bmatrix}, \begin{bmatrix}\mathbf{K}_N & \mathbf{K}_{NM} \\ \mathbf{K}_{MN} & \mathbf{K}_M\end{bmatrix}\right), \tag{1.26}$$

where $\mathbf{K}_{MN}$ is the cross-covariance of the training and test inputs and $\mathbf{K}_M$ the covariance of the latter, standard Gaussian conditioning (see Appendix A) shows that

$$p(\mathbf{f}_\star|\mathbf{X}_\star, \mathbf{X}, \mathbf{f}, \theta) = \mathcal{N}\left(\mathbf{K}_{MN}\mathbf{K}_N^{-1}\mathbf{f}, \mathbf{K}_M - \mathbf{K}_{MN}\mathbf{K}_N^{-1}\mathbf{K}_{NM}\right). \tag{1.27}$$

Equation 1.25 is once again an integral of a product of two Gaussian densities. Thus, the predictive density is a Gaussian:

$$p(\mathbf{f}_\star|\mathbf{X}_\star, \mathbf{X}, \mathbf{y}, \theta) = \mathcal{N}(\boldsymbol{\mu}_\star, \boldsymbol{\Sigma}_\star). \tag{1.28}$$

It can be shown, using identities in Appendix A, that

$$\boldsymbol{\mu}_\star = \mathbf{K}_{MN}\left(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\right)^{-1}\mathbf{y}, \tag{1.29}$$

$$\boldsymbol{\Sigma}_\star = \mathbf{K}_M - \mathbf{K}_{MN}\left(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\right)^{-1}\mathbf{K}_{NM}. \tag{1.30}$$

The marginal likelihood expression in Equation 1.21 and the predictive mean and covariances in Equations 1.29 and 1.30 are the bread and butter of Gaussian process regression and are summarized below:

Gaussian Process Regression Equations

$$\begin{aligned}
\log Z(\theta) &= -\frac{1}{2}\left(\mathbf{y}^\top(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N)^{-1}\mathbf{y} + \log(\det((\mathbf{K}_N + \sigma_n^2\mathbf{I}_N))) + N\log(2\pi)\right) \\
\boldsymbol{\mu}_\star &= \mathbf{K}_{MN}\left(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\right)^{-1}\mathbf{y}. \\
\boldsymbol{\Sigma}_\star &= \mathbf{K}_M - \mathbf{K}_{MN}\left(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\right)^{-1}\mathbf{K}_{NM}.
\end{aligned}$$

Covariance functions

Although we have introduced GP regression for any valid covariance function[1], many of the results in this thesis will involve a couple of frequently used kernels. The common characteristic of the covariance functions used is that they are all functions of a (scaled) distance δ between inputs $\mathbf{x}$ and $\mathbf{x'}$, where

$$\delta^2 = (\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Lambda}(\mathbf{x} - \mathbf{x}'), \tag{1.31}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix with $\Lambda(d,d) \equiv 1/\ell_d^2$ for $d = 1, \ldots, D$. Covariance functions which are a function of absolute distance between input locations are known as *stationary, isotropic* kernels because they are invariant to translating or rotating the observations in input space. The *squared exponential* covariance function[2] is probably the most popular example because it gives rise to very smooth (infinitely mean-square differentiable) functions, and is defined as follows[3]:

$$k(\mathbf{x}, \mathbf{x}') \equiv k(\delta) = \sigma_f^2 \exp\left(-\frac{\delta^2}{2}\right). \tag{1.32}$$

The Matérn(ν) family of covariance functions are used when control over differentiability is required. The functions generated by the Matérn kernel are k-times

[1] Also referred to in the literature as a *kernel* in order to highlight the connection to frequentist kernel methods.

[2] In order to remain consistent with the literature we will stick to the term "squared exponential", while noting that a better name would be *exponentiated quadratic*.

[3] If the covariance function is isotropic we will "overload" it in the text by sometimes referring to as a function of δ only.

mean-square differentiable if and only if $\nu > k$. Expressions for the Matérn kernel for $\nu = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}$ are given below (at half-integer values of ν, the expressions become considerably simpler):

$$k_{\nu=1/2}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp(-\delta), \tag{1.33}$$

$$k_{\nu=3/2}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 (1 + \sqrt{3}\delta) \exp\left(-\sqrt{3}\delta\right), \tag{1.34}$$

$$k_{\nu=5/2}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \left(1 + \sqrt{5}\delta + \frac{1}{3}(\sqrt{5}\delta)^2\right) \exp\left(-\sqrt{5}\delta\right), \tag{1.35}$$

$$k_{\nu=7/2}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \left(1 + \sqrt{7}\delta + \frac{2}{5}(\sqrt{7}\delta)^2 + \frac{1}{15}(\sqrt{7}\delta)^3\right) \exp\left(-\sqrt{7}\delta\right). \tag{1.36}$$

The kernel in Equation 1.33 is also known as the *exponential* covariance function and gives rise to the *Ornstein-Uhlenbeck* process which is not mean-square differentiable yet mean-square continuous. It can be shown that in the limit where $\nu \to \infty$ the Matérn kernel converges to the squared-exponential. In practice, since it is hard to learn high frequency components from noisy observations, the performance of a prior supporting infinitely-differentiable functions will not be significantly different from one supporting up to, say, three-times differentiable functions. For formal definitions of mean-square differentiability and continuity refer to Papoulis et al. [2002].

Figure 1.3 shows samples of function values drawn from GP priors implied by the squared-exponential and Matérn kernels for different values of σ_f^2 and ℓ_1 over a scalar input space. Clearly, $[\ell_1, \ldots, \ell_D]$ control the characteristic *lengthscales* of the function along each input dimension – i.e., how rapidly they wiggle in each direction. The parameter σ_f^2 is simply a scaling of the covariance and thus controls the *amplitude* of the functions generated. Notice how, with $D + 1$ parameters, we have been able to capture most intuitive *functional* properties succinctly. This is a major advantage of GP regression. An example regression with synthetic data is shown for the squared-exponential covariance in Figure 1.4.

Many standard regression models can be cast as a GP with a specific covariance function. This is because *any* regression model which implicitly assigns a Gaussian density to latent function values is by definition doing what a GP does. Looking back at the previous section one can easily see (e.g. from Equation 1.9) that the parametric linear model is assigning the Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{\Phi}^\top \mathbf{\Sigma} \mathbf{\Phi})$ to the functions values at the training inputs, where $\mathbf{\Phi} \equiv [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N]$. Thus this model is a GP

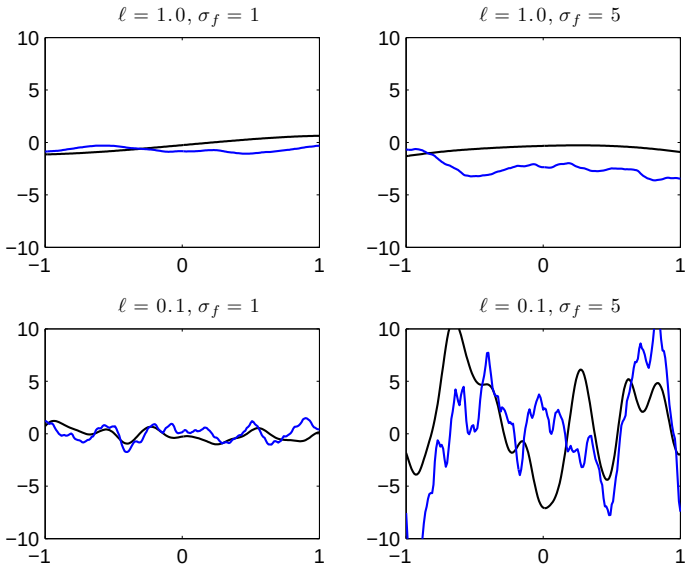


Figure 1.3: Functions drawn from squared-exponential (in black) and Matérn(3/2) (in blue) covariance functions for different hyperparameter values. Matérn(3/2) kernels give rise to "rougher" function draws as they are only once-differentiable.

with covariance function $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \mathbf{\Sigma} \phi(\mathbf{x})$. Many more examples linking a variety of regression techniques (ranging from neural networks to splines) to GPs is given in Rasmussen and Williams [2006].

1.2 Generalised Gaussian Process Regression [▷▷]

In the standard GP regression setting, it is assumed that the likelihood is a fully-factorized Gaussian, i.e.:

$$p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) = \prod_{i=1}^{N} \mathcal{N}(y_i; f_i, \sigma_n^2). \tag{1.37}$$

We will refer to the problem where the likelihood is fully-factorized but *non-Gaussian* as *generalised* GP regression, i.e.:

$$p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) = \prod_{i=1}^{N} \underbrace{p(y_i|f_i, \eta)}_{\text{Non-Gaussian factor}} . \tag{1.38}$$

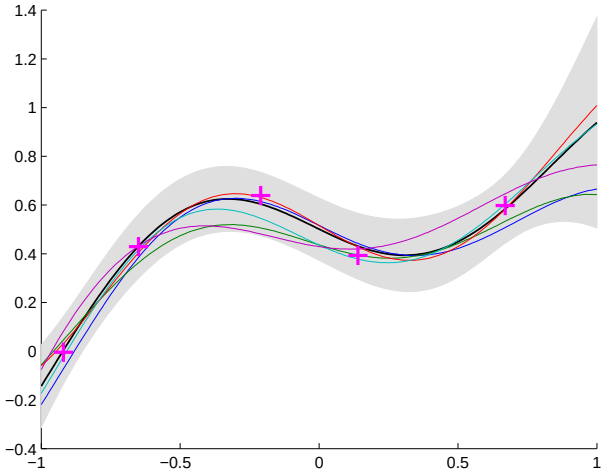


Figure 1.4: An example GP regression with the squared-exponential kernel. The bold black line shows the marginal posterior means and their 95% confidence intervals are show in greyscale. The coloured lines represent 5 samples from the posterior.

This extension captures a very large number of modelling tasks. For example, in the case of binary classification, where the targets $y_i \in \{-1, 1\}$ one cannot use a Gaussian. Instead we would like to be able to use likelihoods such as:

$$p(y_i = 1|f_i, \eta) = 1 - p(y_i = -1|f_i, \eta) = \frac{1}{1 + \exp(-f_i)}, \tag{1.39}$$

or

$$p(y_i = 1|f_i, \eta) = \Phi(y_i f_i), \tag{1.40}$$

where Φ is the Gaussian CDF and η is empty. In a task where we would like to model count data, we may want a likelihood such as:

$$p(y_i|f_i, \eta) = \text{Poisson}(\exp(f_i)), \tag{1.41}$$

and so on.

Much as in the case for standard regression, we would like to be able to compute the following key quantities. The first is the posterior distribution of $\mathbf{f}$ at the training inputs:

$$p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta) = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X}, \theta)}{p(\mathbf{y}|\mathbf{X}, \theta)}, \tag{1.42}$$

where $p(\mathbf{y}|\mathbf{X}, \theta)$ is the marginal likelihood of the data, used for the purposes of model comparison to select suitable θ:

$$p(\mathbf{y}|\mathbf{X}, \theta) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X}, \theta)d\mathbf{f}. \tag{1.43}$$

In addition, we would like to have a predictive distribution at unseen input locations $\mathbf{X}_\star$:

$$p(\mathbf{y}_\star|\mathbf{X}, \mathbf{y}, \mathbf{X}_\star, \theta) = \int p(\mathbf{y}_\star|\mathbf{f}_\star)p(\mathbf{f}_\star|\mathbf{X}, \mathbf{y}, \mathbf{X}_\star, \theta)d\mathbf{f}_\star, \tag{1.44}$$

where

$$p(\mathbf{f}_\star|\mathbf{X}, \mathbf{y}, \mathbf{X}_\star, \theta) = \int p(\mathbf{f}_\star|\mathbf{f}, \mathbf{X}_\star, \theta)p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta)d\mathbf{f}. \tag{1.45}$$

It is sometimes deemed sufficient to compute

$$\hat{\mathbf{f}}_\star \equiv \mathbb{E}\left(\mathbf{f}_\star|\mathbf{X}, \mathbf{y}, \mathbf{X}_\star, \theta\right), \tag{1.46}$$

and the associated predictive probabilities $\hat{\mathbf{p}} \equiv p(\mathbf{y}_\star|\hat{\mathbf{f}}_\star)$. For certain tasks, such as binary classification, labelling test cases using $\hat{\mathbf{p}}$ gives the same test error rate (i.e., the proportion of cases mislabelled) as that computed using Equation 1.44, so we do not lose much by using $\hat{\mathbf{p}}$ instead of the "correct" predictive probabilities given in Equation 1.44.

Given the likelihood $p(\mathbf{y}|\mathbf{f})$ is non-Gaussian, one has to resort to approximate inference techniques for the computation of all these quantities. The field of approximate inference is a very large one, ranging from MCMC to variational Bayes. Two of the most commonly used approximate inference techniques for generalised GP regression include Laplace's Approximation, which we summarize below, and the Expectation Propagation (EP) algorithm due to Minka [2001], which is introduced in Chapter 2 in the context of efficient generalised Gauss-Markov processes. For a good reference on how to use both these techniques in the context of (unstructured) GP regression, see Rasmussen and Williams [2006]. For an in-depth comparison of the approximations provided by these two methods, see Nickisch and Rasmussen [2008]. We will apply both Laplace's approximation and EP in the context of several structured GP models in Chapters 2 and 4. In this thesis, we will be focussing primarily on the generalised regression task involving binary classification, although

most of the techniques presented will be applicable in the general case.

Laplace's Approximation

In Laplace's approximation, the non-Gaussian posterior in Equation 1.42 is approximated as follows:

$$p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta) \approxeq \mathcal{N}(\hat{\mathbf{f}}, \mathbf{\Lambda}^{-1}), \tag{1.47}$$

where $\hat{\mathbf{f}} \equiv \arg\max_{\mathbf{f}} p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta)$ and $\Lambda \equiv -\nabla\nabla \log p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta)|_{\mathbf{f}=\hat{\mathbf{f}}}$. Let's define the following objective:

$$\Omega(\mathbf{f}) \equiv \log p(\mathbf{y}|\mathbf{f}) + \log p(\mathbf{f}|\mathbf{X}, \theta). \tag{1.48}$$

Clearly, $\hat{\mathbf{f}}$ can be found by applying Newton's method to this objective. The central iteration of Newton's method is:

$$\mathbf{f}^{(k+1)} \leftarrow \mathbf{f}^{(k)} - \left(\nabla\nabla\Omega(\mathbf{f}^{(k)})\right)^{-1} \nabla\Omega(\mathbf{f}^{(k)}). \tag{1.49}$$

Given $\mathbf{K}_N$ is the training set covariance matrix, it is straightforward to show that:

$$\nabla\Omega(\mathbf{f}) = \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f}) - \mathbf{K}_N^{-1}\mathbf{f}, \tag{1.50}$$

$$\nabla\nabla\Omega(\mathbf{f}) = \underbrace{\nabla\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})}_{\equiv -\mathbf{W}} - \mathbf{K}_N^{-1}, \tag{1.51}$$

$$\mathbf{f}^{(k+1)} \leftarrow \mathbf{f}^{(k)} + \left(\mathbf{K}_N^{-1} + \mathbf{W}\right)^{-1} \left(\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}} - \mathbf{K}_N^{-1}\mathbf{f}^{(k)}\right). \tag{1.52}$$

Convergence is declared when there is a negligible difference between $\mathbf{f}^{(k+1)}$ and $\mathbf{f}^{(k)}$. Newton's method is guaranteed to converge to the global optimum, given the objective in Equation 1.48 represents a convex optimisation problem w.r.t. $\mathbf{f}$. This is true for many cases of generalised GP regression, as the log-likelihood term is usually concave for many standard likelihood functions, and the prior term is Gaussian and therefore log-concave, giving a typical example of a convex optimisation problem, as illustrated in Boyd and Vandenberghe [2004].

Notice that we can use the second-order Taylor expansion of $\exp(\Omega(\mathbf{f}))$ about $\hat{\mathbf{f}}$ to obtain the following approximation to the true marginal likelihood given in 1.43:

$$p(\mathbf{y}|X,\theta) = \int \exp(\Omega(\mathbf{f}))d\mathbf{f} \tag{1.53}$$

$$\approxeq \exp(\Omega(\hat{\mathbf{f}})) \int \exp\left(-\frac{1}{2}(\mathbf{f}-\hat{\mathbf{f}})^\top \mathbf{\Lambda}(\mathbf{f}-\hat{\mathbf{f}})\right) d\mathbf{f}. \tag{1.54}$$

Thus,

$$\log p(\mathbf{y}|X) \approxeq \Omega(\hat{\mathbf{f}}) - \frac{1}{2}\log\det\mathbf{\Lambda} + \frac{N}{2}\log(2\pi). \tag{1.55}$$

Also, note that since we have a Gaussian approximation to $p(\mathbf{f}|\mathbf{y},\mathbf{X},\theta)$, the predictive distribution in Equation 1.45 can be computed using standard Gaussian identities, although further approximations may be required to compute the expression in Equation 1.44. For further details on these issues, see Rasmussen and Williams [2006].

1.3 The Computational Burden of GP Models

Gaussian Process Regression

GPs achieve their generality and mathematical brevity by basing inference on an N-dimensional Gaussian over the latent function values $\mathbf{f}$. Algorithm 1 gives the pseudo-code used for computing the marginal likelihood $Z(\theta)$ and the predictive density $\mathcal{N}(\boldsymbol{\mu}_\star, \mathbf{\Sigma}_\star)$, and shows that the computational cost of working with an N-dimensional Gaussian is $\mathcal{O}(N^3)$ in execution time and $\mathcal{O}(N^2)$ in memory.

The complexity of the Cholesky decomposition central to GP regression[1] is quite severe for large datasets. For example, a dataset containing 20,000 observations (this is not even classed as large nowadays) will take *hours* to return. Of course, this is assuming that it runs *at all* – 20,000 observations will require roughly 7GB of memory, and many operating systems will refuse to run a program with such memory requirements! One can argue that the exponential increase in computing power will allow computational resources to catch up with the complexity inherent in GP models. However, typical dataset sizes in industry are also experiencing

[1]The Cholesky decomposition is the preferred way to compute $\mathbf{K}^{-1}\mathbf{y}$ due to its numerical stability.

Algorithm 1: Gaussian Process Regression

inputs : Training data $\{\mathbf{X}, \mathbf{y}\}$, Test locations $\mathbf{X}_\star$, Covariance Function `covfunc`, hyperparameters $\theta = [\sigma_f^2; \ell_1 \dots \ell_D; \sigma_n^2]$

outputs: Log-marginal likelihood $\log Z(\theta)$, predictive mean $\mu_\star$ and covariance $\Sigma_\star$

1 $\mathbf{K}_N \leftarrow$ `covfunc(`$\mathbf{X}$`,`θ`);` `//Evaluate covariance at training inputs`
2 $\mathbf{K}_N \leftarrow \mathbf{K}_N + \sigma_n^2 \mathbf{I}_N$; `//Add noise`
3 $\mathbf{L} \leftarrow$ `Cholesky(`$\mathbf{K}_N$`);` `//`$\mathcal{O}(N^3)$ `time,` $\mathcal{O}(N^2)$ `memory`
4 $\boldsymbol{\alpha} \leftarrow$ `Solve(`$\mathbf{L}^\top$`, Solve(`$\mathbf{L}$`,` $\mathbf{y}$`));` `//`$\boldsymbol{\alpha} = (\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N)^{-1}\mathbf{y}$
5 $[\mathbf{K}_M, \mathbf{K}_{MN}] \leftarrow$ `covfunc(`$\mathbf{X}$`,` $\mathbf{X}_\star$`,` θ`);`
6 $\mathbf{V} \leftarrow$ `Solve(`$\mathbf{L}$`,` $\mathbf{K}_{MN}$`);`
7 $\boldsymbol{\mu}_\star \leftarrow \mathbf{K}_{MN}\boldsymbol{\alpha}$;
8 $\boldsymbol{\Sigma}_\star \leftarrow \mathbf{K}_M - \mathbf{V}^\top\mathbf{V}$;
9 $\log Z(\theta) \leftarrow -\frac{1}{2}\left(\mathbf{y}^\top\boldsymbol{\alpha} + \sum_{i=1}^N L(i,i) + \frac{N}{2}\log(2\pi)\right)$;

rapid growth as more and more aspects of the physical world are computerized and recorded. As a result, there is a strong incentive for applied machine learning research to develop techniques which scale in an acceptable way with dataset size N.

A significant amount of research has gone into *sparse approximations* to the full GP regression problem. In the full GP, N function values located at the given input locations give rise to the dataset. Most sparse GP methods relax this by assuming that there are actually P latent function values, $\bar{\mathbf{f}}$, located at "representative" input locations[1], $\bar{\mathbf{X}}$, giving rise to the N input-output pairs, where $P \ll N$. The training targets $\mathbf{y}$ are used to derive a posterior distribution over $\bar{\mathbf{f}}$ and this posterior is used to perform model selection and predictions. This has the effect of reducing run-time complexity to $\mathcal{O}(P^2N)$ and memory complexity to $\mathcal{O}(PN)$, a noteworthy improvement. The pseudo-input locations are learned, manually selected or set to be some subset of the original input locations, depending on the algorithm. A good example of this type of method is given by the FITC approximation, see Snelson and Ghahramani [2006]. Other sparse GP techniques involve *low-rank* matrix approximations for solving the system $(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N)^{-1}\mathbf{y}$ (e.g., see Williams and Seeger [2001]) and frequency domain (e.g., Lázaro-Gredilla [2010]) methods. For an excellent review of

[1] also known as *pseudo-inputs*.

sparse GP approximations, see Quiñonero-Candela and Rasmussen [2005].

The improved computational complexity of sparse GPs comes at the expense of modelling accuracy, and, at present, an increase in the number of failure modes. The former is inevitable: we are throwing away information, albeit intelligently. An example of the latter can be seen in the FITC algorithm. The need to select or learn $\bar{\mathbf{X}}$ runs the risk of pathologically interacting with the process of hyperparameter learning and reviving the problem of overfitting as explained in Snelson and Ghahramani [2006]. A very clear advantage of these methods however is their *generality* – they work with *any* valid covariance function, and with an arbitrary set of inputs from $\mathbb{R}^D$.

Generalised Gaussian Process Regression

The runtime complexity of generalised GP regression is $\mathcal{O}(N^3)$ although usually with a constant that is a multiple of the one for the standard regression task. The memory complexity is similar and $\mathcal{O}(N^2)$. Thus, unsurprisingly, generalised GP regression suffers from similar scaling limitations. For example, referring to Equation 1.52, we can see that each Newton iteration of Laplace's approximation runs in $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ space. A similar iterative scheme comes about as a result of applying EP – see Rasmussen and Williams [2006].

Despite suffering from higher computational demands, there is a much smaller number of techniques in the literature that deal with improving the complexity of generalised GP regression. It is fair to say that this is an active area of GP research. Current solutions can be characterized as attempts to combine the idea of sparse GP methods with approximate inference methods necessary to handle non-Gaussian likelihoods. A good example is the Informative Vector Machine (IVM), which combines a greedy data subset selection strategy based on information theory and assumed density filtering (which is used to deal with non-Gaussianity) Lawrence et al. [2003]. Another example is the Generalised FITC approximation of Naish-Guzman and Holden [2007] which combines FITC with EP. Similar to their regression counterparts, these methods trade-off modelling accuracy with reduced complexity, although arguably they suffer from a larger array of failure modes. Furthermore, these methods do not assume any structure in the underlying covariance,

and thus are, in a sense, orthogonal to the approaches we will use in this thesis to improve efficiency in the generalised regression setting.

1.4 Structured Gaussian Processes

The central aim of this thesis is to demonstrate that there exist a variety of useful *structured* GP models which allow *efficient and exact* inference, or at the very least allow a superior accuracy/run-time trade-off than what is currently available for generic GP regression. We define what we mean by a structured GP as follows:

Definition 2. *A Gaussian process is* structured *if its marginals* $p(\mathbf{f}|\mathbf{X}, \theta)$ *contain exploitable structure that enables reduction in the computational complexity of performing regression.*

A somewhat trivial example is given by the GP with the linear regression covariance function introduced in the previous section, for which $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}^\top \boldsymbol{\Sigma} \boldsymbol{\phi}$ and thus $\mathbf{K}_N = \boldsymbol{\Phi}^\top \boldsymbol{\Sigma} \boldsymbol{\Phi}$. Let's assume the size of $\boldsymbol{\Phi}$ is $F \times N$ with $F \ll N$ and that, as usual, $\mathbf{K}_{\text{noise}}$ is diagonal. We can then compute the troublesome quantities $(\mathbf{K}_N + \mathbf{K}_{\text{noise}})^{-1}\mathbf{y}$, $(\mathbf{K}_N + \mathbf{K}_{\text{noise}})^{-1}\mathbf{K}_{NM}$ and $\det(\mathbf{K}_N + \mathbf{K}_{\text{noise}})$ in $\mathcal{O}(NF^2)$ time and $\mathcal{O}(NF)$ memory by appealing to the matrix inversion lemma and the associated matrix determinant lemma:

$$(\mathbf{K}_N + \mathbf{K}_{\text{noise}})^{-1} = \mathbf{K}_{\text{noise}}^{-1} - \mathbf{K}_{\text{noise}}^{-1}\Phi^\top \left(\Sigma^{-1} + \Phi \mathbf{K}_{\text{noise}}^{-1}\Phi^\top\right)^{-1} \Phi \mathbf{K}_{\text{noise}}^{-1}, \quad (1.56)$$

$$\det(\mathbf{K}_N + \mathbf{K}_{\text{noise}}) = \det\left(\Sigma^{-1} + \Phi \mathbf{K}_{\text{noise}}^{-1}\Phi^\top\right) \det(\Sigma) \det(\mathbf{K}_{\text{noise}}). \quad (1.57)$$

Clearly, exact inference can be performed for such a structured GP model much more efficiently. The reduction in complexity is clearly a result of the *low-rank* nature of the (noise-free) covariance function. In the special case where $\phi(\mathbf{x}) = \mathbf{x}$ the covariance function gives rise to covariance matrices of rank D. The improvement in computation comes as a direct consequence of the rather strong assumption that the input-output relationship is linear.

In this thesis, we present a set of algorithms that make different types of assumptions, some of which turn out to be strong for certain datasets, and some which are perfectly suitable for a host of applications.

1.5 Outline

Over the course of the thesis we will make four different types of assumptions which turn out to have significant efficiency implications for standard and generalised GP regression.

The first is where we consider Gaussian processes which are also *Markovian*. This assumption results in a computationally far more attractive factorization of the marginal $p(\mathbf{f}|\mathbf{X}, \theta)$, as demonstrated in Hartikainen and Särkkä [2010]. As Gauss-Markov processes are strictly defined over "time" (or any scalar input space) we will initially restrict attention to regression on $\mathbb{R}$ (Chapter 2), although we will also explore GP models which allow the efficiency gains to be carried over to regression on $\mathbb{R}^D$ in Chapter 4. We will show that, especially in 1D, the Markovian assumption is indeed a very weak one and that, as a result, we do not lose much in terms of generality. Despite this generality, the Markovian assumption allows us to construct exact (or near-exact) algorithms that have $\mathcal{O}(N \log N)$ runtime and $\mathcal{O}(N)$ memory requirements, for both standard and generalised GP regression.

Secondly, we attempt to introduce block structure into covariance matrices by means of assuming *nonstationarity*. Note that nonstationarity generally adds to the computational load, as is apparent in many case studies such as Adams and Stegle [2008] and Karklin and Lewicki [2005]. However, by limiting attention to a *product partition* model, as introduced in Barry and Hartigan [1992], we can exploit the nonstationary nature of the problem to improve complexity. Furthermore, a product partition model allows us to partition the input space into segments and fit independent GPs to each segment, in a manner similar to Gramacy [2005]. Dealing with nonstationarity can have the added benefit of actually producing predictions which are more accurate than the usual stationary GP, although approximations will be needed when working with the posterior over input space partitionings. We restrict attention to sequential time-series modelling in this thesis (Chapter 3), although, again, extensions as in Chapter 4 are applicable for nonstationary regression on $\mathbb{R}^D$. The assumption of nonstationarity is suitable to many real-world applications.

In Chapter 4 we use the assumption of *additivity* to extend the efficiency gains obtained in Chapter 2 to input spaces with dimension $D > 1$. This assumption

would be suitable to cases where the underlying function values can be modelled well as a sum of D univariate functions. Somewhat unsurprisingly, inference and learning for a GP with an additive kernel can be decomposed into a sequence of scalar GP regression steps. As a result, with the added assumption that all additive components can be represented as Gauss-Markov processes, additive GP regression can also be performed in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space. Similar efficiency gains can be also be obtained for Generalised GP regression with Laplace's approximation. The assumption of additivity frequently turns out to be too strong an assumption for many large-scale real-world regression tasks. It is possible to relax this assumption by considering an additive model over a space linearly-related to the original input space. We will show that learning and inference for such a model can be performed by using *projection pursuit* GP regression, with no change to computational complexity whatsoever.

Another interesting route to obtaining structured GPs involves making assumptions about *input locations*. It is a well-known fact that GP regression can be done in $\mathcal{O}(N^2)$ time[1] and $\mathcal{O}(N)$ memory when the inputs are uniformly spaced on $\mathbb{R}$ using *Toeplitz matrix* methods (see, e.g. Zhang et al. [2005] and Cunningham et al. [2008]). What has been unknown until now is that an analogue of this result exists for regression on $\mathbb{R}^D$ for any D. In this case one assumes that the inputs lie on a multidimensional *grid*. Furthermore, the grid locations need not be uniformly spaced as in the 1D case. We demonstrate how to do exact inference in $\mathcal{O}(N)$ time and $\mathcal{O}(N)$ space for any tensor product kernel (most commonly-used kernels are of this form). This result is proved and demonstrated in Chapter 5.

We conclude the thesis in Chapter 6 by discussing some interesting extensions one could follow up to build more GP implementations which scale well.

[1] More involved $\mathcal{O}(N \log N)$ algorithms also exist!

Chapter 2

Gauss-Markov Processes for Scalar Inputs

In this chapter, we firstly aim to drive home the message that Gauss-Markov processes on $\mathbb{R}$ can often be expressed in a way which allows the construction of highly efficient algorithms for regression and hyperparameter learning. Note that most the ideas we present for efficient Gauss-Markov process regression are not new and can be found in, e.g., Kalman [1960], Bar-Shalom et al. [2001], Grewal and Andrews [2001], Hartikainen and Särkkä [2010], Wecker and Ansley [1983]. Due in part to the separation between systems and control theory research and machine learning, the SDE viewpoint of GPs has not been exploited in commonly used GP software packages, such as Rasmussen and Nickisch [2010]. We are thus motivated to provide a concise and unified presentation of these ideas in order to bridge this gap. Once this is achieved, we contribute an extension that handles non-Gaussian likelihoods using EP. The result is a scalar GP classification algorithm which runs in $\mathcal{O}(N \log N)$ runtime and $\mathcal{O}(N)$ space. For appropriate kernels, this algorithm returns exactly the same predictions and marginal likelihood as the standard GP classifier (which also uses EP).

2.1 Introduction

The goal of this chapter is to demonstrate that many GP priors commonly employed over $\mathbb{R}$ can be used to perform (generalised) regression with efficiency orders of

magnitude superior to that of standard GP algorithms. Furthermore, we will derive algorithms which scale as $\mathcal{O}(N \log N)$ in runtime and $\mathcal{O}(N)$ in memory usage. Such gains come about due to the exact (or close to exact) correspondence between the function prior implied by a GP with some covariance function $k(\cdot, \cdot)$, and the prior implied by an order-m linear, stationary stochastic differential equation (SDE), given by:

$$\frac{d^m f(x)}{dx^m} + a_{m-1}\frac{d^{m-1} f(x)}{dx^{m-1}} + \cdots + a_1 \frac{df(x)}{dx} + a_0 f(x) = w(x), \tag{2.1}$$

where $w(x)$ is a white-noise process with mean 0 and covariance function $k_w(x_i, x_j) = q\delta(x_i - x_j)$. Since white-noise is clearly stationary, we will also refer to its covariance function as a function of $\tau \equiv x_i - x_j$, i.e., $k(\tau) = q\delta(\tau)$. q is the variance of the white-noise process driving the SDE. Note that x can be any scalar input, including time. Because $w(x)$ is a stochastic process, the above SDE will have a solution which is also a stochastic process, and because the SDE is linear in its coefficients, the solution will take the form of a *Gaussian* process. This is not an obvious result and we will expand on it further in Section 2.3. Furthermore, the solution function $f(x)$ will be distributed according to a GP with a covariance function which depends on the coefficients of the SDE. We can rewrite Equation 2.1 as a *vector Markov process*:

$$\frac{d\mathbf{z}(x)}{dx} = \mathbf{A}\mathbf{z}(x) + \mathbf{L}w(x), \tag{2.2}$$

where $\mathbf{z}(x) = \left[f(x), \frac{df(x)}{dx}, \ldots, \frac{d^{m-1}f(x)}{dx^{m-1}}\right]^\top$, $\mathbf{L} = [0, 0, \ldots, 1]^\top$[1], and

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ -a_0 & \cdots & \cdots & -a_{m-2} & -a_{m-1} \end{bmatrix}. \tag{2.3}$$

Equation 2.2 is significant because it dictates that given knowledge of $f(x)$ and its m derivatives at input location x, we can *throw away* all information at any other input location which is smaller than x. This means that by jointly sorting

[1]Note that in the exposition of the state-space representation of a GP, we will use capital letters for some vectors in order to remain consistent with the SDE literature. The motivation behind this stems from the fact that in a more general state-space model these objects can be matrices.

our training and test sets and by augmenting our latent-space to include derivatives of the underlying function in addition to the function value itself, we can induce *Markovian* structure on the graph underlying GP inference. All efficiency gains demonstrated in this chapter will arise as a result of the Markov structure inherent in Equation 2.2. In fact, the efficiency gains are so significant that the asymptotic runtime cost is dominated by the operation of sorting the training and test inputs! Figure 2.1 illustrates the reformulation of GP regression into a Markovian *state-space* model (SSM).

In order to construct the SSM corresponding to a particular covariance function, it is necessary to derive the implied SDE. In Section 2.2 we illustrate the SDEs corresponding to several commonly used covariance functions including *all* kernels from the Matérn family and spline kernels, and good approximate SDEs corresponding to the squared-exponential covariance function. Once the SDE is known, we can solve the vector differential equation in Equation 2.2 and therefore compute the time and measurement update equations that specify all the necessary conditional distributions required by the graph in Figure 2.1. The Kalman filtering and Rauch-Tung-Striebel (RTS) smoothing algorithms, which correspond to performing belief propagation on the graph using forward filtering and backward smoothing sweeps, can then be used to perform GP regression efficiently, see Kalman [1960] and Rauch et al. [1965]. Because we are dealing with Gaussian processes, all the conditional distributions involved are Gaussian. We will present the details on how to compute these given a GP prior in the form of an SDE in Section 2.3.

2.2 Translating Covariance Functions to SDEs

In Chapter 1 we described the GP prior in terms of its covariance function (and assumed its mean function be to be zero). In this Chapter, we would like to view the same prior from the perspective of Equation 2.1. In order to analyze whether or not it is possible to write down a SDE for a given covariance function and, if so, how to derive it, requires that we understand how to go in the opposite direction. In other words, given an SDE as in Equation 2.1, what is the implied covariance function of $f(x)$ given that the covariance function of $w(x)$ is $k_w(\tau) = q\delta(\tau)$?

The first step is to view the SDE as representing a linear, time-invariant (LTI)

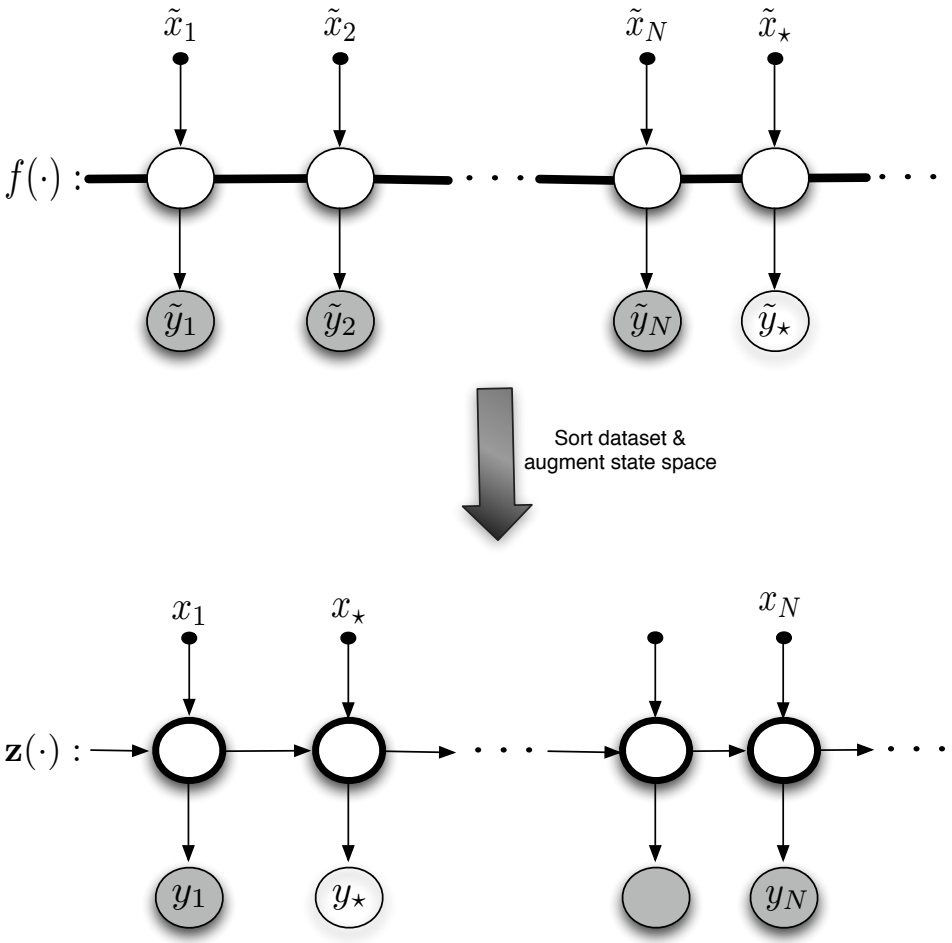


Figure 2.1: Illustrating the Transformation of GP Regression into Inference on a vector Markov process using Graphical Models. Unshaded nodes indicate latent variables, shaded ones are observed variables and solid black dots are variables treated as known in advance. In standard GP regression (illustrated in the top half) the hidden state variables are the underlying function values. They are fully connected as illustrated by the bold line connecting them. By augmenting the latent state variables to include m derivatives, we derive a chain-structured graph (illustrated in the bottom half). Reduction from a fully connected graph to one where the maximal clique size is 2 results in large computational savings. $\mathbf{x}$ and $\mathbf{y}$ are $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ sorted in ascending order. We see that in this example that the test point is the second smallest input location in the whole dataset.

system, driven by white-noise, as shown in Figure 2.2. For a good introduction to LTI systems, refer to Oppenheim et al. [1996]. Conceptually, for every sample of white-noise with variance q, the LTI system produces a sample from a GP with

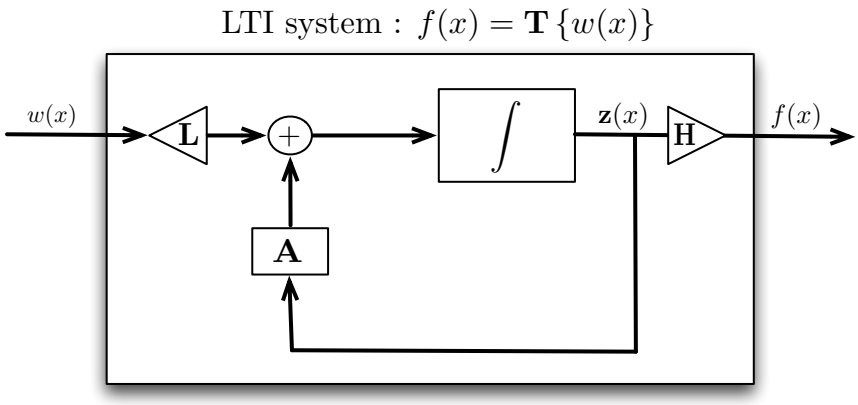


Figure 2.2: The LTI system equivalent to the SDE in Equation 2.2. The components represent the vectors $\mathbf{L}$ and $\mathbf{H}$ and the matrix $\mathbf{A}$ in Equation 2.2. Conceptually, the "integrator" integrates the system in Equation 2.2 over an infinitesimal interval.

some covariance function. In order to determine what this covariance function is, it is necessary to characterize the output produced for an arbitrary, *deterministic* input function, $\phi(x)$.

Because the internals of the black-box producing $f(x)$ from $w(x)$ are a sequence of linear operations, it is clear from this figure that, under appropriate initial conditions, $\mathbf{T}\{\sum_i \alpha_i \phi_i(x)\} = \sum_i \alpha_i \mathbf{T}\{\phi_i(x)\}$ for any α_i and any input function $\phi_i(x)$. As $\mathbf{T}$ does not change as function of x, "time-invariance" is also satisfied.

Every LTI system is fully characterized by its *impulse response function*, which is the output of the system when we input a unit impulse at time 0, i.e.:

$$h(t) = \mathbf{T}\{\delta(t)\}. \tag{2.4}$$

The reason we can fully characterize an LTI system using $h(t)$ is a consequence of the following theorem:

Theorem 2.1. *For any deterministic input $\phi(x)$,*

$$\mathbf{T}\{\phi(x)\} = \phi(x) \star h(x). \tag{2.5}$$

Proof. Since we can trivially write $\phi(x) = \int_{-\infty}^{\infty} \phi(x-\alpha)\delta(\alpha)d\alpha$ for any $\phi(x)$, we have

$$
\begin{aligned}
\mathbf{T}\left\{\phi(x)\right\} &= \mathbf{T}\left\{\int_{-\infty}^{\infty}\phi(x-\alpha)\delta(\alpha)d\alpha\right\}, \\
&= \int_{-\infty}^{\infty}\phi(x-\alpha)\mathbf{T}\left\{\delta(\alpha)\right\}d\alpha, && \text{[linearity]} \\
&= \int_{-\infty}^{\infty}\phi(x-\alpha)h(\alpha)d\alpha, && \text{[time-invariance]} \\
&\equiv \phi(x)\star h(x). && \square
\end{aligned}
$$

Taking the Fourier transform of both sides Equation 2.5 and using the convolution property, we can write:

$$
\mathcal{F}\left(\mathbf{T}\left\{\phi(x)\right\}\right) \equiv F(\omega) = \Phi(\omega)H(\omega), \tag{2.6}
$$

where $\Phi(\omega)$ is Fourier transform of the (arbitrary) input, $F(\omega)$ that of the output and $H(\omega)$ that of the impulse response function, which is commonly known as the *frequency response function*.

Using Theorem 2.1 it is straightforward to prove the following:

Theorem 2.2. *The covariance function of the output of an LTI system driven with a stochastic process with kernel $k_w(\tau)$ is given by:*

$$
k_f(\tau) = k_w(\tau)\star h(\tau)\star h(-\tau). \tag{2.7}
$$

Proof. Given that the system is linear and the mean function of the white-noise process is 0, it is clear that the mean function of the output process is also zero. Hence,

$$
\begin{aligned}
k_f(\tau) &\equiv \mathbb{E}\left(f(x)f(x+\tau)\right), \\
&= \mathbb{E}\left[\left(\int_{-\infty}^{\infty}w(x-\alpha)h(\alpha)d\alpha\right)f(x+\tau)\right], && \text{[2.1]} \\
&= \int_{-\infty}^{\infty}\mathbb{E}\left(w(x-\alpha)f(x+\tau)\right)h(\alpha)d\alpha,
\end{aligned}
$$

$$
\begin{aligned}
&= \int_{-\infty}^{\infty} \mathbb{E}\left(w(x-\alpha) \int_{-\infty}^{\infty} w(x+\tau-\beta)h(\beta)d\beta \right) h(\alpha)d\alpha, && [2.1] \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathbb{E}\left(w(x-\alpha)w(x+\tau-\beta) \right) h(\beta)h(\alpha)d\beta d\alpha, \\
&\equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} k_w(\tau-\beta-\gamma)h(\beta)h(-\gamma)d\beta d\gamma, && [\gamma \equiv -\alpha] \\
&\equiv k_w(\tau) \star h(\tau) \star h(-\tau). && \square
\end{aligned}
$$

Taking the Fourier transform of both sides of Equation 2.7 we can write:

$$
\begin{aligned}
S_f(\omega) &= S_w(\omega)H(\omega)H^*(\omega), && (2.8) \\
&= S_w(\omega)|H(\omega)|^2. && (2.9)
\end{aligned}
$$

where $S_w(\omega)$ and $S_f(\omega)$ are the respective Fourier transforms of $k_w(\tau)$ and $k_f(\tau)$, also known as the *spectral density functions* of the input and output processes. Since, for a white-noise process, $S_w(\omega) = q$ [1], the spectral density of the output GP becomes:

$$
S_f(\omega) = qH(\omega)H^*(\omega) = q|H(\omega)|^2. \tag{2.10}
$$

For linear, constant coefficient SDEs we can derive $H(\omega)$ analytically by taking the Fourier transform of both sides of Equation 2.1.

$$
\begin{gathered}
\sum_{k=0}^{m} a_k \frac{d^k f(x)}{dx^k} = w(x). \\
\Updownarrow \\
\sum_{k=0}^{m} a_k (i\omega)^k F(\omega) = W(\omega).
\end{gathered}
$$

Using Equation 2.6 we see directly that:

$$
H(\omega) = \frac{F(\omega)}{W(\omega)} = \left(\sum_{k=0}^{m} a_k (i\omega)^k \right)^{-1}. \tag{2.11}
$$

Combining Equations 2.10 and 2.11 we see that if the spectral density of our GP

[1] As the name implies!

can be written in the form

$$S_f(\omega) = \frac{q}{\text{polynomial in } \omega^2}. \tag{2.12}$$

then we can immediately write down the corresponding SDE and perform all necessary computations in $\mathcal{O}(N \log N)$ runtime and $\mathcal{O}(N)$ memory using the techniques described in Section 2.3.

2.2.1 The Matérn Family

The covariance function for the Matérn family (indexed by ν) for scalar inputs is given by:

$$k_\nu(\tau) = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} (\lambda\tau)^\nu B_\nu (\lambda\tau), \tag{2.13}$$

where B_ν is the modified Bessel function (see Abramowitz and Stegun [1964]) and we have defined $\lambda \equiv \frac{\sqrt{2\nu}}{\ell}$. We had already seen examples of this family in Equations 1.33 through to 1.36. Recall that ℓ is the characteristic lengthscale and σ_f^2 the signal variance. The Fourier transform of this expression and thus the associated spectral density is (see Rasmussen and Williams [2006]):

$$S(\omega) = \sigma_f^2 \frac{2\pi^{1/2}\Gamma(\nu + 1/2)\lambda^{2\nu}}{\Gamma(\nu)(\lambda^2 + \omega^2)^{\nu+1/2}} \Leftrightarrow \frac{q}{\text{polynomial in } \omega^2}. \tag{2.14}$$

The spectral density of the Matérn kernel is precisely of the form required to allow an SDE representation. All we have to do is to find the frequency response $H(\omega)$ giving rise to this spectral density and then use Equation 2.11 to find the coefficients a_k. Note that Equation 2.14 implies that

$$q = \sigma_f^2 \frac{2\pi^{1/2}\Gamma(\nu + 1/2)\lambda^{2\nu}}{\Gamma(\nu)}. \tag{2.15}$$

Using Equation 2.10 and 2.14 we can write:

$$
\begin{aligned}
H(\omega)H^*(\omega) &= \frac{S(\omega)}{q}, \\
&= (\lambda^2 + \omega^2)^{-(\nu+1/2)}, \\
&= (\lambda + i\omega)^{-(\nu+1/2)}(\lambda - i\omega)^{-(\nu+1/2)}.
\end{aligned}
$$

Thus,

$$
H(\omega) = (\lambda + i\omega)^{-(\nu+1/2)}. \tag{2.16}
$$

For example, when $\nu = 7/2$, the resulting SDE is given, in vector form, as follows:

$$
\frac{d\mathbf{z}(x)}{dx} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\lambda^4 & -4\lambda^3 & -6\lambda^2 & -4\lambda \end{bmatrix} \mathbf{z}(x) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} w(x). \tag{2.17}
$$

where the variance of $w(x)$ is q given by Equation 2.15.

Note that the Matérn kernel is well-defined for $\nu \in \mathbb{R}^+$, although it may not be possible to derive an analytic expression for its corresponding SDE for non-half-integer values of ν.

2.2.2 Approximating the Squared-Exponential Covariance

The squared-exponential is a kernel which is frequently used in practice, due in part to its intuitive analytic form and parameterisation. For completeness, we present ways in which one could approximate this covariance with an SDE, while noting that its support for infinitely differentiable functions makes it computationally rather unattractive.

The Fourier transform of a squared-exponential kernel is itself a squared-exponential of ω, i.e.:

$$
k(\tau) = \sigma_f^2 \exp\left(-\frac{\tau^2}{2\ell^2}\right) \Leftrightarrow S(\omega) = \sqrt{2\pi}\sigma_f^2 \ell \exp\left(-\frac{\omega^2\ell^2}{2}\right). \tag{2.18}
$$

We can write the spectral density in a form similar to Equation 2.12 using the Taylor

expansion of $\exp(x)$.

$$S(\omega) = \frac{\sqrt{2\pi}\sigma_f^2 \ell}{\left(1 + \frac{\omega^2\ell^2}{2} + \frac{1}{2!}\left(\frac{\omega^2\ell^2}{2}\right)^2 + \dots\right)}. \tag{2.19}$$

As the denominator is an infinitely long polynomial we cannot form an SDE which exactly corresponds to the SE kernel. This should come as no surprise since the SE covariance gives rise to infinitely differentiable functions. Therefore, conceptually, we would need to know an infinite number of derivatives to induce Markov structure. However, we can obtain an approximation by simply truncating the Taylor expansion at some maximum order M. By doing so, the approximate spectral density becomes a special case of Equation 2.12, and we can proceed to compute the frequency response function:

$$\begin{aligned} H(\omega)H^*(\omega) &= \frac{S(\omega)}{q}, \\ &= \left(1 + \frac{\omega^2\ell^2}{2} + \frac{1}{2!}\left(\frac{\omega^2\ell^2}{2}\right)^2 + \dots + \frac{1}{M!}\left(\frac{\omega^2\ell^2}{2}\right)^M\right). \end{aligned}$$

$H(\omega)$ is thus a polynomial of $i\omega$, however, unlike the case for the Matérn kernel, we cannot find the coefficients of this polynomial analytically and will therefore have to resort to numerical methods (for further details, see Hartikainen and Särkkä [2010]).

Alternatively, one may choose to assume that a member of the Matérn family with high enough ν (e.g. $\nu \geq 7/2$) is an acceptable approximation. This has the advantage of avoiding numerical routines for computing the SDE coefficients, at the expense of a small loss in approximation quality. Figure 2.3 illustrates the two alternatives for approximating the squared-exponential covariance.

2.2.3 Splines

Splines are regression models which are popular in the numerical analysis and frequentist statistics literatures (for a good introduction see Hastie et al. [2009]). Following Wahba [1990] and Rasmussen and Williams [2006], a connection can be made between splines of arbitrary order m and GPs. The frequentist smoothing problem

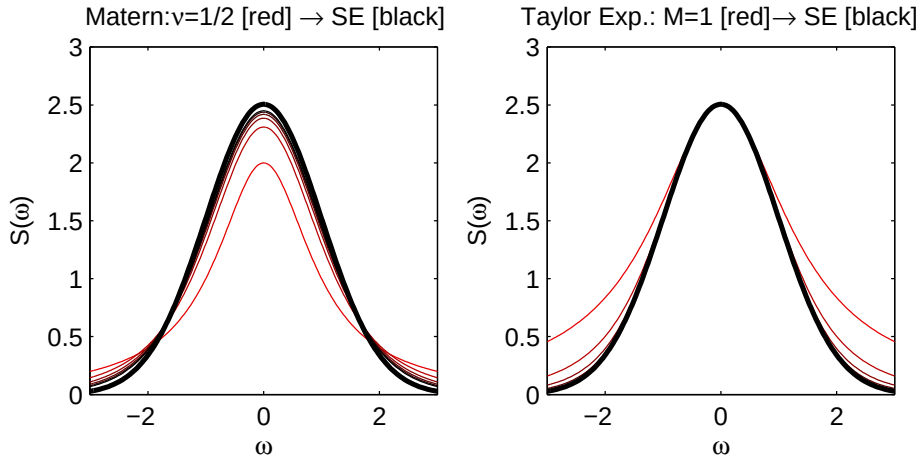


Figure 2.3: (Left) Spectral density of the Matérn kernels (with unit lengthscale and amplitude) for $\nu = [1/2, 3/2, \ldots, 11/2]$ in shades of red; true squared-exponential spectral density in black. (Right) Spectral density of the Taylor approximations (with unit lengthscale and amplitude) for $M = [1, \ldots, 6]$ in shades of red; true squared-exponential spectral density in black. For a given order of approximation the Taylor expansion is better, however, we cannot find coefficients analytically.

is to minimize the cost-functional

$$\Omega(f(\cdot)) \equiv \sum_{i=1}^{N}(f(x_i) - y_i)^2 + \lambda \int_a^b (f^{(m)}(x))^2 dx. \tag{2.20}$$

where λ is a regularization parameter (usually fit using cross-validation) which controls the trade-off between how well the function fits the targets (first term) and how wiggly/complex it is (second term). The minimization is over all possible functions $f(x)$ living in a Sobolev space of the interval $[a, b]$. For an excellent introduction on functional analysis see Kreyszig [1989]. Interestingly, one can obtain a unique global minimizer which can be written as a *natural spline*: a piecewise polynomial of order $2m - 1$ in each interval $[x_i, x_{i-1}]$ and a polynomial of order $m - 1$ in the intervals $[a, x_1]$ and $[x_N, b]$. A spline is generally not defined over all of $\mathbb{R}$ but over a finite interval from a to b, where $a < \min(x_1, \ldots, x_N)$ and $b > \max(x_1, \ldots, x_N)$. Therefore, without loss of generality, we can assume $a = 0$ and $b = 1$, requiring that

inputs be scaled to lie between 0 and 1. The natural spline solution is given by:

$$f(x) = \sum_{j=0}^{m-1} \beta_j x^j + \sum_{i=1}^{N} \gamma_i (x - x_i)_+^{2m-1}, \text{ where } (z)_+ \equiv \begin{cases} z & \text{if } z > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.21}$$

The parameters $\boldsymbol{\beta} \equiv \{\beta_j\}$ and $\boldsymbol{\gamma} \equiv \{\gamma_i\}$ can be determined in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space due to the Reinsch algorithm described in Green and Silverman [1994] and Reinsch [1967]. This algorithm also requires the sorting of inputs and the attains its efficiency by re-expressing the optimization as a linear system which is band-diagonal (with a band of width $2m + 1$). The *representer theorem* Kimeldorf and Wahba [1971] indicates that the solution can be written in the following form:

$$f(x) = \sum_{j=0}^{m-1} \beta_j x^j + \sum_{i=1}^{N} \alpha_i R(x, x_i), \tag{2.22}$$

where $R(x, x')$ is a positive definite function, as shown in Seeger [1999].

Both the nature of the solution and the complexity of attaining it have analogues in an equivalent GP (Bayesian) construction. The key to obtaining an efficient algorithm again lies in the SDE representation of GP priors, although in this case it won't be possible to represent the SDE as an LTI system due to the nonstationarity of the spline kernel (see below). Let's consider the following generative model representing our prior:

$$\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \mathbf{B}), \tag{2.23}$$

$$f_{\text{sp}}(x) \sim \mathcal{GP}(0, k_{\text{sp}}(x, x')), \tag{2.24}$$

$$f(x) = \sum_{j=0}^{m-1} \beta_j x^j + f_{\text{sp}}(x), \tag{2.25}$$

$$y_i = f(x_i) + \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \sigma_n^2). \tag{2.26}$$

This setup is different from the standard GP prior introduced in Chapter 1 for two reasons. First, all inputs are assumed to be between 0 and 1. Secondly, it is a combination of a parametric model with a polynomial feature mapping $\boldsymbol{\phi}(x) = [1, x, \ldots, x^{m-1}]$ and a separate GP prior which is effectively modelling the residuals

of the parametric component. In fact, it is an instance of an *additive* model (see Chapter 4). Due to the additive nature of the prior and the fact that we can represent the parametric component as a GP (as in Chapter 1), it is clear that the function prior implied by Equation 2.25 also gives rise to a GP. Its covariance function is given by the sum of the covariance functions of the two components, namely:

$$k_f(x, x') = k_{sp}(x, x') + \boldsymbol{\phi}(x)^\top \mathbf{B} \boldsymbol{\phi}(x). \tag{2.27}$$

Note that we haven't yet specified $k_{sp}(x, x')$. We would like to set it so that after observing a dataset of size of N our maximum-a-posteriori (MAP) function estimate is in line with the frequentist solution presented in Equation 2.22. For GPs the MAP estimate is given by the posterior mean function, which we derived in Equation 1.29:

$$\mu(x) = \mathbf{k}(x)^\top \left(\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N\right)^{-1} \mathbf{y}, \tag{2.28}$$

where $\mathbf{k}(x)$ is the $\mathbf{K}_{MN}$ matrix evaluated at the singleton test location, x, and $\mathbf{K}_N$ is the training covariance matrix associated with the kernel in Equation 2.27. After some rearrangement, we can rewrite Equation 2.28 in a form consistent with Equation 2.22

$$\mu(x) = \phi(x)^\top \hat{\boldsymbol{\beta}} + \mathbf{k}_{sp}^\top(x) \underbrace{\tilde{\mathbf{K}}_{\text{sp}}^{-1} \left(\mathbf{y} - \boldsymbol{\Phi}^\top \hat{\boldsymbol{\beta}}\right)}_{\equiv \hat{\boldsymbol{\alpha}}}, \tag{2.29}$$

where $\mathbf{k}_{\text{sp}}(x)$ is the cross-covariance vector associated with the kernel $k_{\text{sp}}(x, x')$, $\tilde{\mathbf{K}}_{\text{sp}} \equiv \mathbf{K}_{\text{sp}} + \sigma_n^2 \mathbf{I}_N$ and $\boldsymbol{\Phi}$ is the feature mapping applied to all training inputs. $\hat{\boldsymbol{\beta}}$ is the posterior mean of the parameters of the linear component of the model and is given by:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{B}^{-1} + \boldsymbol{\Phi} \tilde{\mathbf{K}}_{\text{sp}}^{-1} \boldsymbol{\Phi}^\top\right)^{-1} \boldsymbol{\Phi} \tilde{\mathbf{K}}_{\text{sp}}^{-1} \mathbf{y}. \tag{2.30}$$

Because the coefficients $\boldsymbol{\beta}$ are assumed to be arbitrary in the frequentist setup, we would like to consider the limit where $\mathbf{B}^{-1} \to \mathbf{0}$, thus obtaining the posterior over $\boldsymbol{\beta}$ given an improper prior:

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{\Phi} \tilde{\mathbf{K}}_{\text{sp}}^{-1} \boldsymbol{\Phi}^\top\right)^{-1} \boldsymbol{\Phi} \tilde{\mathbf{K}}_{\text{sp}}^{-1} \mathbf{y}. \tag{2.31}$$

Equation 2.29 makes intuitive sense since it states that the mean prediction at x is

the sum of the mean prediction of the linear component and the mean prediction of a GP which models the residuals $\mathbf{y} - \mathbf{\Phi}^\top \hat{\boldsymbol{\beta}}$.

Wahba [1990] showed that the mean function in Equation 2.29 is in line with the solution implied by the representer theorem if and only if we set $k_{\text{sp}}(\cdot, \cdot)$ as the covariance function of the $(m-1)$-fold-integrated Wiener process of amplitude σ_f^2 [1]. From a computational perspective this is great news because the $(m-1)$-fold-integrated Wiener process is the solution to the following SDE:

$$\frac{d^m f(x)}{dx^m} = w(x). \tag{2.32}$$

For example, a quintic spline prior gives rise to the following vector-Markov process:

$$\frac{d\mathbf{z}(x)}{dx} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{z}(x) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w(x), \tag{2.33}$$

where the variance of $w(x)$ is $q \equiv \sigma_f^2$. It is thus possible to compute the expressions in Equations 2.29 and 2.31 in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space, using the techniques described in the following section. Additionally, given our Bayesian approach, we can compute the uncertainty in our regression estimate, using the standard equations in Chapter 1, at close to no additional computational cost. The predictive variance at test location $x_\star$ can be written as follows:

$$\sigma^2(x_\star) = k_{\text{sp}}(x_\star, x_\star) - \mathbf{k}_{\text{sp}}^\top(x_\star) \tilde{\mathbf{K}}_{\text{sp}}^{-1} \mathbf{k}_{\text{sp}}(x_\star) + \boldsymbol{\rho}^\top \left(\mathbf{\Phi} \tilde{\mathbf{K}}_{\text{sp}}^{-1} \mathbf{\Phi}^\top \right) \boldsymbol{\rho}, \tag{2.34}$$

where

$$\boldsymbol{\rho} = \boldsymbol{\phi}(x_\star) - \mathbf{\Phi} \tilde{\mathbf{K}}_{\text{sp}}^{-1} \mathbf{k}_{\text{sp}}(x_\star). \tag{2.35}$$

[1] These are fundamental nonstationary stochastic processes.

2.3 GP Inference using State-Space Models

2.3.1 The State-Space Model

In the previous section we showed how many different GP priors can be written as a vector Markov process, with state dynamics described by the equation:

$$\frac{d\mathbf{z}(x)}{dx} = \mathbf{A}\mathbf{z}(x) + \mathbf{L}w(x). \tag{2.36}$$

Strictly, Equation 2.36 should be written in its integral form:

$$d\mathbf{z}(x) = \mathbf{A}\mathbf{z}(x)dx + \mathbf{L}\sqrt{q}d\beta(x), \tag{2.37}$$

where $d\beta(x) \equiv w(x)dx$ is the infinitesimal increment of standard Brownian motion $\beta(x)$ (also known as the Weiner process). For our purposes, Equation 2.37 is equivalent to 2.36 because we are only interested in integrating the SDE forward in time. Let's assume that $\mathbf{z}(x)$ is Gaussian distributed with $\mathbf{z}(x) \sim \mathcal{N}(\boldsymbol{\mu}(x), \boldsymbol{\Sigma}(x))$. Taking the expectation of both sides of Equation 2.37 we can derive the *ordinary* linear differential equation describing the evolution of the mean:

$$\begin{aligned} d\boldsymbol{\mu}(x) &= \mathbf{A}\boldsymbol{\mu}(x)dx, \\ \Rightarrow \frac{d\boldsymbol{\mu}(x)}{dx} &= \mathbf{A}\boldsymbol{\mu}(x), \end{aligned} \tag{2.38}$$

where we have used the fact that $\mathbb{E}(d\beta(x)) = 0$. Similarly, we can derive the evolution of the covariance matrix by considering the covariance of both sides of Equation 2.37.

$$\begin{aligned} d\boldsymbol{\Sigma}(x) &= \left(\mathbf{A}\boldsymbol{\Sigma}(x) + (\mathbf{A}\boldsymbol{\Sigma}(x))^\top\right) dx + q\mathbf{L}\mathbf{L}^\top dx \\ \Rightarrow \frac{d\boldsymbol{\Sigma}(x)}{dx} &= \mathbf{A}\boldsymbol{\Sigma}(x) + \boldsymbol{\Sigma}(x)\mathbf{A}^\top + q\mathbf{L}\mathbf{L}^\top \end{aligned} \tag{2.39}$$

where we have used $\mathrm{Cov}(d\beta(x)) = \mathbf{I}_m dx$ and the fact that the Brownian motion is independent of $\mathbf{z}(x)$. This matrix differential equation is commonly referred to as the *matrix Riccati* differential equation (see, e.g., Willems [1971]). We can now use

Equations 2.38 and 2.39 to derive analytic expressions for $\boldsymbol{\mu}(x)$ and $\boldsymbol{\Sigma}(x)$ given the initial conditions $\boldsymbol{\mu}(x_0) = \boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}(x_0) = \mathbf{0}$ (i.e., at x_0 the process is deterministic). Furthermore, using the standard results in Riley et al. [2006] and Arnold [1992] we have:

$$\boldsymbol{\mu}(x) = \exp(\mathbf{A}(x - x_0))\boldsymbol{\mu}_0, \tag{2.40}$$

where we have used a *matrix exponential*. The matrix exponential is the exponential function generalized to square matrix arguments. Formally it is defined as follows:

$$\exp(\mathbf{X}) = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{X}^k, \tag{2.41}$$

where $\mathbf{X}^k$ is the matrix $\mathbf{X}$ matrix-multiplied with itself k times. For the covariance, we have the following unique and non-negative definite solution:

$$\boldsymbol{\Sigma}(x) = q \int_{x_0}^{x} \exp(-\mathbf{A}(x - s))\mathbf{L}\mathbf{L}^\top \exp(-\mathbf{A}(x - s))ds, \tag{2.42}$$

where we have used the identity $(\exp(\mathbf{X}))^{-1} = \exp(-\mathbf{X})$. The integral in this Equation is over a matrix and is defined as the element-wise integral of its argument. Given the deterministic starting condition (which we have limited ourselves to) the density of $\mathbf{z}(x)$ is Gaussian for all x, and therefore $\boldsymbol{\mu}(x)$ and $\boldsymbol{\Sigma}(x)$ capture all the uncertainty about $\mathbf{z}(x)$. See Arnold [1992] for a proof of this.

Thus, all the conditional distributions in Figure 2.1 are Gaussians for the GP regression task and are listed below:

$$\textbf{Initial state :} \quad p(\mathbf{z}(x_1)) = \mathcal{N}(\mathbf{z}(x_1); \boldsymbol{\mu}, \mathbf{V}). \tag{2.43}$$

$$\textbf{State update :} \quad p(\mathbf{z}(x_i)|\mathbf{z}(x_{i-1})) = \mathcal{N}(\mathbf{z}(x_i); \boldsymbol{\Phi}_{i-1}\mathbf{z}(x_{i-1}), \mathbf{Q}_{i-1}). \tag{2.44}$$

$$\textbf{Emission :} \quad p(y(x_i)|\mathbf{z}(x_i)) = \mathcal{N}(y(x_i); \mathbf{H}\mathbf{z}(x_i), \sigma_n^2). \tag{2.45}$$

where we assume that the inputs x_i are sorted in ascending order and may include test inputs, in which case we simply do not consider the emission distribution at that location. Using the above, we can write:

$$\mathbf{\Phi}_{i-1} = \exp(\mathbf{A}\delta_i), \tag{2.46}$$

$$\mathbf{Q}_{i-1} = q \int_0^{\delta_i} \mathbf{c}_{i-1}(\delta_i - h)\mathbf{c}_{i-1}^{\top}(\delta_i - h)dh, \tag{2.47}$$

where $\delta_i \equiv x_i - x_{i-1}$ and $\mathbf{c}_{i-1}$ is the last column of the matrix $\exp(\mathbf{A}(\delta_i - h))$. The form of the emission distribution is very simple. $\mathbf{H}$ simply picks out the first element of the vector $\mathbf{z}(x_i)$ which corresponds to the latent function value inferred at location x_i.

In the case of the Matérn family and spline kernels, where we have obtained an analytic expression for $\mathbf{A}$, it is also possible to derive an analytic expression for $\exp(\mathbf{A}t)$ using the following identity:

$$\exp(\mathbf{A}t) = \mathcal{L}^{-1}\left\{(sI_M - \mathbf{A})^{-1}\right\}, \tag{2.48}$$

where $\mathcal{L}^{-1}$ denotes the *inverse Laplace transform* and $s \in \mathbb{C}$ is a complex number. Equation 2.48 can be derived straightforwardly using properties of the Laplace transform. As a result, we can compute both the expressions in Equation 2.46 and 2.47 in closed form. For example, for the Matérn(3/2) kernel we obtain:

$$\mathbf{\Phi}_{i-1} = \frac{1}{\exp(\lambda\delta_i)} \begin{bmatrix} (\lambda\delta_i + 1) & \delta_i \\ -(\lambda^2\delta_i) & (1 - \lambda\delta_i) \end{bmatrix}. \tag{2.49}$$

Note that closed form expressions can be obtained for any Matérn order using a symbolic math package such as MATLAB's Symbolic Math Toolbox (see Appendix B for code to generate these expressions automatically). Furthermore, when analytic expressions exist, the Symbolic Math Toolbox can also be used to to compute the integrals involved in computing $\mathbf{Q}_{i-1}$ (see Equation 2.47). In fact, there is a neat trick called *matrix fraction decomposition* which obviates the use of integrals for computing covariance matrices, as shown in Särkkä [2006]. Let's return to the matrix Riccati differential equation in Equation 2.39. Assuming $\mathbf{\Sigma}(x)$ can be written as:

$$\mathbf{\Sigma}(x) = \mathbf{C}(x)\mathbf{D}(x)^{-1}, \tag{2.50}$$

then by simply plugging in Equation 2.50 into both sides of the differential equation in 2.39, it can be seen that:

$$\frac{d\mathbf{C}(x)}{dx} = \mathbf{A}\mathbf{C}(x) + q\mathbf{L}\mathbf{L}^\top\mathbf{D}(x), \tag{2.51}$$

$$\frac{d\mathbf{D}(x)}{dx} = -\mathbf{A}^\top\mathbf{D}(x). \tag{2.52}$$

Amazingly, these system of equations can be expressed as one larger *linear* ODE, which in turn can be solved using matrix exponentials *only*:

$$\begin{bmatrix} \frac{d\mathbf{C}(x)}{dx} \\ \frac{d\mathbf{D}(x)}{dx} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & q\mathbf{L}\mathbf{L}^\top \\ \mathbf{0} & -\mathbf{A}^\top \end{bmatrix} \begin{bmatrix} \mathbf{C}(x) \\ \mathbf{D}(x) \end{bmatrix}. \tag{2.53}$$

For a spline kernel of order m we can derive analytic expression for both the $\mathbf{\Phi}_{i-1}$ and the $\mathbf{Q}_{i-1}$. Using Equations 2.46 and 2.47 we obtain:

$$\mathbf{\Phi}_{i-1} = \begin{bmatrix} 1 & \delta_i & \frac{\delta_i^2}{2!} & \dots & \frac{\delta_i^{m-1}}{(m-1)!} \\ 0 & 1 & \delta_i & \dots & \frac{\delta_i^{m-2}}{(m-2)!} \\ \dots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}, \tag{2.54}$$

$$\mathbf{Q}_{i-1}(j,k) = q\int_0^{\delta_i} \frac{(\delta_i - h)^{m-j}}{(m-j)!}\frac{(\delta_i - h)^{m-k}}{(m-k)!}dh \tag{2.55}$$

$$= \frac{q\delta_i^{(m-j)+(m-k)+1}}{((m-j)+(m-k)+1)(m-j)!(m-k)!}. \tag{2.56}$$

These expressions are consistent with those derived in Wecker and Ansley [1983], who do not use the SDE representation of splines explicitly.

If we do not have an analytic expression for $\mathbf{A}$, as was the case for the Taylor-expansion based technique for approximating the squared-exponential kernel, then it is necessary to compute $\mathbf{\Phi}_{i-1}$ and $\mathbf{Q}_{i-1}$ numerically. As the coefficients in $\mathbf{A}$ are also found numerically (by solving polynomial equations), there is a tendency for numerical errors to propagate (causing, for example, matrices to become non-positive definite). Indeed, we find that, in practice, this is a major disadvantage of using the Taylor-expansion based approximation, and will therefore stick to using high-order Matérn kernels for this task. In addition, one should be be wary of the

fact that computing derivatives of hyperparameters analytically becomes impossible if $\mathbf{\Phi}_{i-1}$ and $\mathbf{Q}_{i-1}$ are computed numerically.

For SDEs described by LTI systems, $\boldsymbol{\mu} = \mathbf{0}$, because the starting state of the SDE (at $x = -\infty$) is assumed to be $\mathbf{0}$ and because it is generally the case that $\lim_{t\to\infty} \exp(\mathbf{A}t) = \mathbf{0}$, a property which is easily verified for Equation 2.49. More generally, the *stability* of a linear system implies non-positivity of the eigenvalues of $\mathbf{A}$, so for any stable linear system it must be that $\boldsymbol{\mu} = \mathbf{0}$. We can compute $\mathbf{V}$ by considering what the process covariance tends to after running for an infinite interval, i.e.,

$$\mathbf{V} = q \int_0^\infty \mathbf{c}(h)\mathbf{c}^\top(h)dh, \tag{2.57}$$

where $\mathbf{c}$ is the last column of the matrix $\exp(\mathbf{A}h)$. The integral in Equation 2.57 always converges for LTI systems with absolutely integrable impulse response, i.e., for which

$$\int_{-\infty}^{\infty} |h(\tau)|d\tau < \infty. \tag{2.58}$$

It can be shown that the Matérn family and approximations to the squared-exponential covariance all give rise to LTI systems which satisfy this property. Alternatively, it is also possible to obtain $\mathbf{V}$ by solving the matrix Riccati equation, as in Bar-Shalom et al. [2001]:

$$\frac{d\mathbf{V}}{dx} = \mathbf{A}\mathbf{V} + \mathbf{V}\mathbf{A}^\top + \mathbf{L}q\mathbf{L}^\top = \mathbf{0}. \tag{2.59}$$

Recall that for the spline kernel we limited the evolution of the SDE between input locations 0 and 1. To remain consistent with the literature, it suffices to set $\boldsymbol{\mu} = \mathbf{0}$ and

$$\mathbf{V} = q \int_0^{x_1} \mathbf{c}(h)\mathbf{c}^\top(h)dh. \tag{2.60}$$

Intuitively, we are starting the integrated Wiener process at the origin at $x = 0$ and considering its dynamics until $x = 1$.

2.3.2 Inference and Learning [▷▷]

Now that we have specified how to compute all the necessary conditionals forming the state-space model, GP regression becomes a special case of performing inference on the chain-structured graph in Figure 2.1. After sorting the inputs, this

can be done in $\mathcal{O}(N)$ runtime and space using the *filtering-smoothing* algorithm, which corresponds to running the generic belief propagation algorithm on our chain-structured graph. We would like to compute $p(\mathbf{z}_k|\mathbf{y}, \mathbf{x}, \theta)$ for $k = 1 \dots K$, where $K = N + M$ as we have jointly sorted our training and input locations to form the vector $\mathbf{x}$. The inference procedure consists firstly of a "forward" *filtering* run where $p(\mathbf{z}_k|y_1, \dots, y_{i(k)}, \mathbf{x}, \theta)$ are computed from $k = 1, \dots, K$ in a recursive manner. $i(k)$ gives the index of the target corresponding to state k in the Markov chain. The recursion is commonly known as the *Kalman filter* Kalman [1960]. The key equations are given below:

$$
\begin{aligned}
p(\mathbf{z}_{k-1}|y_1, \dots, y_{i(k-1)}, \mathbf{x}, \theta) &= \mathcal{N}(\boldsymbol{\mu}_{k-1}^{(f)}, \mathbf{V}_{k-1}^{(f)}), \\
&\Downarrow \text{ "Time" update} \\
p(\mathbf{z}_k|y_1, \dots, y_{i(k-1)}, \mathbf{x}, \theta) &= \mathcal{N}(\boldsymbol{\Phi}_{k-1}\boldsymbol{\mu}_{k-1}^{(f)}, \underbrace{\boldsymbol{\Phi}_{k-1}\mathbf{V}_{k-1}^{(f)}\boldsymbol{\Phi}_{k-1}^\top + \mathbf{Q}_{k-1}}_{\equiv \mathbf{P}_{k-1}}), \\
&\Downarrow \text{ Measurement update} \\
p(\mathbf{z}_k|y_1, \dots, y_{i(k)}, \mathbf{x}, \theta) &= \mathcal{N}(\boldsymbol{\mu}_k^{(f)}, \mathbf{V}_k^{(f)}),
\end{aligned}
$$

where $\boldsymbol{\mu}_k^{(f)} = \boldsymbol{\Phi}_{k-1}\boldsymbol{\mu}_{k-1}$, and $\mathbf{V}_k^{(f)} = \mathbf{P}_{k-1}$ if $i(k) = i(k-1)$. In the case where $i(k) = i(k-1) + 1$, we have

$$
\boldsymbol{\mu}_k^{(f)} = \boldsymbol{\Phi}_{k-1}\boldsymbol{\mu}_{k-1}^{(f)} + \mathbf{G}_k[y_{i(k)} - \mathbf{H}\boldsymbol{\Phi}_{k-1}\boldsymbol{\mu}_{k-1}^{(f)}], \tag{2.61}
$$

$$
\mathbf{V}_k^{(f)} = \mathbf{P}_{k-1} - \mathbf{G}_k\mathbf{H}\mathbf{P}_{k-1}, \tag{2.62}
$$

where $\mathbf{G}_k$ is known as the *Kalman gain matrix*:

$$
\mathbf{G}_k = \mathbf{P}_{k-1}\mathbf{H}^\top \left(\mathbf{H}^\top\mathbf{P}_{k-1}\mathbf{H} + \sigma_n^2\right)^{-1}. \tag{2.63}
$$

The recursion is initialized with $\boldsymbol{\mu}_1^{(f)} = \boldsymbol{\mu}$ and $\mathbf{V}_1^{(f)} = \mathbf{V}$. Once the filtering densities are known, a backward recursive procedure, commonly known as the RTS smoother, updates $p(\mathbf{z}_k|y_1, \dots, y_{i(k)}, \mathbf{x}, \theta)$ to $p(\mathbf{z}_k|\mathbf{y}, \mathbf{x}, \theta) \equiv \mathcal{N}(\boldsymbol{\mu}_k, \mathbf{V}_k)$ for $k = K, \dots, 1$, as follows:

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_k^{(f)} + \mathbf{L}_k \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\Phi}_k \boldsymbol{\mu}_k^{(f)} \right), \tag{2.64}$$

$$\mathbf{V}_k = \mathbf{V}_k^{(f)} + \mathbf{L}_k \left(\mathbf{V}_{k+1} - \mathbf{P}_k \right) \mathbf{L}_k^\top, \tag{2.65}$$

where $\mathbf{L}_k \equiv \mathbf{V}_k \boldsymbol{\Phi}_k^\top \mathbf{P}_k^{-1}$ is known as the *Lyapunov matrix*. The recursion is initialized using $\boldsymbol{\mu}_K = \boldsymbol{\mu}_K^{(f)}$ and $\mathbf{V}_K = \mathbf{V}_K^{(f)}$. Note that the predictive density $p(f(x_k)|\mathbf{y}, \mathbf{x}, \theta)$ has mean and variance given by $\mu_k = \boldsymbol{\mu}_k(1)$ and $\sigma_k^2 = \mathbf{V}_k(1,1)$. Algorithm 2 gives the pseudo-code for performing GP regression using state-space models. Note that this algorithm has been presented in the case where we know the locations of the test points in advance. This may not be the case in practice. Fortunately, due to the Markovian structure of the state-space model, we can predict at any new test input using the posterior distributions over the latent states at neighbouring training inputs (and there can be at most 2 neighbours).

Recall that in the standard GP regression algorithm, the hyperparameters were learned by maximizing the marginal likelihood $p(\mathbf{y}|\mathbf{x}, \theta)$. When we can represent the GP model in state-space form, it is possible to use the Expectation-Maximization (EM) algorithm Dempster et al. [1977] to maximize the marginal likelihood with respect to the hyperparameters. The purpose of the EM algorithm is to avoid the direct maximization of the marginal likelihood, since finding analytic expressions for this quantity and its derivatives is often too difficult (or, indeed, not possible). Instead, one iteratively alternates between computing the *expected complete data log-likelihood* ($\mathbb{E}_C$) (E-step) and optimising it w.r.t. the hyperparameters (M-step). Both these steps are often analytically tractable, which explains the wide-spread use of the EM algorithm. $\mathbb{E}_C$ is given by:

$$\mathbb{E}_C(\theta) \equiv \mathbb{E}_{p(\mathbf{Z}|\mathbf{y},\mathbf{x},\theta^{\text{old}})} \left(p(\mathbf{Z}, \mathbf{y}|\mathbf{x}, \theta) \right), \tag{2.66}$$

where $\mathbf{Z} \equiv \{\mathbf{z}_k\}_{k=1}^K$. Referring to the graph in Figure 2.1 we can see that

$$p(\mathbf{Z}, \mathbf{y}|\mathbf{x}, \theta) = \log p(\mathbf{z}(x_1), \theta) + \sum_{t=2}^{K} \log p(\mathbf{z}(x_t)|\mathbf{z}(x_{t-1}), \theta) + \sum_{t=1}^{K} \log p(y(x_t)|\mathbf{z}(x_t)). \tag{2.67}$$

As we can push the expectation in Equation in 2.66 to the individual summands in

Algorithm 2: Gaussian Process Regression using SSMs

inputs : Jointly sorted training and test input locations $\mathbf{x}$. Targets $\mathbf{y}$ associated with training inputs. State transition function `stfunc` that returns $\mathbf{\Phi}$ *and* $\mathbf{Q}$ *matrices*. Hyperparameters θ

outputs: Log-marginal likelihood $\log Z(\theta)$. Predictive means $\boldsymbol{\mu}_\star$ and variances $\mathbf{v}_\star$. E-step moments: $\mathbb{E}(\mathbf{z}_t\mathbf{z}_t^\top)$, $\mathbb{E}(\mathbf{z}_t\mathbf{z}_{t-1}^\top)$

$\boldsymbol{\mu}_0^{(f)} \leftarrow \boldsymbol{\mu}$; $\mathbf{V}_0^{(f)} \leftarrow \mathbf{V}$; $Z(\theta) = 0$;
for $t \leftarrow 1$ **to** K **do**
  if $t > 1$ **then** $[\mathbf{\Phi}_{t-1}, \mathbf{Q}_{t-1}] \leftarrow$ `stfunc(`θ, $\mathbf{x}(t) - \mathbf{x}(t-1)$`)`;
  else $[\mathbf{\Phi}_{t-1}, \mathbf{Q}_{t-1}] \leftarrow$`stfunc(`$\theta$, ∞`);` `//Prior process covariance`
  $\mathbf{P}_{t-1} \leftarrow \mathbf{\Phi}_{t-1}\mathbf{V}_{t-1}^{(f)}\mathbf{\Phi}_{t-1}^\top + \mathbf{Q}_{t-1}$;
  if `is_train(`t`)` **then**
    $\log Z(\theta) \leftarrow \log Z(\theta) +$ `gausslik(`$\mathbf{y}(i(t))$, $\mathbf{H}\mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)}$, $\mathbf{H}\mathbf{P}_{t-1}\mathbf{H}^\top + \sigma_n^2$`)`;
    $\boldsymbol{\mu}_t^{(f)} = \mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)} + \mathbf{G}_t[\mathbf{y}(i(t)) - \mathbf{H}\mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)}]$;
    $\mathbf{V}_t^{(f)} = \mathbf{P}_{t-1} - \mathbf{G}_t\mathbf{H}\mathbf{P}_{t-1}$;
  else
    $\boldsymbol{\mu}_t^{(f)} \leftarrow \mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)}$; $\mathbf{V}_t^{(f)} \leftarrow \mathbf{P}_{t-1}$;
  end
end
$\boldsymbol{\mu}_K \leftarrow \boldsymbol{\mu}_K^{(f)}$; $\mathbf{V}_K \leftarrow \mathbf{V}_K^{(f)}$; $\boldsymbol{\mu}_\star(K) \leftarrow \mathbf{H}\boldsymbol{\mu}_K$; $\mathbf{v}_\star(K) \leftarrow \mathbf{H}\mathbf{V}_K\mathbf{H}^\top$;
$\mathbb{E}(\mathbf{z}_K\mathbf{z}_K^\top) \leftarrow \mathbf{V}_K + \boldsymbol{\mu}_K\boldsymbol{\mu}_K^\top$;
$\mathbb{E}(\mathbf{z}_K\mathbf{z}_{K-1}^\top) \leftarrow (I_D - \mathbf{G}_K\mathbf{H})\mathbf{\Phi}_{K-1}\mathbf{V}_{K-1}$;
for $t \leftarrow K-1$ **to** 1 **do**
  $\mathbf{L}_t \leftarrow \mathbf{V}_t\mathbf{\Phi}_t^\top\mathbf{P}_t^{-1}$;
  $\boldsymbol{\mu}_t \leftarrow \boldsymbol{\mu}_t^{(f)} + \mathbf{L}_t\left(\boldsymbol{\mu}_{t+1} - \mathbf{\Phi}_t\boldsymbol{\mu}_t^{(f)}\right)$; $\boldsymbol{\mu}_\star(t) \leftarrow \mathbf{H}\boldsymbol{\mu}_t$;
  $\mathbf{V}_t \leftarrow \mathbf{V}_t^{(f)} + \mathbf{L}_t\left(\mathbf{V}_{t+1} - \mathbf{P}_t\right)\mathbf{L}_t^\top$; $\mathbf{v}_\star(t) \leftarrow \mathbf{H}\mathbf{V}_t\mathbf{H}^\top$;
  $\mathbb{E}(\mathbf{z}_t\mathbf{z}_t^\top) \leftarrow \mathbf{V}_t + \boldsymbol{\mu}_t\boldsymbol{\mu}_t^\top$;
  if $t < K-1$ **then**
    $\mathbb{E}(\mathbf{z}_t\mathbf{z}_{t-1}^\top) \leftarrow \mathbf{V}_{t+1}^{(f)}\mathbf{L}_t^\top + \mathbf{L}_{t+1}\left(\mathbb{E}(\mathbf{z}_{t+1}\mathbf{z}_t^\top) - \mathbf{\Phi}_{t+1}\mathbf{V}_{t+1}\right)\mathbf{L}_t^\top$;
  end
end

Equation 2.67, the expectations computed in Algorithm 2 are sufficient to compute $\mathbb{E}_C$, and constitute the E-step of the EM algorithm. In the M-step we maximize Equation 2.66 with respect to θ, and repeat the process until convergence, as shown in Algorithm 3. The minimization can be accomplished using any off-the-shelf function minimizer, such as the one found in Rasmussen and Nickisch [2010].

Algorithm 3: Learning the hyperparameters using EM

inputs : Training data $\{\mathbf{x}, \mathbf{y}\}$. Initial hypers θ_{init}.
outputs: Learned hyperparameters θ_{opt}.

$\theta_{\text{old}} \leftarrow \theta_{\text{init}}$;
while $(Z(\theta_{old}) - Z(\theta_{opt})) > \epsilon$ **do**
    Compute E-step moments using Algorithm 2 for θ_{old}; `//E-step`
    Use E-step moments to evaluate and minimize $-\mathbb{E}_C(\theta)$ w.r.t θ; `//M-step`
    $\theta_{\text{old}} \leftarrow \theta_{\text{opt}}$;
end

2.4 Generalised Gauss-Markov Process Regression

So far, we have focussed on the case where the state-space representation of the GP prior has a Gaussian emission density (Equation 2.45). For many practical applications this assumption would not be appropriate, be it because the targets are categorical (as in classification) or represent counts (as in Poisson regression) or simply because Gaussian errors cannot handle outliers (in which case a Student-t error model would be a better choice). All of these cases can be dealt with by replacing the Gaussian emission density with a suitable likelihood function for $p(y(x_i)|\mathbf{z}(x_i))$. For example, for 2-class classification where $y(x_i) \in \{-1, 1\}$ one can use a *probit* likelihood:

$$p(y(x_i)|\mathbf{z}(x_i)) = \Phi((\mathbf{H}\mathbf{z}(x_i))y_i), \tag{2.68}$$

where $\Phi(\cdot)$ is the standard Gaussian CDF. Inference and learning with non-Gaussian emissions necessitates the use of approximations to the true posterior over the latent-state Markov chain in Figure 2.1. There exist many algorithms for performing such an approximation, including variational inference and sequential Monte Carlo methods (for a thorough introduction, see Doucet et al. [2001]). However, we will

focus on Expectation propagation (EP), due to its superior approximation qualities (see e.g. Nickisch and Rasmussen [2008]) for GPs with non-Gaussian likelihoods. Moreover, it is shown in Minka [2001] that EP performs well for tree-structured graphs (it basically boils down to a "projected" Gaussian belief propagation) and we have spent most of this Chapter converting the GP regression graph into a chain structure!

2.4.1 Expectation Propagation [▷▷]

The EP algorithm falls under the general framework of approximating an analytically and/or computationally intractable posterior distribution $p(\mathbf{Z}|\mathcal{D},\theta)$ with a "simpler" distribution $q(\mathbf{Z}|\mathcal{D},\theta)$, where $\mathbf{Z}$ represents all latent variables, $\mathcal{D}$ is the observed data and θ is a set of fixed hyperparameters. q is usually simpler than p in two ways. First, it is deemed to belong to the exponential family (for analytic convenience), and second, it is assumed to have a simpler factorization than p (for computational reasons). In this section, we will first present the EP algorithm for an arbitrary factor graph representing p, and assume that q is a *fully-factorized Gaussian* distribution over $\mathbf{Z}$. We will illustrate the ideas using the application of EP to the graph in Figure 2.1, which is presented as a factor graph in Figure 2.5.

Generally, the exact posterior will be proportional to a product of factors:

$$p(\mathbf{Z}|\mathcal{D},\theta) \propto \prod_i \phi_i(\mathbf{Z}_{\mathrm{ne}(\phi_i)}), \tag{2.69}$$

where $\mathbf{Z}_{\mathrm{ne}(\phi_i)}$ is the subset of latent variables which are neighbours of factor ϕ_i. Our aim is to approximate this posterior using a product of Gaussian factors, each of which is a neighbour to only a single latent variable $\mathbf{z}_k$, i.e.:

$$q(\mathbf{Z}|\mathcal{D},\theta) \propto \prod_i \prod_{k\in\mathrm{ne}(\phi_i)} \tilde{\phi}_{ik}(\mathbf{z}_k). \tag{2.70}$$

Ideally we would like to be able minimize

$$\mathrm{KL}\left(\frac{1}{p(\mathcal{D}|\theta)}\prod_i \phi_i(\mathbf{Z}_{\mathrm{ne}(\phi_i)}) \;\middle\|\; \frac{1}{Z_{\mathrm{approx}}}\prod_i \prod_{k\in\mathrm{ne}(\phi_i)} \tilde{\phi}_{ik}(\mathbf{z}_k)\right), \tag{2.71}$$

by matching the first and second moments of the marginals over the $\mathbf{z}_k$ of the right-hand-side to those of the left-hand-side. The approximation to the evidence $p(\mathcal{D}|\theta)$, given by Z_{approx}, can then be computed by normalising the unnormalised product of Gaussians on the right-hand-side. However, this ideal KL minimisation is intractable because it requires performing marginalization over the true posterior. In EP, we instead iterate across the individual factors ϕ_i according to a pre-determined schedule [1] and minimize, at each iteration, the following:

$$\text{KL}\left(\frac{\left[\prod_{j\neq i}\prod_{k\in\text{ne}(\phi_j)}\tilde{\phi}_{jk}^{\text{old}}(\mathbf{z}_k)\right]\phi_i(\mathbf{Z}_{\text{ne}(\phi_i)})}{Z_i} \;\Big\|\; \frac{1}{Z_{\text{approx}}}\prod_i\prod_{k\in\text{ne}(\phi_i)}\tilde{\phi}_{ik}^{\text{new}}(\mathbf{z}_k)\right), \tag{2.72}$$

where ne$(\cdot)$ returns the variable neighbours of one or more factor variables, and vice versa. In many applications of EP, minimizing the divergence in Equation 2.72 via the matching of first and second moments is a tractable operation. Furthermore, marginals over the variables that are not in the neighbourhood of ϕ_i are unchanged. For variables which *are* in the neighbourhood of ϕ_i we have the following update equation:

$$\forall k \in \text{ne}(\phi_i) \tag{2.73}$$

$$\tilde{\phi}_{ik}^{\text{new}}(\mathbf{z}_k) \propto \frac{\text{proj}\left[\prod_{\substack{j\in\text{ne}(\mathbf{z}_k)\\ j\neq i}}\tilde{\phi}_{jk}^{\text{old}}(\mathbf{z}_k)\int_{\mathbf{Z}_{\neg k}}\phi_i(\mathbf{Z}_{\text{ne}(\phi_i)})\prod_{\mathbf{z}_m\in\mathbf{Z}_{\neg k}}\prod_{\ell\in\text{ne}(\mathbf{z}_m)}\tilde{\phi}_{\ell m}^{\text{old}}(\mathbf{z}_m)\,d\mathbf{Z}_{\neg k}\right]}{\prod_{\substack{j\in\text{ne}(\mathbf{z}_k)\\ j\neq i}}\tilde{\phi}_{jk}^{\text{old}}(\mathbf{z}_k)}.$$

where we have defined $\mathbf{Z}_{\neg k}$ to be the set of variables connected to factor ϕ_i, excluding $\mathbf{z}_k$. This update is illustrated in Figure 2.4.

The proj $[\cdot]$ operator takes a non-Gaussian argument and *projects* it to a Gaussian so that first and second moments of the Gaussian match those of the argument. If the argument turns out to be Gaussian then proj $[\cdot]$ is the identity. The proj $[\cdot]$ operator also computes the zeroth moment of its argument (since it is usually unnormalized)

[1] In practice, the performance of EP is, unsurprisingly, sensitive to the choice of schedule, as discussed in Kuss and Rasmussen [2005].

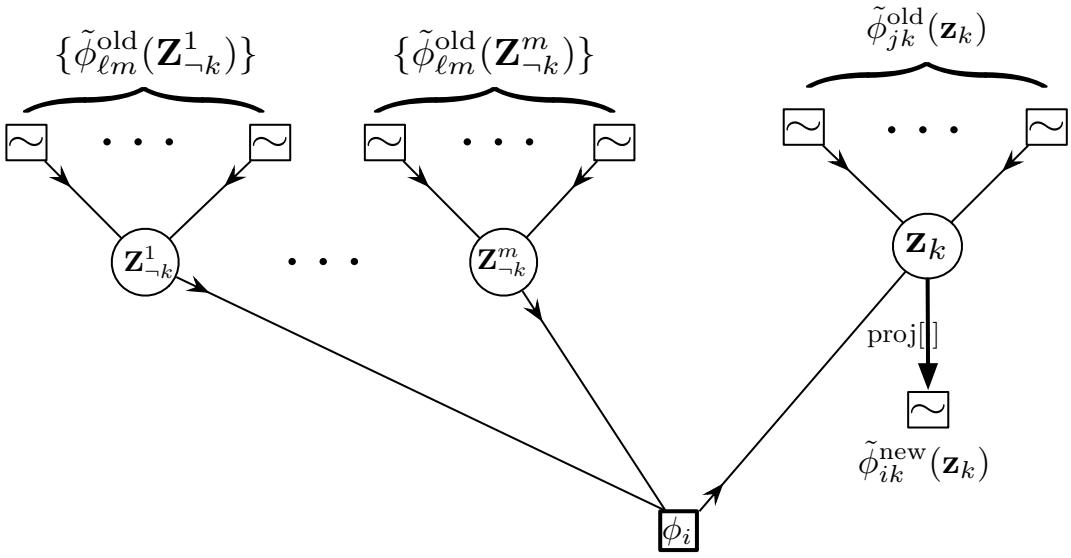


Figure 2.4: Graphical illustration of an individual EP update. The arrows conceptually represent the flow of information during the update. The boxes with the $\sim$ sign represent an approximate factor: $\tilde{\phi}^{\text{old}}$. Recall that, as in standard belief propagation, variable nodes multiply the messages sent to them by neighbouring factors, and factor nodes "multiply-and-marginalise" the messages sent to them by neighbouring variables.

as this quantity is necessary to evaluate the new normalisation constant of the updated factor $\tilde{\phi}^{\text{new}}_{ik}(\mathbf{z}_k)$. The normalisation of each updated factor is then used to compute Z_{approx}. Looking more closely at Equation 2.73, we can see that every step of EP involves multiplication and division of (multivariate) Gaussian distributions. Equations for implementing these operations are given in Appendix A. Notice that if the argument to proj $[\cdot]$ is a Gaussian then Equation 2.73 simplifies to:

$$\phi^{\text{new}}_{ik}(\mathbf{z}_k) = \int_{\mathbf{Z}_{\neg k}} \phi_i(\mathbf{Z}_{\text{ne}(\phi_i)}) \prod_{\mathbf{z}_m \in \mathbf{Z}_{\neg k}} \prod_{\ell \in \text{ne}(\mathbf{z}_m)} \tilde{\phi}^{\text{old}}_{\ell m}(\mathbf{z}_m) \; d\mathbf{Z}_{\neg k}, \tag{2.74}$$

which is precisely the expression for the message factor ϕ_i would send to variable $\mathbf{z}_k$ in (loopy) belief propagation (with the messages sent by neighbouring variables to ϕ_i folded in). Indeed, it is a well-known result that when the approximating distribution is fully-factorized and projections are not needed then EP reduces to loopy belief propagation, see Minka [2001] and Bishop [2007]. In practice, when projections are involved and/or the graph is not a tree, it is necessary to run multiple passes over

all the ϕ_i to achieve improved accuracy. Usually, one runs EP until the change in the $\tilde{\phi}$ is below some threshold (or indeed, the change in Z_{approx} is negligible).

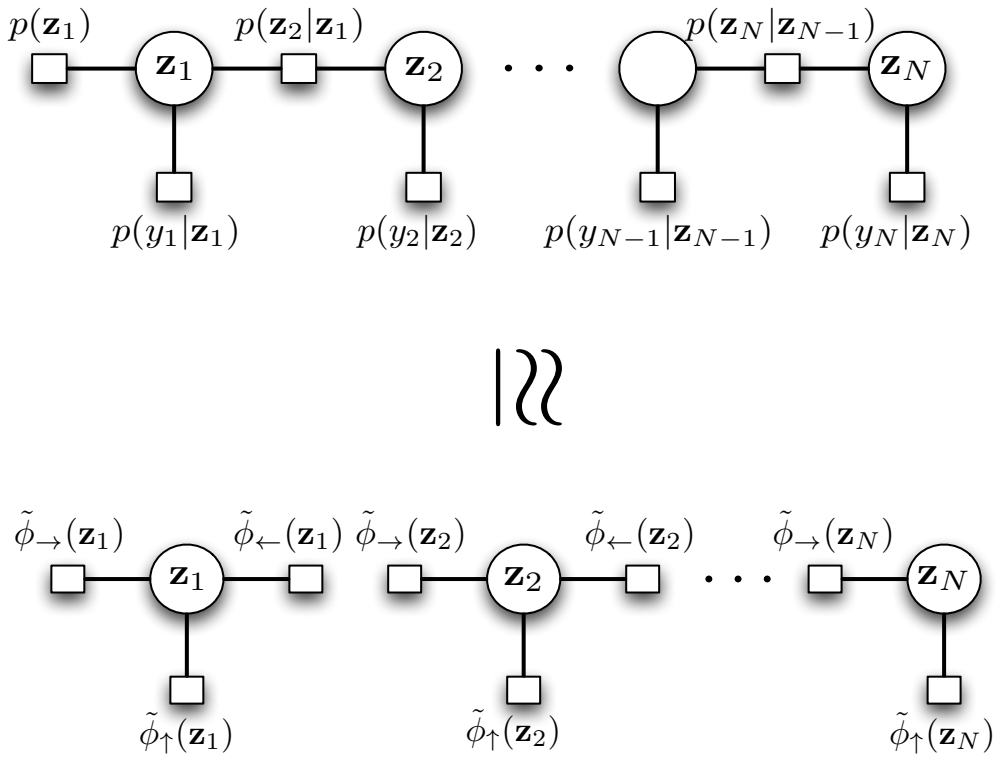


Figure 2.5: (Top) True factor graph for the SSM. Likelihood factors are non-Gaussian. (Bottom) Fully-factorised approximate factor graph used by EP. Likelihood factors *are* Gaussian.

Applying EP to the State-Space Model

We now derive the EP update equations for the graph in Figure 2.1. The approximation process is illustrated in Figure 2.5. In this example the data $\mathcal{D}$ consists of the target vector $\mathbf{y}$. As $p(\mathbf{Z}|\mathbf{y},\theta) \propto p(\mathbf{Z},\mathbf{y}|\theta)$ we can write:

$$p(\mathbf{Z}|\mathbf{y},\theta) \propto p(\mathbf{z}_1)p(y_1|\mathbf{z}_1)\prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1})p(y_n|\mathbf{z}_n). \tag{2.75}$$

We would like to approximate 2.75 with a fully-factorized Gaussian as in Equation 2.70. Using the general relation given in Equation 2.73, it is possible to show that

the update equations are given by:

$$\tilde{\phi}_{\rightarrow}(\mathbf{z}_1) = p(\mathbf{z}_1), \tag{2.76}$$

$$\begin{aligned}\tilde{\phi}_{\rightarrow}(\mathbf{z}_n) &= \frac{\text{proj}\left[\tilde{\phi}_{\uparrow}(\mathbf{z}_n)\tilde{\phi}_{\leftarrow}(\mathbf{z}_n)\int p(\mathbf{z}_n|\mathbf{z}_{n-1})\tilde{\phi}_{\rightarrow}(\mathbf{z}_{n-1})\tilde{\phi}_{\uparrow}(\mathbf{z}_{n-1})d\mathbf{z}_{n-1}\right]}{\tilde{\phi}_{\uparrow}(\mathbf{z}_n)\tilde{\phi}_{\leftarrow}(\mathbf{z}_n)} \\ &= \int p(\mathbf{z}_n|\mathbf{z}_{n-1})\tilde{\phi}_{\rightarrow}(\mathbf{z}_{n-1})\tilde{\phi}_{\uparrow}(\mathbf{z}_{n-1})d\mathbf{z}_{n-1}, \quad n = 2, \ldots, N. \end{aligned} \tag{2.77}$$

$$\tilde{\phi}_{\leftarrow}(\mathbf{z}_N) = 1, \tag{2.78}$$

$$\begin{aligned}\tilde{\phi}_{\leftarrow}(\mathbf{z}_n) &= \frac{\text{proj}\left[\tilde{\phi}_{\rightarrow}(\mathbf{z}_n)\tilde{\phi}_{\uparrow}(\mathbf{z}_n)\int p(\mathbf{z}_{n+1}|\mathbf{z}_n)\tilde{\phi}_{\uparrow}(\mathbf{z}_{n+1})\tilde{\phi}_{\leftarrow}(\mathbf{z}_{n+1})d\mathbf{z}_{n+1}\right]}{\tilde{\phi}_{\rightarrow}(\mathbf{z}_n)\tilde{\phi}_{\uparrow}(\mathbf{z}_n)} \\ &= \int p(\mathbf{z}_{n+1}|\mathbf{z}_n)\tilde{\phi}_{\uparrow}(\mathbf{z}_{n+1})\tilde{\phi}_{\leftarrow}(\mathbf{z}_{n+1})d\mathbf{z}_{n+1}, \quad n = 1, \ldots, N-1. \end{aligned} \tag{2.79}$$

$$\tilde{\phi}_{\uparrow}(\mathbf{z}_n) = \frac{\text{proj}\left[\tilde{\phi}_{\rightarrow}(\mathbf{z}_n)p(y_n|\mathbf{z}_n)\tilde{\phi}_{\leftarrow}(\mathbf{z}_n)\right]}{\tilde{\phi}_{\rightarrow}(\mathbf{z}_n)\tilde{\phi}_{\leftarrow}(\mathbf{z}_n)}, \quad n = 1, \ldots, N. \tag{2.80}$$

Note that in these update equations we have used the notation given in Figure 2.5.

2.4.2 Inference and Learning

In Equations 2.77 and 2.79, we have used the fact that the state transition function is still a Gaussian on the previous state, and that therefore the whole argument to the proj[·] operator is a Gaussian. As a consequence, these updates are consistent with standard Kalman filtering and smoothing, although, strictly, the Kalman recursions are defined over $\tilde{\phi}_{\rightarrow}(\mathbf{z}_n)\tilde{\phi}_{\uparrow}(\mathbf{z}_n)$ for filtering and over $\tilde{\phi}_{\rightarrow}(\mathbf{z}_n)\tilde{\phi}_{\uparrow}(\mathbf{z}_n)\tilde{\phi}_{\leftarrow}(\mathbf{z}_n)$ during smoothing, so that when the run is complete we are left with the marginals over the $\mathbf{z}_n$ (see Algorithm 2). The difference between the regression and generalized

regression settings presents itself only in Equation 2.80. For the fully-Gaussian SSM we have $\tilde{\phi}_{\uparrow}(\mathbf{z}_n) = p(y_n|\mathbf{z}_n)$, which, due to the nature of the emission density, is a message which only affects the first element of $\mathbf{z}_n$. This property also applies to Equation 2.73, so, in practice, we only need to do Gaussian multiplications, divisions and projection over the first component of $\mathbf{z}_n$. The resulting first and second moments (which are scalar) can be treated as *continuous pseudo-targets* and *Gaussian pseudo-noise variables* that can be plugged straight into routine presented in Algorithm 2! This allows the practitioner to extend standard Kalman filtering and smoothing code to handle, in principle, *any* likelihood function for which Equation 2.73 has been implemented, with only minor alterations to the core code. Example implementations for cases where the targets represent class labels are given in Appendix B. Of course, the approximate inference code will require more than one pass over all factors unlike the fully-Gaussian case where only one pass is required to compute the exact posterior. These ideas are illustrated in Algorithm 4. As mentioned, the performance of EP is sensitive to the schedule one uses to update messages. Algorithm 4 illustrates a schedule which works well in practice. In the first iteration the likelihood messages are updated using the only forward filtering messages as the context, i.e., none of the backward messages are updated until the end of the first run of filtering. In future iterations we alternate between performing full forward-backward message passing using fixed likelihood messages, and a pass where we update all likelihood messages given a fixed posterior over the latent chain.

Finally, notice that the expected sufficient statistics for the latent variables are computed in exactly the same way for generalized GP regression. Therefore, for the purposes of hyperparameter learning nothing needs to be changed – we simply use the output of Algorithm 4 as part of the EM routine given in Algorithm 3.

2.5 Results

In this section, we report the runtimes of Algorithm 4 for a sequence of synthetic binary classification datasets with length $N = [500, 1000, 2000, \ldots, 128000]$ and compare it to those of the standard GP classification routine, which also uses the EP approximation. The synthetic data is generated according to the following generative model:

Algorithm 4: Generalized Gaussian Processs Regression using SSMs

inputs : Jointly sorted training and test input locations $\mathbf{x}$. Targets $\mathbf{y}$ associated with training inputs. `stfunc` returns $\mathbf{\Phi}$ and $\mathbf{Q}$ matrices. `likfunc` implements Equation 2.73. Hypers θ.

outputs: Approximate log-marginal likelihood $\log Z(\theta)$. Predictive means $\boldsymbol{\mu}_\star$ and variances $\mathbf{v}_\star$. E-step moments: $\mathbb{E}(\mathbf{z}_t\mathbf{z}_t^\top)$, $\mathbb{E}(\mathbf{z}_t\mathbf{z}_{t-1}^\top)$.

$\boldsymbol{\mu}_0^{(f)} \leftarrow \boldsymbol{\mu}$; $\mathbf{V}_0^{(f)} \leftarrow \mathbf{V}$; $\mathbf{P}_0 \leftarrow \mathbf{V}$; $\log Z(\theta) = 0$; $j \leftarrow 1$;
while $\Delta \log Z(\theta) > \epsilon$ **do**
  for $t \leftarrow 1$ **to** K **do**
    if $t > 1$ **then** $[\mathbf{\Phi}_{t-1}, \mathbf{Q}_{t-1}] \leftarrow$ `stfunc`$(\theta, \mathbf{x}(t) - \mathbf{x}(t-1))$;
    else $[\mathbf{\Phi}_{t-1}, \mathbf{Q}_{t-1}] \leftarrow$`stfunc`$(\theta, \infty)$; `//Prior process covariance`
    $\mathbf{P}_{t-1} \leftarrow \mathbf{\Phi}_{t-1}\mathbf{V}_{t-1}^{(f)}\mathbf{\Phi}_{t-1}^\top + \mathbf{Q}_{t-1}$;
    if `is_train`(t) **then**
      $\mu_c \leftarrow \mathbf{H}\mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)}$; $v_c \leftarrow \mathbf{H}\mathbf{P}_{t-1}\mathbf{H}^\top$;
      if *first EP iter* **then**
        $[\tilde{y}(j), \tilde{\sigma_n^2}(j)] \leftarrow$`likfunc`$(\mathbf{y}(i(t)), \mu_c, v_c)$;
      end
      $\log Z(\theta) \leftarrow \log Z(\theta) +$ `gausslik`$(\tilde{y}(j), \mu_c, v_c + \tilde{\sigma_n^2}(j))$;
      $\mathbf{G}_t \leftarrow \mathbf{P}_{t-1}\mathbf{H}^\top / (v_c + \tilde{\sigma_n^2}(j))$;
      $\boldsymbol{\mu}_t^{(f)} = \mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)} + \mathbf{G}_t[\tilde{y}(j) - \mu_c]$;
      $\mathbf{V}_t^{(f)} = \mathbf{P}_{t-1} - \mathbf{G}_t\mathbf{H}\mathbf{P}_{t-1}$; $j \leftarrow j + 1$;
    else
      $\boldsymbol{\mu}_t^{(f)} \leftarrow \mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)}$; $\mathbf{V}_t^{(f)} \leftarrow \mathbf{P}_{t-1}$;
    end
  end
  `//Smoothing is as presented in Algorithm 2`
  `//Update likelihood messages`
  $j \leftarrow 1$;
  for $t \leftarrow 1$ **to** K **do**
    if `is_train`(t) **then**
      $[\tilde{y}(j), \tilde{\sigma_n^2}(j)] \leftarrow$`likfunc`$(\mathbf{y}(i(t)), \boldsymbol{\mu}_\star(i(t)), \mathbf{v}_\star(i(t)))$;
      $j \leftarrow j + 1$;
    end
  end
end

$$
\begin{aligned}
y_i &\sim p_i \qquad i = 1, \ldots, N, \\
p_i &= \Phi\left(\mathbf{f}(X_i)\right), \\
\mathbf{f}(\cdot) &\sim \mathcal{GP}\left(0; k(x, x')\right),
\end{aligned} \tag{2.81}
$$

where we set $k(\cdot, \cdot)$ to be the Matérn(7/2) kernel, and use the state-space model to sample the GP for all N. The input locations are sampled over a grid between -5 and 5. In order to reduce fluctuations in runtime we averaged empirical runtimes over 10 independent runs and we also set the number of EP iterations to be fixed at 10 for both algorithms. The results, shown in Figure 2.6, clearly illustrates the improvement from cubic runtime complexity to $\mathcal{O}(N \log N)$.

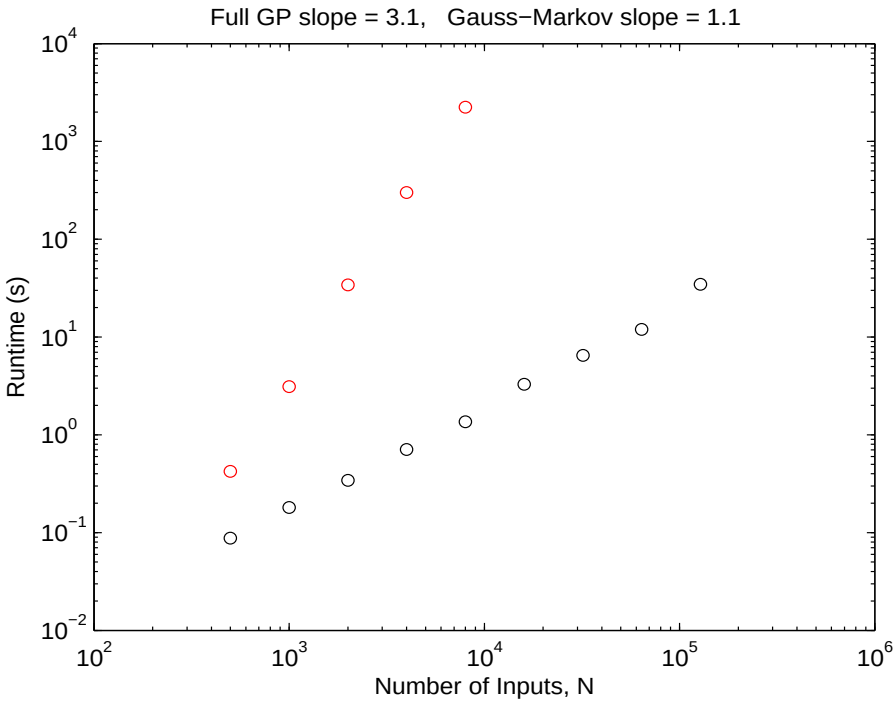


Figure 2.6: Runtime comparison of the full GP and the Gauss-Markov process for synthetic classification tasks of varying size. The slope of the full GP in the log-log plot above is found to be 3.1 and that of the state-space model is 1.1. This clearly illustrates the improvement in runtime complexity from cubic to $\mathcal{O}(N \log N)$.

Figure 2.7 illustrates the results of inference for a synthetic binary classification dataset of size $N = 10000$ (generated in the same way as the above).

Figure 2.8 illustrates the results of inference for a period in the Whistler snowfall

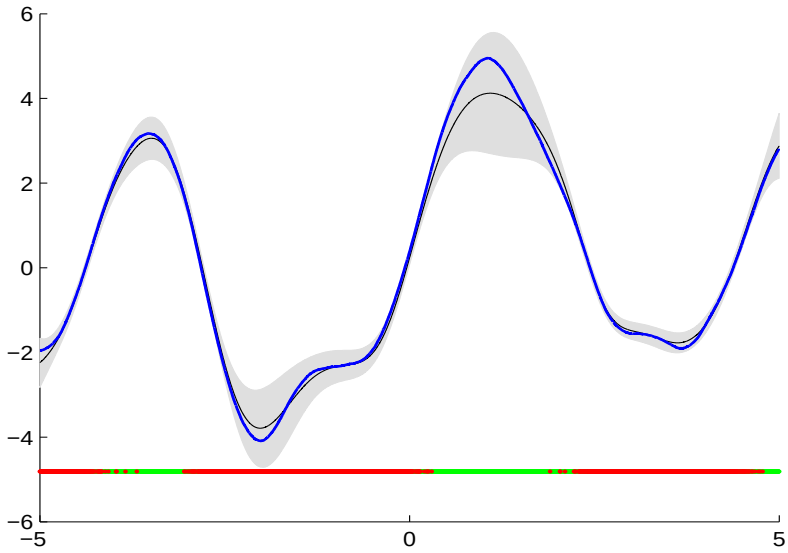


Figure 2.7: Example of inference using Gauss-Markov process classification. The bold blue line is the true function from which the binary labels are generated (via the probit transformation). The black line and grayscale give the posterior means and 2 standard deviation errorbars for the marginal predictions using the approximate posterior GP. Red and green dots indicate negative and positive observations, respectively.

dataset where the amount of yearly snowfall shows a surprising increase. Note that, although the Whistler snowfall data is ordinarily a regression dataset where the target represents the amount of snowfall, it has been converted to a binary classification task by thresholding at 0. Thus, the classes represent whether it snowed on that day or not. The hyperparameters were optimised using EM and we found to be 37 for the lengthscale (in days) and 2.3 for the signal amplitude. Note that the full learning process took only circa 400 seconds for this large dataset ($N = 13880$). For fixed hyperparameters, approximate inference is complete in around 8 seconds! Note that Figure 2.8 shows the inference result for only a subset of the time series. For the undisplayed part of the time series the periodicity is found to be far more regular and similar to the shape seen in the first and last 500 input points in Figure 2.8.

For results specific to standard Gauss-Markov process regression (with Gaussian emission density), refer to Hartikainen and Särkkä [2010]. The speedup attained for this case will also be made apparent in Chapter 4 where we use Algorithm 2 as a core routine for learning additive GP models.

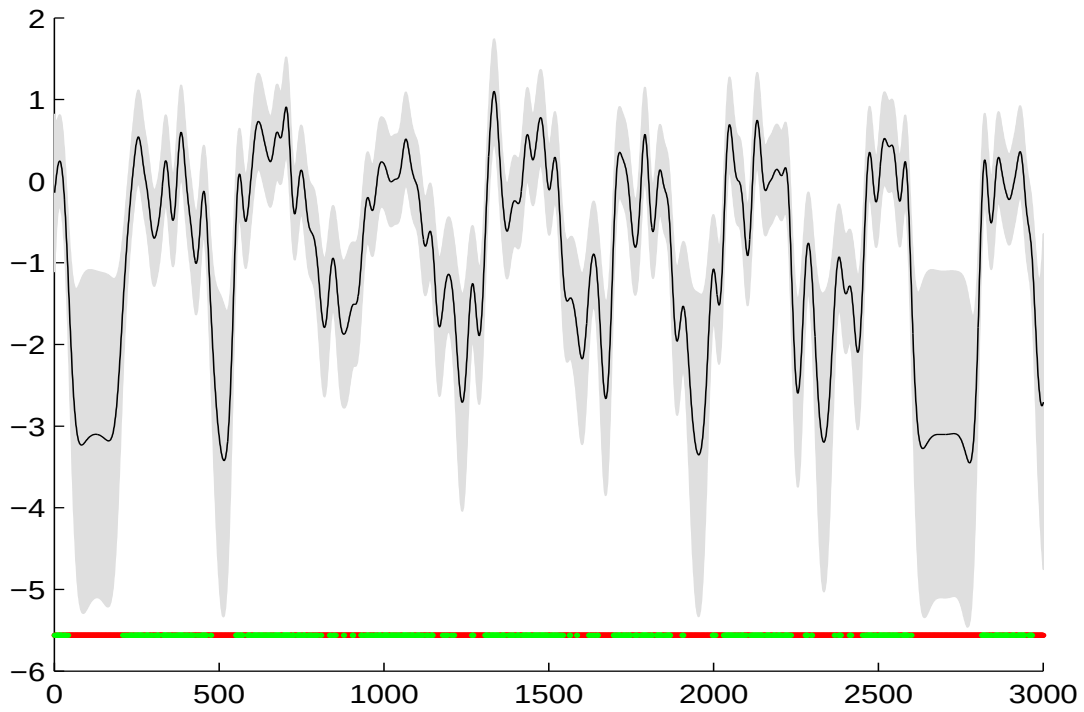


Figure 2.8: Result of running Algorithm 4 on the Whistler snowfall dataset. The time interval corresponds to a sequence of years where snowfall is more frequent than usual.

Chapter 3

Gaussian Processes with Change Points

This chapter describes the combination of sequential GP models with the Bayesian online changepoint detection algorithm of Adams and MacKay [2007], for the purposes of handling nonstationarity in the underlying data stream. It is an adaptation of the material already published in Saatçi et al. [2010]. As we have been focussing on using improving the efficiency of GPs using problem-specific structure, we will focus more on cases where we can use nonstationarity to improve prediction accuracy *and* computational complexity. In particular, we will use K-best pruning to curb the growing size of the hypothesis space. We will also generally assume that the underlying GP kernel cannot be represented as a Gauss-Markov process, and that the inputs are not in discrete time, in which case one can additionally use Toeplitz matrix methods, as done in Cunningham et al. [2008]. We note that if the GP kernel does indeed have such additional structure, then it is reasonable to expect that using GPs with change points will add to the overall complexity.

3.1 Introduction

In Chapter 2 we presented algorithms which exploited structured kernels that can be represented as a finite order SDE. This enabled us to perform (generalised) regression in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space – a significant improvement over standard GP regression. However, there are many covariance functions that cannot be rep-

resented as a Gauss-Markov process for scalar inputs. For example, no Markovian representation could be found for the squared-exponential covariance, although, in practice it is possible to approximate it with a high-order Matern kernel (see Section 2.2.2). Another example would be any kernel which has a *periodic* component, since this would add a delta spike to the spectral density. It is not possible to accurately model a spike with a polynomial in ω^2 in the denominator of the spectral density. In addition, more complicated kernels which have been constructed as functions of more basic covariance functions often corrupt the Markovian structure which may be present in their building blocks.

In this chapter, instead of making assumptions about the kernel, we will make assumptions about *the data*. In particular we will analyze the case where we have *time-series data* (i.e., our input space is time) which is expected to exhibit *nonstationarity*. Nonstationarity, or changes in generative parameters, is often a key aspect of real world time series which is usually not modelled by a GP prior (since it is stationary). As an inability to react to regime changes can have a detrimental impact on predictive performance, the aim of this chapter is to improve both runtime *and* predictive performance by expecting the given time series to be nonstationary. We will combine GP regression with Bayesian online change point detection (BOCPD) in order to recognize regime change events and adapt the predictive model appropriately. We focus on online algorithms as many time series data arise as part of a online stream of data. Examples of settings where such data arise includes automatic trading systems, satellite security systems, and adaptive compression algorithms, to name a few.

The Bayesian Online CPD (BOCPD) algorithm was recently introduced in Adams and MacKay [2007], and similar work has been done by Barry and Hartigan [1993], Barry and Hartigan [1992], Chib [1998], Ó Ruanaidh et al. [1994] and Fearnhead [2006]. Central to the online predictor is the time since the last change point, namely the *run length*. One can perform exact online inference about the run length for every incoming observation, given an *underlying predictive model* (UPM) and a *hazard function*. Given all the observations so far obtained, $y(t_1) \dots y(t_{i-1}) \in \mathbb{R}$, the UPM is defined to be $p(y(t_i)|y(t_{i-r}), \dots, y(t_{i-1}), \theta_m)$ for any *runlength* $r \in [0, \dots, (i-1)]$. θ_m denotes the set of hyperparameters associated with the UPM. The UPM can be thought of as a simpler base model whose parameters change at every change

point. Viewed from a generative perspective, during a change point these parameters change to be a different random sample from the parameter prior controlled by hyperparameters θ_m. Therefore, in order to construct nonstationary GPs using BOCPD, it is sufficient to simply implement predictions of the form $p(y(t_i)|y(t_{i-r}), \ldots, y(t_{i-1}), \theta_m)$ where θ_m parameterizes a prior distribution over GP hyperparameters. The other crucial ingredient of BOCPD is the hazard function $H(r|\theta_h)$, which describes how likely we believe a change point is given an observed run length r. Notice that through $H(r|\theta_h)$ we can specify, *a priori*, arbitrary duration distributions for parameter regimes. These duration distributions are parameterized by θ_h.

The standard BOCPD algorithm treats its hyper-parameters, $\theta := \{\theta_h, \theta_m\}$, as fixed and known. It is clear empirically that the performance of the algorithm is highly sensitive to hyperparameter settings and thus, we will extend the Algorithm in Adams and MacKay [2007] such that hyperparameter learning can be performed in a principled manner.

In the following, we will use the notation y_i to mean $y(t_i)$ (and similarly for the runlength variable). Notice that the time points t_i are allowed to be located at any point on the positive real line (i.e., they don't have to be equispaced).

3.2 The BOCPD Algorithm

BOCPD calculates the posterior run length at time t_i, i.e. $p(r_i|y_1, \ldots, y_{i-1})$, sequentially. This posterior can be used to make online predictions robust to underlying regime changes, through marginalization of the run length variable:

$$\begin{aligned} p(y_{i+1}|y_1, \ldots, y_i) &= \sum_{r_i} p(y_{i+1}|y_1, \ldots, y_i, r_i) p(r_i|\mathbf{y}_{1:i}), \\ &= \sum_{r_i} p(y_{i+1}|\mathbf{y}^{(r_i)}) p(r_i|\mathbf{y}_{1:i}) \,, \end{aligned} \tag{3.1}$$

where $\mathbf{y}^{(r_i)}$ refers to the last r_i observations of $\mathbf{y}$, and $p(y_{i+1}|\mathbf{y}^{(r_i)})$ is computed using the UPM. The run length posterior can be found by normalizing the joint likelihood: $p(r_i|\mathbf{y}_{1:i}) = \frac{p(r_i, \mathbf{y}_{1:i})}{\sum_{r_i} p(r_i, \mathbf{y}_{1:i})}$. The joint likelihood can be updated online using a recursive

message passing scheme

$$\begin{aligned}\gamma_i \equiv p(r_i, \mathbf{y}_{1:i}) &= \sum_{r_{i-1}} p(r_i, r_{i-1}, \mathbf{y}_{1:t}), \\ &= \sum_{r_{i-1}} p(r_i, y_i | r_{i-1}, \mathbf{y}_{1:i-1}) p(r_{i-1}, \mathbf{y}_{1:i-1}), \qquad (3.2) \\ &= \sum_{r_{i-1}} \underbrace{p(r_i|r_{i-1})}_{\text{hazard}} \underbrace{p(y_i|\mathbf{y}^{(r_{i-1})})}_{\text{UPM}} \underbrace{p(r_{i-1}, \mathbf{y}_{1:i-1})}_{\gamma_{i-1}}. \qquad (3.3)\end{aligned}$$

In the final step we have assumed that $p(r_i|r_{i-1}, \mathbf{y}_{1:i}) = p(r_i|r_{i-1})$, i.e., the hazard function is independent of the observations $\mathbf{y}$. The validity of such an assumption will depend on the nature of the nonstationarity observed in the data[1]. Equation 3.3 defines a forward message passing scheme to recursively calculate γ_i from γ_{i-1}. The conditional can be restated in terms of messages as $p(r_i|\mathbf{y}_{1:i}) \propto \gamma_i$. All the distributions mentioned so far are implicitly conditioned on the set of hyperparameters θ.

Example BOCPD model. A simple example of BOCPD would be to use a constant hazard function $H(r|\theta_h) := \theta_h$, meaning $p(r_i = 0|r_{i-1}, \theta_h)$ is independent of r_{i-1} and is constant, giving rise, *a priori*, to geometric inter-arrival times[2] for change points. The UPM can be set to the predictive distribution obtained when placing a Normal-Inverse-Gamma prior on IID Gaussian observations (i.e., a Student-t predictive):

$$y_i \sim \mathcal{N}(\mu, \sigma^2), \qquad (3.4)$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma^2/\kappa), \ \sigma^{-2} \sim \Gamma(\alpha, \beta). \qquad (3.5)$$

In this example $\theta_m := \{\mu_0, \kappa, \alpha, \beta\}$. Figure 3.1 illustrates a dataset drawn from the generative model of the example BOCPD model in Equations 3.4 and 3.5 (with $[\mu_0, \kappa, \alpha, \beta] = [0, 0.01, 10, 1]$, and $\theta_h = 0.1$), and the results of online runlength inference using the true hyperparameter values.

[1] A good example of where this assumption may fail would be a bond yield time series, which may exhibit a switch in volatility if it goes past an arbitrary psychological barrier such as 7%!

[2] Note that the Geometric inter-arrival is in terms of the number of observations which may not be equidistant in time.

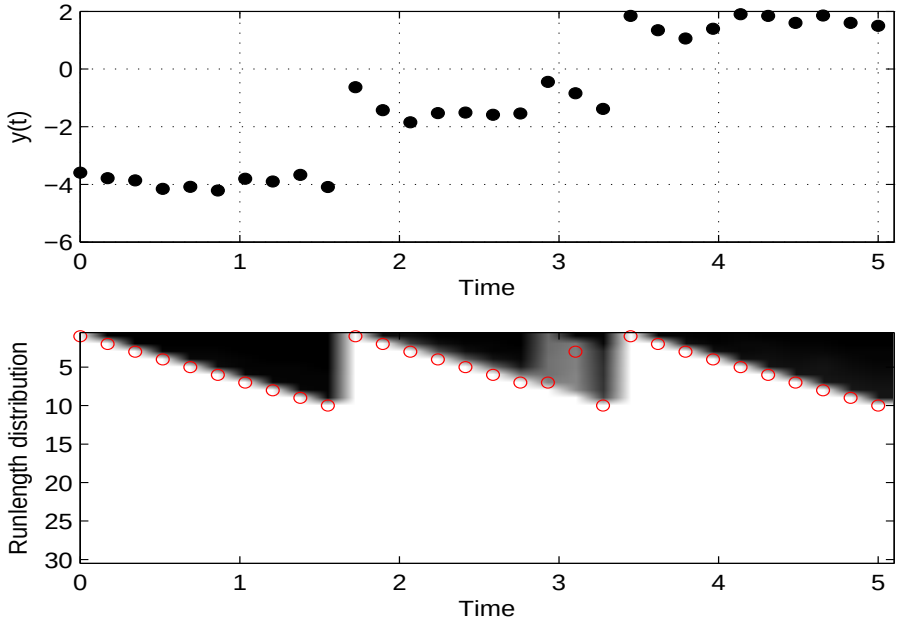


Figure 3.1: (Top) Data drawn from the example BOCPD model, where there is a regime change every 10 time steps. The time inputs are uniformly spaced in this particular example. (Bottom) The resulting inferred runlength distribution, where the red dots give Median$(r_i|y_{1:i})$, and the shading represents the marginal c.d.f. at time-step i.

3.3 Hyper-parameter Learning

It is possible to evaluate the (log) marginal likelihood of the BOCPD model at time T, as it can be decomposed into the one-step-ahead predictive likelihoods (see (3.1)):

$$\log p(\mathbf{y}_{1:T}|\theta) = \sum_{t=1}^{T} \log p(y_i|\mathbf{y}_{1:i-1}, \theta). \tag{3.6}$$

Hence, we can compute the derivatives of the log marginal likelihood using the derivatives of the one-step-ahead predictive likelihoods. These derivatives can be found in the same recursive manner as the predictive likelihoods. Using the derivatives of the UPM, $\frac{\partial p(y_i|r_{i-1},\mathbf{y}^{r_{i-1}},\theta_m)}{\partial \theta_m}$, and those of the hazard function, $\frac{\partial p(r_t|r_{t-1},\theta_h)}{\partial \theta_h}$, the derivatives of the one-step-ahead predictors can be propagated forward using the chain rule. The derivatives with respect to the hyper-parameters can be plugged into a conjugate gradient optimizer to perform hyper-parameter learning. The full BOCPD algorithm is given in Algorithm 5.

Algorithm 5: BOCPD Algorithm (with derivatives).

inputs : UPM $\mathcal{U}$ and Hazard function $\mathcal{H}$. Online feed of targets $\mathbf{y}$.
outputs: Runlength distributions $\{p(r_i|\mathbf{y}_{1:i})\}_{i=1,\ldots,T}$.
Marginal likelihood and derivatives.

for $i = 1, \ldots, T$ **do**
  `// Define` $\bar{\boldsymbol{\gamma}}_i$ `as` $\boldsymbol{\gamma}_i[2 : i+1]$
  $\boldsymbol{\pi}_i^{(r)} \leftarrow \mathcal{U}(y_i|\mathbf{y}_{1:i-1})$;
  $\mathbf{h} \leftarrow \mathcal{H}(1 : i)$;
  $\bar{\boldsymbol{\gamma}}_i \leftarrow \boldsymbol{\gamma}_{i-1}\boldsymbol{\pi}_i^{(r)}(1 - \mathbf{h})$; `//Update messages: no new change point.`
  $\partial_h\bar{\boldsymbol{\gamma}}_i \leftarrow \boldsymbol{\pi}_i^{(r)}(\partial_h\boldsymbol{\gamma}_{i-1}(1 - \mathbf{h}) - \boldsymbol{\gamma}_{i-1}\partial_h\mathbf{h})$;
  $\partial_m\bar{\boldsymbol{\gamma}}_i \leftarrow (1 - \mathbf{h})(\partial_m\boldsymbol{\gamma}_{i-1}\boldsymbol{\pi}_i^{(r)} + \boldsymbol{\gamma}_{i-1}\partial_m\boldsymbol{\pi}_i^{(r)})$;
  $\boldsymbol{\gamma}_i[1] \leftarrow \sum \boldsymbol{\gamma}_{i-1}\boldsymbol{\pi}_i^{(r)}\mathbf{h}$; `//Update messages: new change point.`
  $\partial_h\boldsymbol{\gamma}_i[1] \leftarrow \sum \boldsymbol{\pi}_i^{(r)}(\partial_h\boldsymbol{\gamma}_{i-1}\mathbf{h} + \boldsymbol{\gamma}_{i-1}\partial_h\mathbf{h})$;
  $\partial_m\boldsymbol{\gamma}_i[1] \leftarrow \sum \mathbf{h}(\partial_m\boldsymbol{\gamma}_{i-1}\boldsymbol{\pi}_i^{(r)} + \boldsymbol{\gamma}_{i-1}\partial_m\boldsymbol{\pi}_i^{(r)})$;
  $p(r_i|\mathbf{y}_{1:i}) \leftarrow$ normalise $\boldsymbol{\gamma}_i$;
end
$p(\mathbf{y}|\theta) \leftarrow \sum \boldsymbol{\gamma}_T$; `//Marginal likelihood.`
$\partial p(\mathbf{y}|\theta) \leftarrow (\sum \partial_h\boldsymbol{\gamma}_T, \sum \partial_m\boldsymbol{\gamma}_T)$; `//...and its derivatives.`

3.4 GP-based UPMs

In this chapter, we are primarily concerned with online, GP-based time-series prediction problems where standard GP hyperparameters (e.g., lengthscales) are assumed to switch at every change point. If one desires to model changes in GP hyperparameters at every change point, then the BOCPD algorithm dictates that one should integrate them out within the UPM. As a result, the desired UPM is given by:

$$p(y_i|\mathbf{y}^{r_{i-1}}, \theta_m) = \int p(y_i|\mathbf{y}^{r_{i-1}}, \lambda) p(\lambda|\mathbf{y}^{r_{i-1}}) \, d\lambda, \tag{3.7}$$

$$= \frac{1}{Z} \int p(y_i|\mathbf{y}^{r_{i-1}}, \lambda) p(\mathbf{y}^{r_{i-1}}|\lambda) p(\lambda|\theta_m) \, d\lambda \,, \tag{3.8}$$

where $Z := \int p(\mathbf{y}^{r_{i-1}}|\lambda) p(\lambda|\theta_m) \, d\lambda$. λ are the standard (scalar) GP hyperparameters, e.g. $\lambda \equiv \{\ell, \sigma_f^2, \sigma_n^2\}$ for the squared-exponential kernel. Notice that the density $p(y_i|\mathbf{y}^{r_{i-1}}, \lambda) = \mathcal{N}(\mu_i, v_i)$ is the standard GP predictive density given in Chapter 1. $p(\mathbf{y}^{r_{i-1}}|\lambda)$ is simply the marginal likelihood of the GP. As the log marginal likelihood is a nonlinear function of λ, both the integrals present in (3.8) are intractable, even if one sets $p(\lambda|\theta_m)$ to be a Gaussian prior over λ. Consequently, we approximate these integrals using two methods, each with their own set of pros and cons. In the first technique we place a grid ($\{\lambda_g\}$) over a subspace of GP hyper-parameters that is assumed to be reasonable for the problem at hand (assigning uniform prior mass for each grid point). The integrals can then be approximated with sums:

$$p(y_i|\mathbf{y}^{r_{i-1}}, \theta_m) \approx \sum_{\lambda_g} p(y_i|\mathbf{y}^{r_{i-1}}, \lambda_g) \left(\frac{p(\mathbf{y}^{r_{i-1}}|\lambda_g)}{\sum_{\lambda_g} p(\mathbf{y}^{r_{i-1}}|\lambda_g)} \right) . \tag{3.9}$$

Recall that it is tricky to apply more sophisticated quadrature algorithms for (3.8) as the target function is positive, and the interpolant runs the risk of becoming negative, although there are cases in the literature where one does interpolate despite this risk, see Rasmussen and Ghahramani [2003] and Garnett et al. [2009]. The grid method does not scale with increasing dimensionality, however, it offers the opportunity to use *rank-one updates* for computing predictive means and variances as one only considers a fixed set of hyper-parameter settings (see Appendix A). In addition, it is generally possible to integrate out the signal variance hyperparameter

$1/\sigma_f^2$ *analytically*. Thus, for a kernel such as the squared-exponential, it is possible to perform hyperparameter integration using a grid over a *two-dimensional* space. This makes the grid method much more manageable in practice. The trick is to rewrite the kernel as follows:

$$k(t_i, t_j) = \sigma_f^2 \exp\left(-\frac{(t_i - t_j)^2}{\ell^2}\right) + \sigma_n^2 \delta(t_i, t_j), \tag{3.10}$$

$$= \frac{1}{\tau}\left[\exp\left(-\frac{(t_i - t_j)^2}{\ell^2}\right) + \xi\delta(t_i, t_j)\right]. \tag{3.11}$$

where $\tau \equiv \frac{1}{\sigma_f^2}$ and $\xi \equiv \frac{\sigma_n^2}{\sigma_f^2}$. Notice that ξ can be interpreted as a *signal-to-noise-ratio* hyperparameter. Full hyperparameter integration can now be performed by analytically integrating out τ using a $\Gamma(\alpha_0, \beta_0)$ prior and integrating out $\lambda_g = \{\ell, \xi\}$ using a grid. Referring back to Equation 3.9, we can compute $p(\mathbf{y}^{r_{i-1}}|\lambda_g)$ using:

$$p(\mathbf{y}|\lambda_g) = \frac{p(\mathbf{y}|\tau, \lambda_g)p(\tau)}{p(\tau|\mathbf{y})}, \tag{3.12}$$

$$= \frac{\mathcal{N}(\mathbf{y}|\mathbf{0}, K(\lambda_g)/\tau)\Gamma(\tau|\alpha_0, \beta_0)}{\Gamma(\tau|\alpha_N, \beta_N)}, \tag{3.13}$$

$$\propto \text{Student-t}_{2\alpha_0}\left(\mathbf{0}, \frac{\beta_0}{\alpha_0}K(\lambda_g)\right), \tag{3.14}$$

where $\alpha_N = \alpha_0 + \frac{N}{2}$ and $\beta_N = \beta_0 + \frac{1}{2}\mathbf{y}^\top K(\lambda_g)^{-1}\mathbf{y}$. Similarly, we compute $p(y_i|\mathbf{y}^{r_{i-1}}, \lambda_g)$ using the posterior predictive given by:

$$p(y_\star|\mathbf{y}, \lambda_g) = \int p(y_\star|\mathbf{y}, \tau, \lambda_g)p(\tau|\mathbf{y}, \lambda_g)d\tau, \tag{3.15}$$

$$= \int \mathcal{N}(y_\star|\mu_\star, \sigma_\star/\tau)\Gamma(\tau|\alpha_N, \beta_N)d\tau, \tag{3.16}$$

$$= \text{Student-t}_{2\alpha_N}\left(\mu_\star, \frac{\beta_N}{\alpha_N}\sigma_\star^2\right). \tag{3.17}$$

$\mu_\star$ and $\sigma_\star^2$ are given by the standard GP predictive means and variances for given τ and λ_g. Note that in the above we have omitted conditioning on the input locations on the time axis in order to reduce clutter. Also note that, in practice, we do not use Equation 3.14 explicitly. Rather, given the sequential nature of the BOCPD algorithm, we use the chain rule to calculate $p(\mathbf{y}|\lambda_g)$ by repeatedly using Equation

3.17. In a similar vein, we use standard rank-one update equations to calculate $\mu_\star$ and $\sigma_\star^2$ (see Appendix A).

An alternative to the grid-based method which does scale with a higher number of hyperparameters is to use Hamiltonian Monte Carlo (HMC), as introduced in Duane et al. [1987]. Say we have computed samples $\{\lambda_s\}$ representing the posterior $p(\lambda|\mathbf{y}^{r_{i-1}})$. Then,

$$p(y_i|\mathbf{y}^{r_{i-1}}, \theta_m) \approx \sum_{\lambda_s} p(y_i|\mathbf{y}^{r_{i-1}}, \lambda_s)\,. \tag{3.18}$$

The samples can be updated sequentially for each run length hypothesis considered. The samples for $r_t = 0$ are straightforward as they come from the Gaussian prior $p(\lambda|\theta_m) = \mathcal{N}(\phi_m, \phi_v)$: we can trivially obtain IID samples at this stage. Note that, in this case $\theta_m \equiv \{\phi_m, \phi_v\}$. As the posterior $p(\lambda|\mathbf{y}_{(t-\tau):(t-1)})$, represented by samples $\{\lambda_s^{(t-1)}\}$, will look similar to $p(\lambda|\mathbf{y}_{(t-\tau):t})$, we can initialize the HMC sampler at $\{\lambda_s^{(t-1)}\}$ and run it for a short number of iterations for each sample. In practice, we have found that 4 *trajectories* with a mean of 11 *leapfrog steps* give respectable results. For the HMC algorithm it is not necessary to integrate out the signal variance hyperparameter as the difference in performance between sampling over a 3-dimensional hyperparameter space versus a 2-dimensional one is negligible. Note that other Sequential Monte Carlo (SMC) methods could be used also, though we have not explored these options.

3.5 Improving Efficiency by Pruning

The nonstationary GP model realised by using the UPMs in the previous section will always run slower than the standard, stationary GP over the entire time-series, in its vanilla form. This is because we need to update the runlength distribution at time-step i (this requires i updates from the previous step) and because each update requires an approximate integration over GP hyperparameters, as made evident by Equation 3.8. Given a time-series of length T, the complexity of the nonstationary GP would be $\mathcal{O}(T^4)$ if we are using the grid-based method and $\mathcal{O}(T^5)$ if we're using HMC. Indeed, this is not very surprising since the nonstationary GP model is much more flexible than a stationary one, and it makes sense to pay in computation for

the potential gains made in predictive accuracy. Nonetheless, it is common to find, for many nonstationary datasets, that the runlength distributions are *highly peaked*. This is evident for the toy example in Figure 3.1. There are many approximate inference schemes one could attempt in order to exploit the peaked nature of the runlength distributions. We will focus on the relatively simple strategy of *pruning out* runlength hypotheses which become extremely unlikely during the online run of the algorithm. We will consider two methods of pruning:

- Set all runlength hypotheses with mass less than a threshold ϵ to 0.
- Only consider the $\min(K, i)$ most probable runlength hypotheses at any time step i.

With pruning, the runtime complexity will be $\mathcal{O}(T\tilde{R}^2)$ for the grid-based method and $\mathcal{O}(T^2\tilde{R}^2)$ for HMC, where $\tilde{R}$ is the maximal run length not pruned out. Crucially, the memory complexity will also be limited to $\mathcal{O}(\tilde{R})$. Thus, for datasets which exhibit significant nonstationarity, it is reasonable to expect an improvement in efficiency *in addition* to improved accuracy, since typically $\tilde{R} \ll T$.

Note that there are more sophisticated pruning strategies one could use, such as those used for infinite hidden Markov models (see Van Gael et al. [2008]), although we do not consider them here.

3.6 Results

3.6.1 Experimental Setup

We will compare two alternative GP time-series models. The first is the stationary GP with the squared-exponential kernel, adapted to perform sequential predictions in an online manner using rank-one updates, as explained in Appendix A. The hyperparameters are optimised on the first N_{train} observations using Algorithm 5, and are kept fixed for the remainder of the time-series. The value of N_{train} will depend on the length of the particular time-series involved. We will refer to this method as **Stationary-GP**. The second model is a nonstationary GP where it is assumed, a priori, that at every change point the hyperparameters of the underlying GP undergo a switch. We focus on placing a uniform, discrete (i.e., grid-based) prior

on the hyperparameters as described in the previous section. Thus, we use the UPM given in Equation 3.9. We assume that the underlying GPs are well-modelled using squared-exponential kernels. We integrate out the signal variance hyperparameter analytically and place a grid on the rest, as explained in Section 3.4. Furthermore, we set $\alpha_0 = 1,\ \beta_0 = 1$ for the Gamma prior. We place a uniform grid of length 15 over $\log(\ell)$ between 0 and 10 and over $\log(\xi)$ between -5 and 0. We boost efficiency by using K-*best* pruning, with $K = 50$. We will refer to the nonstationary GP method as **NSGP-grid**. Recall that the complexity of **NSGP-grid** depends on the maximal runlength not pruned out, which in turn depends on the amount of nonstationarity present in the time-series, with respect to a GP-based UPM.

Note that all the results given here use the *vanilla* online GP algorithm either as the predictive model itself or as the UPM. Since we are using the squared-exponential kernel it would make sense to use the techniques of Chapter 2 to speed up computation significantly. In addition, for many of the real world time series analysed, the inputs are in discrete time and this enables the use of a well-known structured GP algorithm based on Toeplitz matrix methods (see Cunningham et al. [2008] for an excellent reference). Again, we will ignore this structure in this particular section, as we would like to focus on structure introduced by nonstationary data, as opposed to any other aspect of the problem at hand.

We use three performance metrics to compare the two algorithms above:

- Runtime, in seconds.
- Mean-squared error, as evaluated on the test set:

$$\text{MSE} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (y_i - \mathbb{E}(y_i | \mathbf{y}_{1:i-1}, \mathbf{y}_{\text{train}}))^2. \tag{3.19}$$

  This only tests the quality of the predictive means. In order to assess the quality of predictive variances we use the measure below.

- Mean Negative Log Likelihood, as evaluated on the test set:

$$\text{MNLL} = -\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \log p(y_i | \mathbf{y}_{1:i-1}, \mathbf{y}_{\text{train}}). \tag{3.20}$$

Recall that the predictive means and densities are calculated using Equation 3.6 for **NSGP-grid**.

3.6.2 Runtimes on Synthetic Data

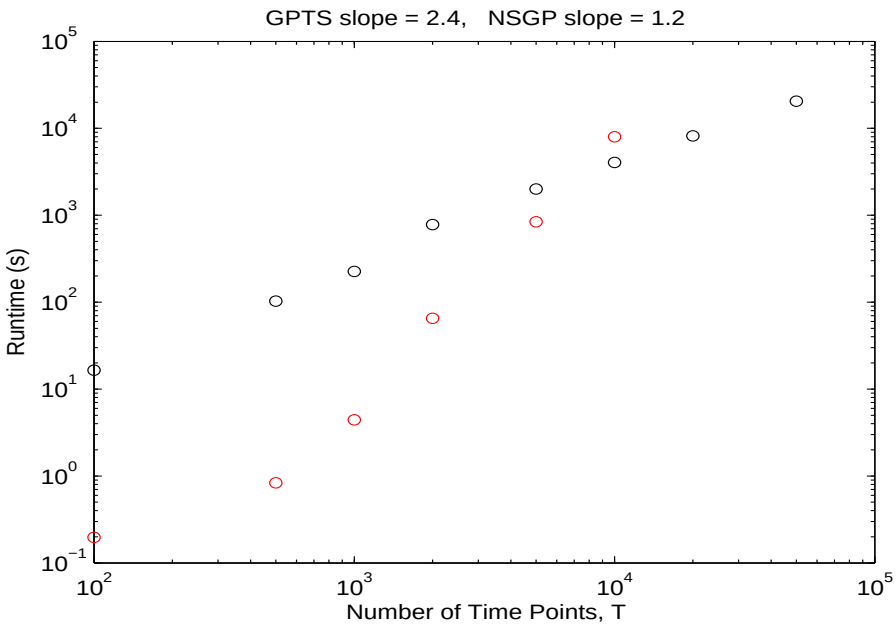


Figure 3.2: Comparison of runtimes for **Stationary-GP** (in red) and **NSGP-grid** (in black), using a log-log plot. **Stationary-GP** exhibits the familiar cubic scaling with T (its slope is 2.4), whereas **NSGP-grid** has linear scaling since the hazard rate ensures that regime lengths are extremely unlikely to exceed a length of around 500 (its slope is 1.2).

We measure the runtimes of **Stationary-GP** and **NSGP-grid** on time series of length $T = [100, 500, 1000, 2000, 5000, 10000, 20000, 50000]$. We generate synthetic nonstationary time series with GP-UPMs. We set the hazard rate to be constant at 0.01, so that the average segment length is 100. At every change point we sample the hyperparameters according to:

$$\log(\ell) \sim \mathcal{N}(0, 4), \;\; \log(\sigma_f) \sim \mathcal{N}(0, 2), \;\; \log(\sigma_n) \sim \mathcal{N}(-3, 1), \tag{3.21}$$

We sample the time points $t_1, \ldots, t_T$ according to the following scheme: if we are in

the first regime we generate time points from Uniform$(0, 1)$. For all other segments we generate time points from $t_{\text{final}} + \text{Uniform}(0, 1)$ where t_{final} is the time of the last observation of the previous segment. For these segments, we also condition the GP draw on the final observation of the previous segment, in order to avoid introducing discontinuities. The longest segment length sampled in the entire experiment was 642, and accordingly the longest runlength considered by the K-best scheme was 685. Consequently, we expect **NSGP-grid** to scale well with T, and this is clearly visible in Figure 3.2. Note that we have drawn data precisely from the generative model corresponding to **NSGP-grid** so we also expect superior performance in terms of MSE and MNLL. An example of this is given in Table 3.1. The posterior runlength distribution for $T = 500$ is given in Figure 3.3.

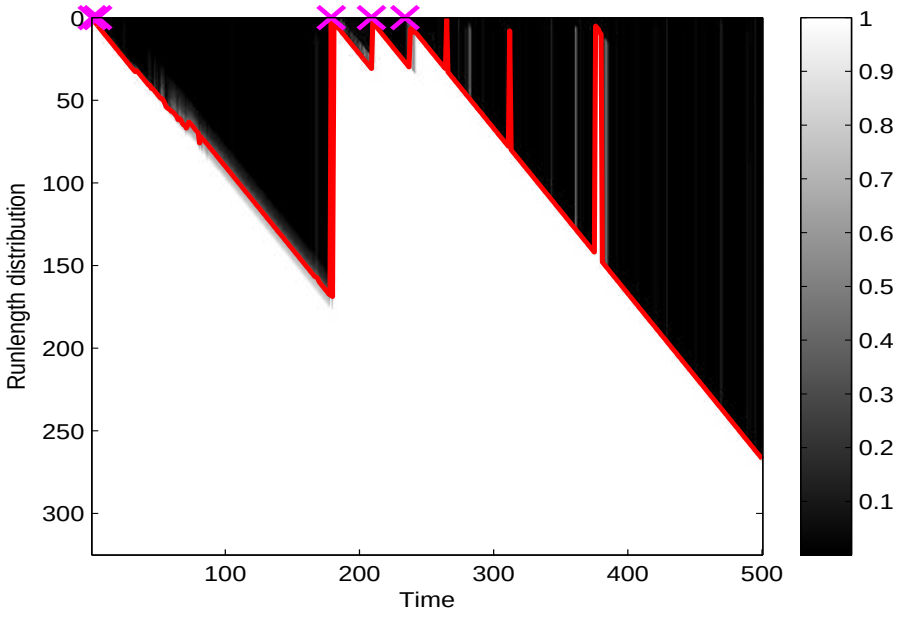


Figure 3.3: The posterior runlength distribution for synthetic nonstationary GP data after running **NSGP-grid**. True changepoints are marked in magenta. The change points arrive according to a constant hazard rate of 0.01.

3.6.3 Performance on Real-World Nonstationary Datasets

We first consider the Nile data set,[1] which has been used to test many change point methods Garnett et al. [2009]. The data set is a record of the lowest annual water levels on the Nile river during 622–1284 measured at the island of Roda, near Cairo, Egypt. There is domain knowledge suggesting a change point in year 715 due to an upgrade in ancient sensor technology to the Nilometer.

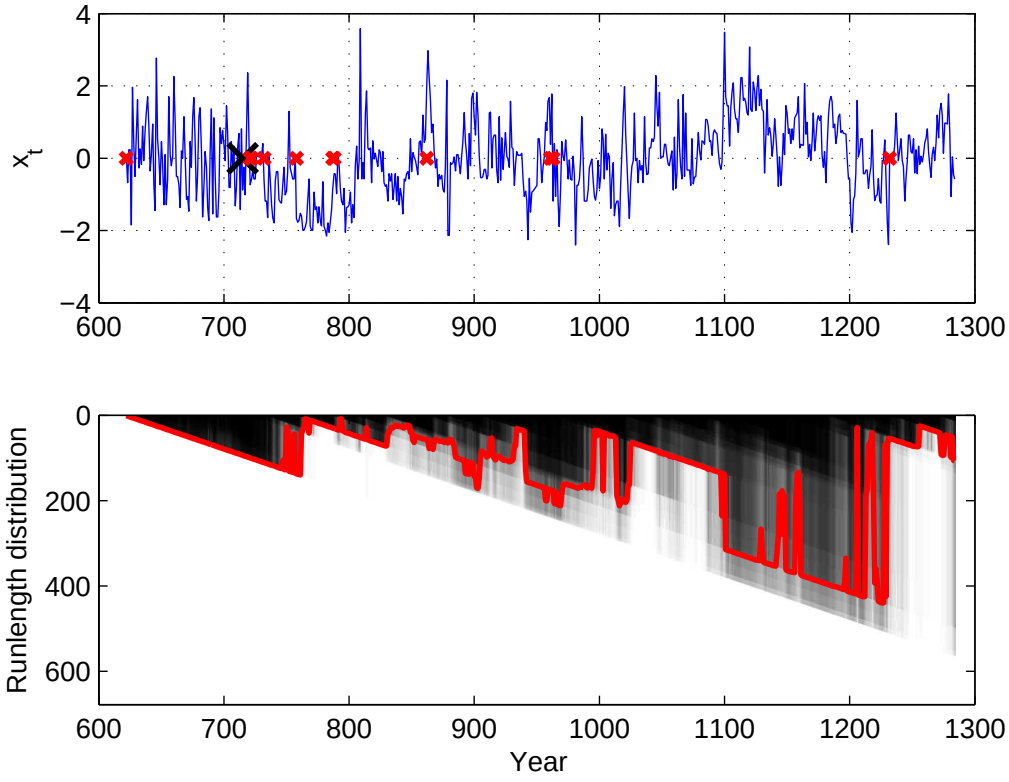


Figure 3.4: The output of applying NSGP to the Nile data, 622–1284. The large black cross marks the installation of the nilometer in 715. The small red crosses mark alert locations. We define an alert to be the points in the time series where the posterior mass corresponding to a change point is the highest *in total* over the entire run of the BOCPD algorithm. In this Figure we mark the top ten highest alert points – as can be seen, one of them is very close to the nilometer installation date.

We trained the (hyper) parameters of **Stationary-GP** on data from the first 200 years (622–821). The predictive performance of both the algorithms was evaluated

[1] http://lib.stat.cmu.edu/S/beran

on the following years, 822–1284. The run length posterior of **NSGP-grid** on the Nile data can be seen in Figure 3.4. The installation of the nilometer is the most visually noticeable change in the time series. Quantitative results in predictive performance are shown in Table 3.1.

Table 3.1: Performance on synthetic and real-world datasets. We see that **NSGP-grid** usually outperforms **Stationary-GP** in terms of predictive accuracy, and for larger datasets begins to dominate in terms of runtime as well.

Algorithm	MNLP	MSE	Runtime (s)
Synthetic Nonstationary Data ($N_{train} = 200$, $N_{test} = 800$)			
Stationary-GP	2.641	0.930	**36**
NSGP-grid	**−1.415**	**0.708**	311
Nile Data ($N_{train} = 200$, $N_{test} = 463$)			
Stationary-GP	1.373	0.873	**13**
NSGP-grid	**1.185**	**0.743**	268
Bee Waggle-Dance Data ($N_{train} = 250$, $N_{test} = 807$)			
Stationary-GP	1.587	0.786	**327**
NSGP-grid	**1.231**	**0.698**	838
Whistler Snowfall Data ($N_{train} = 500$, $N_{test} = 13380$)			
Stationary-GP	1.482	0.776	13273
NSGP-grid	**−1.985**	**0.618**	6243

We also consider the Bee waggle-dance dataset as introduced in Oh et al. [2008]. Honey bees perform what is known as a waggle dance on honeycombs. The three stage dance is used to communicate with other honey bees about the location of pollen and water. Ethologists are interested in identifying the change point from one stage to another to further decode the signals bees send to one another. The bee data set contains six videos of sequences of bee waggle dances.[1] The video files have been preprocessed to extract the bee's position and head-angle at each frame. While many in the literature have looked at the cosine and sine of the angle, we chose to analyze angle *differences*. Although the dataset includes the bee's head position and angle, we only consider running BOCPD on head angle differences, as we would like to remain in the domain of univariate time series. A more thorough

[1] http://www.cc.gatech.edu/~borg/ijcv_psslds/

treatment, where all the features are used as part of a multivariate time series is given in Saatçi et al. [2010]. We illustrate applying NSGP to "sequence 1" of the bee data in Figure 3.5. We trained **Stationary-GP** on the first 250 frames (four change points) and tested both algorithms on the remaining 807 frames (15 change points). We see that there is a reasonable correspondence between the likely change points identified by **NSGP-grid** and those marked by an expert. A more rigorous comparison can be made by simply using the ground-truth change points and evaluating the predictive performance of the resulting nonstationary GP model. This gives an MNLP of 1.438 and an MSE of 0.723, which is slightly worse than **NSGP-grid**. However, one must be wary of jumping to the conclusion that we are beating the experts, since the expert change point locations are based on using all features, not just the difference in head angle.

We also used historical daily snowfall data in Whistler, BC, Canada,[1] to evaluate our change point models. The models were trained on two years of data. We evaluated the models' ability to predict next day snowfall using 35 years of test data. A probabilistic model of the next day snowfall is of great interest to local skiers. In this data set, being able to adapt to different noise levels is key: there may be highly volatile snowfall during a storm and then no snow in between storms. Hence, **NSGP-grid** has an advantage in being able to adapt its noise level.

[1] http://www.climate.weatheroffice.ec.gc.ca/ (Whistler Roundhouse station, identifier 1108906).

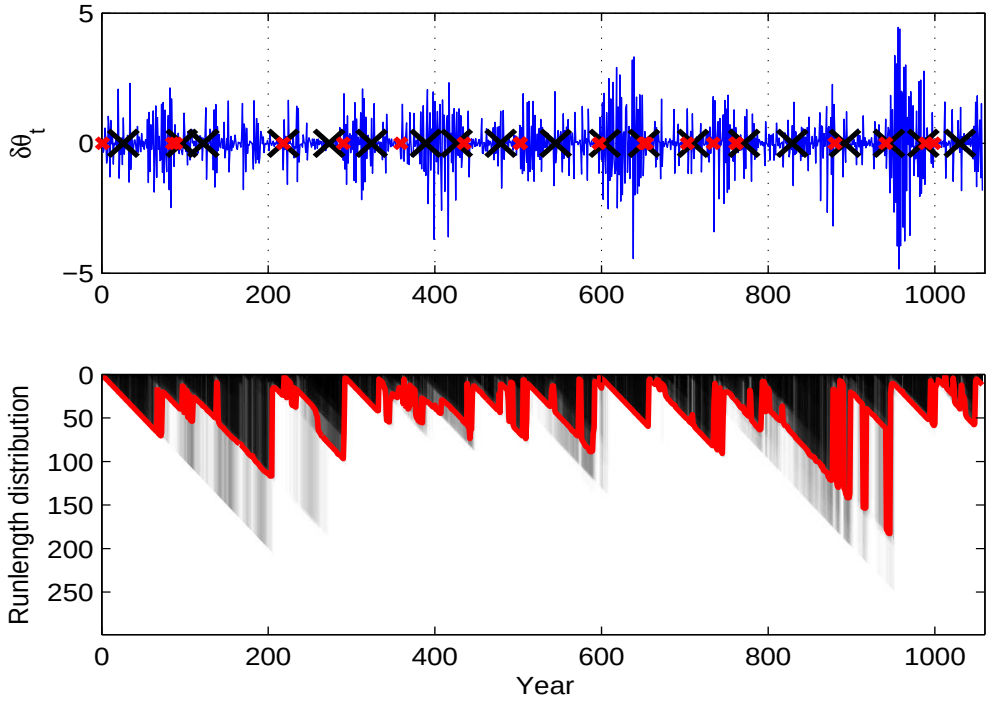


Figure 3.5: The output of applying NSGP to the Bee Waggle-Dance angle-difference time series. The large black marks correspond to changepoint locations labelled by an expert. The small red crosses mark alert locations. We define an alert to be the points in the time series where the posterior mass corresponding to a change point is the highest *in total* over the entire run of the BOCPD algorithm. In this Figure we mark the top twenty highest alert points – as can be seen, there is a good correspondence between the manually labelled changepoint locations and those inferred using BOCPD.

Chapter 4

Additive Gaussian Processes

As made clear in Hastie et al. [2009], a nonparametric regression technique (such as the spline smoother) which allows a scalable fit over a scalar input space can be used to fit an additive model over a D-dimensional space with the same overall asymptotic complexity. In this chapter we demonstrate that the same is true for Bayesian nonparametric regression using GPs. Thus, the central contribution is the synthesis of classical algorithms such as *backfitting* (Hastie and Tibshirani [1990]) and *projection-pursuit regression* (Friedman and Stuetzle [1981]) with efficient scalar GP regression techniques introduced in chapter 2.

4.1 Introduction

Chapters 2 and 3 have focussed on scalable inference and learning techniques for GPs defined on scalar input spaces. While there are many application domains for which the assumption of scalar inputs is appropriate (e.g. time series modelling), for most regression tasks it is usually the case that inputs are multidimensional. The central issue addressed in this chapter (and, to a certain extent, in chapter 5) is whether it is possible to incorporate and extend the ideas in previous chapters to multivariate input spaces. It turns out that this is indeed possible, if one is prepared to make some additional assumptions. In particular, we construct a number of GP regression and classification algorithms that run in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space (i.e., with the same complexity as the scalar Gauss-Markov process), by assuming *additivity*. Many of the algorithms presented in this chapter are founded on techniques used in

classical nonparametric regression, including *backfitting* Hastie and Tibshirani [1990] and Buja et al. [1989], *projection-pursuit regression* Friedman and Stuetzle [1981] and Friedman et al. [1984] and the *local scoring* algorithm Hastie and Tibshirani [1990] and Hastie et al. [2009]. The main contribution of this chapter is to introduce, adapt and extend these ideas in the context of efficient (generalised) GP regression.

Additive GP regression can be described using the following generative model:

$$y_i = \sum_{d=1}^{D} \mathbf{f}_d(X_{i,d}) + \epsilon \qquad i = 1, \ldots, N, \tag{4.1}$$

$$\mathbf{f}_d(\cdot) \sim \mathcal{GP}\left(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; \theta_d)\right) \qquad d = 1, \ldots, D, \tag{4.2}$$

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2).$$

θ_d represent the dimension-specific hyperparameters and σ_n^2 is the (global) noise hyperparameter ($\theta = [\theta_1, \ldots, \theta_D, \sigma_n^2]$). An additive model considers a restricted class of nonlinear multivariate functions which can be written as a sum of univariate ones. In terms of the flexibility of the functions supported, an additive prior lies between a linear model (which is also additive) and a nonlinear regression prior where arbitrary input interactions can be modelled (as is the case with, for example, the ARD kernel). Although interactions between input dimensions are not modelled, an additive model does offer *interpretable* results – one can simply plot, say, the posterior mean of the individual $\mathbf{f}_d$ to visualize how each predictor relates to the target. The usual way to perform inference and learning for an additive GP is to use the generic GP regression algorithm in Chapter 1. The covariance function can be written as:

$$k(\mathbf{x}, \mathbf{x}'; \theta) = \sum_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d; \theta_d), \tag{4.3}$$

as it is assumed that the individual univariate functions are independent, a priori. For example, for an additive squared-exponential GP, we have the following covariance function:

$$k(\mathbf{x}, \mathbf{x}'; \theta) = \sum_{d=1}^{D} v_d \exp\left(-\frac{(\mathbf{x}_d - \mathbf{x}'_d)^2}{2\ell_d^2}\right). \tag{4.4}$$

Equation 4.3 can be used to compute the training set covariance matrix $\mathbf{K}_N$ which

is then plugged into Algorithm 1. Naturally, this approach suffers from the same complexity as any other GP regression model. In Section 4.2, we will show that inference and learning can be performed using a sequence of *univariate* GP regression steps which can be executed efficiently using techniques presented in chapter 2, assuming that the kernel has an SDE representation in 1D. The resulting set of algorithms, all of which are novel adaptations of backfitting to GP regression, will be shown to scale much like the univariate regression scheme they are based on, thereby offering a significant improvement in complexity. Equation 4.4 provides an example of a kernel for which fully-Bayesian learning can be performed in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space.

Additivity in the original space of the covariates is an assumption which many real-world datasets do not satisfy, as predictors usually jointly affect the target variable. It is possible to remain within the boundaries of tractability *and* relax this assumption by considering a different feature space linearly related to the space in which the inputs live. This corresponds to the following generative model which we will refer to as the *projected* additive GP regression (PAGPR) model:

$$y_i = \sum_{m=1}^{M} \mathbf{f}_m(\mathbf{w}_m^\top \mathbf{x}_i) + \epsilon \qquad i = 1, \ldots, N, \tag{4.5}$$

$$\mathbf{f}_d(\cdot) \sim \mathcal{GP}\left(\mathbf{0}; k_m(\mathbf{w}_m^\top \mathbf{x}, \mathbf{w}_m^\top \mathbf{x}'; \theta_d)\right) \qquad m = 1, \ldots, M, \tag{4.6}$$

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2).$$

M represents the dimensionality of the feature space and can be greater than or less than D (these cases are referred to as overcomplete and undercomplete PAGPR, respectively). In Section 4.3 we construct a greedy learning algorithm for PAGPR which is as efficient as the case where the projection is simply the identity. The resulting algorithm can be viewed as a novel Bayesian extension of the classical projection-pursuit regression algorithm and will be referred to as projection-pursuit GP regression (PPGPR).

Somewhat surprisingly, it is in fact possible to extend the efficiency of additive GP regression to problems where *non-Gaussian* likelihoods are required, as is the case with, for example, classification where the target vector $\mathbf{y}$ consists of class labels rather than continuous quantities. In particular, we will be focussing on the

setup where the targets are related to the underlying function values via some *link function*, $g(\cdot)$. The generative model for such *generalised* additive regression models is given by:

$$\begin{aligned} y_i &\sim p_i \qquad i = 1, \ldots, N, \\ p_i &= g\left(\sum_{d=1}^{D} \mathbf{f}_d(X_{i,d})\right), \\ \mathbf{f}_d(\cdot) &\sim \mathcal{GP}\left(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; \theta_d)\right) \qquad d = 1, \ldots, D. \end{aligned} \tag{4.7}$$

In Section 4.4 we present an algorithm which implements Laplace's approximation (see Chapter 1) for generalised additive GP regression in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space, and highlight its connection to the local scoring algorithm.

4.2 Efficient Additive GP Regression

4.2.1 Backfitting and Exact MAP Inference

The backfitting algorithm for fitting additive regression models was first introduced in Breiman and Friedman [1985]. It is an intuitive iterative method for finding an additive fit, and is presented in Algorithm 6. Notice that backfitting is strictly only concerned with providing estimates of the underlying function values, as opposed to computing a posterior over them which is the central task in Bayesian regression. In practice, when running backfitting, it is necessary to zero-mean the targets as

Algorithm 6: The Classical Backfitting Algorithm

inputs : Training data $\{\mathbf{X}, \mathbf{y}\}$. Axis-aligned smoother parameters.
outputs: Smoothing Estimates : $\hat{\mathbf{f}}_d$.

Zero-mean the targets $\mathbf{y}$;
Initialise the $\hat{\mathbf{f}}_d$ (e.g. to $\mathbf{0}$);
while *The change in* $\hat{\mathbf{f}}_d$ *is above a threshold* **do**
    for $d = 1, \ldots, D$ **do**
        $\hat{\mathbf{f}}_d \leftarrow \mathbb{S}_d\left[\mathbf{y} - \sum_{j \neq d} \hat{\mathbf{f}}_j | \mathbf{X}_{:,d}\right]$; `//Use any 1D smoother as` $\mathbb{S}_d$
    end
end

any deterministic non-zero offset adds a component which is not identifiable by means of an additive model. The backfitting algorithm derives its efficiency from the fact that the smoothing operation in Algorithm 6, Line 5 can be performed in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ for a variety of smoothing schemes including spline and wavelet smoothing. In general the complexity of such an update is, of course, cubic in N. For a thorough exposition to the theory behind why backfitting is a valid algorithm to fit an additive regression model (using functional analysis), see Hastie and Tibshirani [1990].

In this chapter, we motivate the backfitting algorithm using a Bayesian perspective. Figure 4.1 gives the graphical model for additive GP regression. Note that in the Bayesian setup, the $\mathbf{f}_d$ are random variables, rather than quantities to be estimated, as is the case in Algorithm 6. Algorithm 7 gives the backfitting-style algorithm used to compute the posterior mean of an additive GP model. It is pos-

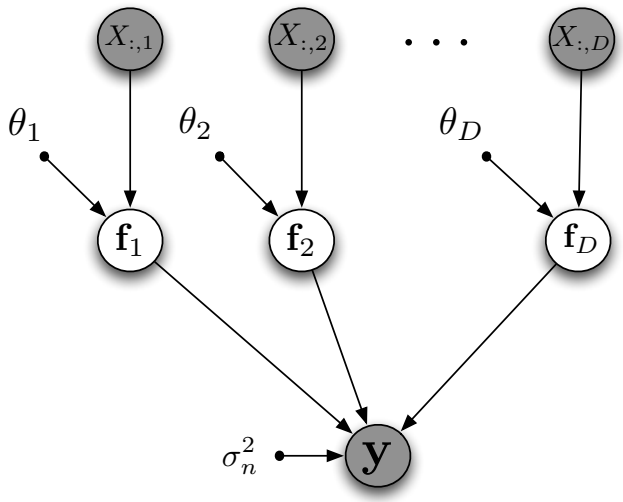


Figure 4.1: Graphical Model for Additive Regression. The targets $\mathbf{y}$ are assumed to be generated by means of a sum of D univariate functions, $\mathbf{f}_1, \ldots, \mathbf{f}_D$, corrupted by IID noise $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$.

sible to show that Algorithm 7 is nothing more than a block Gauss-Seidel iteration that computes the posterior means $\mathbb{E}(\mathbf{f}_d | \mathbf{y}, \mathbf{X}_{:,d}, \theta_d, \sigma_n^2)$ for $d = 1, \ldots, D$. We prove this important observation below. Recall that Gauss-Seidel is an iterative technique to solve the linear system of equations $\mathbf{A}\mathbf{z} = \mathbf{b}$. The ith component of $\mathbf{z}$ is updated by keeping all other $\mathbf{z}_j, j \neq i$, fixed and solving for $\mathbf{z}_i$ in $\sum_k A_{i,k}\mathbf{z}_k = \mathbf{b}_i$, giving the

Algorithm 7: Efficient Computation of Additive GP Posterior Mean

inputs : Training data $\{\mathbf{X}, \mathbf{y}\}$. Suitable covariance function. Hypers $\theta = \bigcup_{d=1}^{D} \{\theta_d\} \cup \sigma_n^2$.

outputs: Posterior training means: $\sum_{d=1}^{D} \boldsymbol{\mu}_d$, where $\boldsymbol{\mu}_d \equiv \mathbb{E}(\mathbf{f}_d | \mathbf{y}, \mathbf{X}, \theta_d, \sigma_n^2)$.

```
Zero-mean the targets y;
Initialise the μ_d (e.g. to 0);
while The change in μ_d is above a threshold do
    for d = 1, ..., D do
        μ_d ← E(f_d | y − Σ_{j≠d} μ_j, X_{:,d}, θ_d, σ_n^2);    //Use methods of Chap. 2
    end
end
```

following iterative update rule:

$$\mathbf{z}_i^{(k+1)} \leftarrow \frac{1}{A_{i,i}} \left[\mathbf{b}_i - \sum_{j>i} A_{i,j} \mathbf{z}_j^{(k)} - \sum_{j<i} A_{i,j} \mathbf{z}_j^{(k+1)} \right]. \tag{4.8}$$

The algorithm keeps cycling through the $\mathbf{z}_i$ until the change in $\mathbf{z}$ is negligible.

Theorem 4.1. *Algorithm 7 computes the posterior mean of an additive Gaussian Process given by*

$$\mathbb{E}(\mathbf{f}_1, \ldots, \mathbf{f}_D | \mathbf{y}, \mathbf{X}, \theta),$$

by means of a block Gauss-Seidel iteration, exactly.

Proof. Let $\mathbf{F}$ be a column-wise stacking of the individual $\mathbf{f}_d$, i.e. $\mathbf{F} \equiv [\mathbf{f}_1; \ldots; \mathbf{f}_D]$. Let $\mathbf{S}$ similarly be defined as a column-wise stacking of D N-by-N identity matrices: $\mathbf{S} \equiv [\mathbf{I}_N; \ldots; \mathbf{I}_N]$. Define $\mathbf{K}$ as a block-diagonal matrix given by:

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_N^{(1)} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_N^{(2)} & \ldots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{K}_N^{(D)} \end{bmatrix},$$

where $\mathbf{K}_N^{(d)}$ is $k_d(\cdot, \cdot)$ evaluated at the training inputs. We can then write:

$$p(\mathbf{F}|\mathbf{y}, X, \theta) \propto p(\mathbf{y}|\mathbf{F}, \theta)p(\mathbf{F}|\mathbf{X}, \theta) \tag{4.9}$$

$$= \mathcal{N}\left(\mathbf{y}; \mathbf{S}^\top \mathbf{F}, \sigma_n^2 \mathbf{I}_N\right) \mathcal{N}\left(\mathbf{F}; \mathbf{0}, \mathbf{K}\right). \tag{4.10}$$

Thus, the posterior $p(\mathbf{F}|\mathbf{y}, \mathbf{X}, \theta)$ is also Gaussian and the mean is given by:

$$\mathbb{E}(\mathbf{F}|\mathbf{X}, \mathbf{y}, \theta) = \left[\mathbf{K}^{-1} + \mathbf{S}(\sigma_n^{-2}\mathbf{I}_N)\mathbf{S}^\top\right]^{-1} \mathbf{S}(\sigma_n^{-2}\mathbf{I}_N)\mathbf{y}, \tag{4.11}$$

and is thus the solution to the following system of equations

$$\begin{bmatrix} (\mathbf{K}_N^{(1)})^{-1} + \sigma_n^{-2}\mathbf{I}_N & \dots & \sigma_n^{-2}\mathbf{I}_N \\ \vdots & \ddots & \vdots \\ \sigma_n^{-2}\mathbf{I}_N & \dots & (\mathbf{K}_N^{(D)})^{-1} + \sigma_n^{-2}\mathbf{I}_N \end{bmatrix} \begin{bmatrix} \mathbb{E}(\mathbf{f}_1) \\ \vdots \\ \mathbb{E}(\mathbf{f}_D) \end{bmatrix} = \begin{bmatrix} \sigma_n^{-2}\mathbf{I}_N\mathbf{y} \\ \vdots \\ \sigma_n^{-2}\mathbf{I}_N\mathbf{y} \end{bmatrix}. \tag{4.12}$$

An individual Gauss-Seidel update is then given by:

$$\mathbb{E}^{(k+1)}(\mathbf{f}_d) \leftarrow \left((\mathbf{K}_N^{(d)})^{-1} + \sigma_n^{-2}\mathbf{I}_N\right)^{-1} \sigma_n^{-2}\mathbf{I}_N \left[\mathbf{y} - \sum_{j>d} \mathbb{E}^{(k)}(\mathbf{f}_j) - \sum_{j<d} \mathbb{E}^{(k+1)}(\mathbf{f}_j)\right]$$

$$= \mathbf{K}_N^{(d)} \left(\mathbf{K}_N^{(d)} + \sigma_n^2\mathbf{I}_N\right)^{-1} \left[\mathbf{y} - \sum_{j>d} \mathbb{E}^{(k)}(\mathbf{f}_j)^{(k)} - \sum_{j<d} \mathbb{E}^{(k+1)}(\mathbf{f}_j)\right], \tag{4.13}$$

where in the final step we have used the matrix identity A.2. □

Equation 4.13 is identical to the expression giving the posterior mean (evaluated at input locations $\mathbf{X}_{:,d}$) of a *univariate* GP trained on targets $\left[\mathbf{y} - \sum_{j\neq d} \mathbb{E}(\mathbf{f}_j)\right]$. Given that the $k_d(\cdot,\cdot)$ are kernels which can be translated into an SDE, the expression in Equation 4.13 can be evaluated efficiently using the techniques of Chapter 2. Note that the posterior mean predictions of underlying function values at the training inputs are given by $\sum_{d=1}^{D} \boldsymbol{\mu}_d$ (see Algorithm 7). Mean predictions at test inputs can be included trivially by running the scalar GPs jointly over training and test inputs, as in Algorithm 2.

The equivalence between the frequentist estimate of the $\hat{\mathbf{f}}_d$ and the posterior mean of the corresponding GP-based Bayesian model occurs frequently in the domain of nonparametric regression. For example, in the case of splines, the 1D

smoothing operation has been shown (see Section 2.2.3) to be equivalent to computing the posterior mean of a spline-based GP prior, i.e.

$$\mathbb{S}_d[\mathbf{y}|\mathbf{X}_{:,d}] = \mathbb{E}(\mathbf{f}_d|\mathbf{y}, \mathbf{X}_{:,d}, \theta_{\text{sp}}). \tag{4.14}$$

As it is now clear that backfitting is a block Gauss-Seidel iteration, the convergence properties relate to those of the Gauss-Seidel algorithm. In general, for a linear system $\mathbf{Az} = \mathbf{b}$, Gauss-Seidel is guaranteed to converge, irrespective of initialisation, to the solution $\mathbf{z} = \mathbf{A}^{-1}\mathbf{b}$, given that $\mathbf{A}$ is a symmetric, positive definite matrix. In the case of Equation 4.11, it is straightforward to show, given positive definiteness of the individual K_N^d, that

$$\mathbf{v}^\top \left(\mathbf{K} + \mathbf{S}(\sigma_n^{-2}\mathbf{I}_N)\mathbf{S}^\top\right)\mathbf{v} > 0, \tag{4.15}$$

where $\mathbf{v}$ is a column-wise stacking of D arbitrary, real-valued, length-N vectors. The speed of convergence depends on the eigenvalues of the ND-by-ND linear mapping $\mathbf{G}$ relating the estimated solution at iteration $k+1$ to that at iteration k given by:

$$\begin{bmatrix} \mathbb{E}^{(k+1)}(\mathbf{f}_1) \\ \vdots \\ \mathbb{E}^{(k+1)}(\mathbf{f}_D) \end{bmatrix} = \mathbf{G} \begin{bmatrix} \mathbb{E}^{(k)}(\mathbf{f}_1) \\ \vdots \\ \mathbb{E}^{(k)}(\mathbf{f}_D) \end{bmatrix}. \tag{4.16}$$

Furthermore, the error in the solution at iteration k tends to zero as $\rho(\mathbf{G}) \equiv \max(|\lambda_G|)^k$ where λ_G is the set of eigenvalues of $\mathbf{G}$. Note that $\rho(G) < 1$ given the system of equations is positive definite. See Golub and Van Loan [1996] for more details about the proofs of these properties. The proximity of $\rho(G)$ to 1 depends on the hyperparameters of the GP. Note, however, that convergence is improved significantly by the fact that we are jointly updating blocks of N variables at a time. Consequently, in practice, the number of iterations required is far less than updating the unknowns one by one. We conjecture that the overall backfitting procedure in Algorithm 7 can be completed in $\mathcal{O}((N \log N)D^3)$ time and $\mathcal{O}(N)$ space, given we use the right type of additive kernel (in general, it would run in $\mathcal{O}((ND)^3)$ time and $\mathcal{O}(N^2)$ space).

4.2.2 Algorithms for a Fully-Bayesian Implementation

There are two important issues which we have not addressed so far. The first is the task of learning the hyperparameters $\theta = \bigcup_{d=1}^{D}\{\theta_d\} \cup \{\sigma_n^2\}$. The second is computing estimates of uncertainty about the underlying function values at training and test inputs, namely $\mathbb{V}(\mathbf{f}_d|\mathbf{y}, \mathbf{X}, \theta)$ and $\mathbb{V}(\mathbf{f}_{\star d}|\mathbf{y}, \mathbf{X}, \mathbf{X}_\star, \theta)$, for some fixed θ. In practice, one is usually interested only in the marginal variances of these quantities, so attention will be restricted to computing these only. As classical backfitting is non-Bayesian, it suffices to compute the estimates $\hat{\mathbf{f}}$ as done in Algorithm 6, however tuning of the hyperparameters is still required in practice. The classical solution uses generalised cross-validation to learn the hyperparameters of the smoothing operators, however, doing this without resorting to an $\mathcal{O}(N^3)$ algorithm requires approximations. Further details see, for example, the BRUTO algorithm in Hastie and Tibshirani [1990].

Similarly, for the Bayesian additive GP model, both hyperparameter learning and the computation of predictive variances requires the use of approximate inference techniques, if one is interested in preserving efficiency. In particular, we will resort to MCMC and a deterministic approximation scheme based on variational Bayes.

Markov Chain Monte Carlo

Figure 4.2 shows the graphical model for an additive GP model where we have additionally placed a prior over the hyperparameters θ. We have extended the model to include a prior over the hyperparameters because for an MCMC algorithm it is more natural to (approximately) integrate them out. This provides the additional benefit of reduced overfitting at the hyperparameter level. Equations 4.17 through to 4.21 give details of the generative model used.

$$\log \ell_d \sim \mathcal{N}(\mu_\ell, v_\ell) \qquad d = 1, \ldots, D, \tag{4.17}$$

$$\tau_d \sim \Gamma(\alpha_\tau, \beta_\tau) \qquad d = 1, \ldots, D, \tag{4.18}$$

$$\mathbf{f}_d(\cdot) \sim \mathcal{GP}\left(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}_d'; \ell_d, \tau_d)\right) \qquad d = 1, \ldots, D, \tag{4.19}$$

$$\tau_n \sim \Gamma(\alpha_n, \beta_n), \tag{4.20}$$

$$\mathbf{y} \sim \mathcal{N}\left(\sum_{d=1}^{D} \mathbf{f}_d(\mathbf{X}_{:,d}), \sigma_n^2 \mathbf{I}_N\right), \tag{4.21}$$

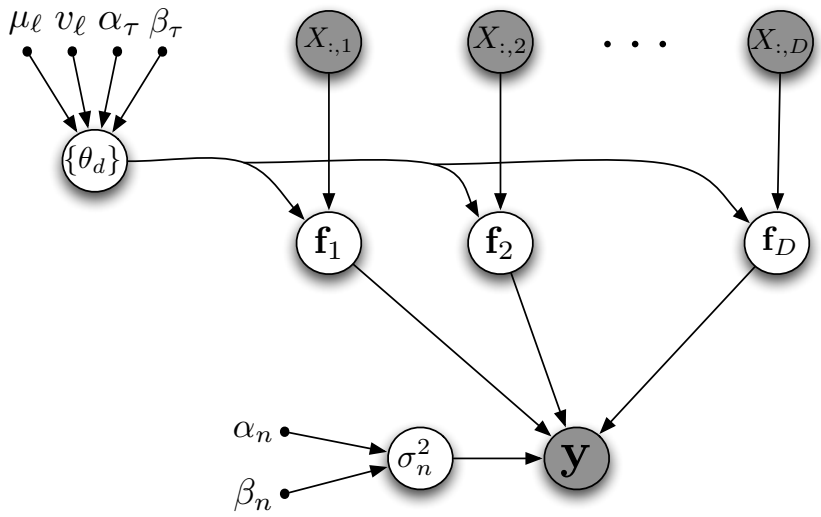


Figure 4.2: Graphical Model for fully-Bayesian Additive GP Regression. The hyperparameters for each univariate function $\mathbf{f}_d$ are given a prior parameterised by $\{\mu_l, v_l, \alpha_\tau, \beta_\tau\}$. We also place a $\Gamma(\alpha_n, \beta_n)$ prior over the noise precision hyperparameter. Inference using MCMC involves running Gibbs sampling over this graph.

where we have defined $\tau_d \equiv 1/v_d$ and $\tau_n \equiv 1/\sigma_n^2$, and have assumed that the $k_d(\cdot,\cdot)$ are parameterized with amplitude and lengthscale parameters (this captures most standard covariances). The overall algorithm runs full Gibbs sampling for this generative model. The parameters of the hyper-priors are set in advance and are kept fixed. For a probabilistic model over V variables, $X_1, \ldots, X_V$, Gibbs sampling runs a Markov Chain whose stationary distribution is the joint distribution over all variables, i.e. $p(X_1, \ldots, X_V)$. Each step of the Markov chain changes *one* variable at a time. The updates are performed using the univariate conditional distribution $p(X_i|X_{\neg i})$. Gibbs sampling can be viewed as a Metropolis-Hastings (MH) algorithm with a proposal distribution which results in an acceptance probability of 1. It has the advantage that there are no parameters to tune, unlike many MH proposal distributions. For further details, see Neal [1993] and MacKay [2003]. Application of Gibbs Sampling to the graph in Figure 4.2 results in the following conditional densities that need to be sampled from:

$$p(\mathbf{f}_d|\mathbf{f}_{\neg d}, \mathbf{y}, \mathbf{X}, \theta) \propto p(\mathbf{y}|\mathbf{f}_d, \mathbf{f}_{\neg d}, \mathbf{X}, \theta)p(\mathbf{f}_d|\mathbf{X}_{:,d}, \theta_d), \tag{4.22}$$

$$p(\ell_d|\mathbf{f}_d, \mathbf{X}_{:,d}, \tau_d, \mu_\ell, v_\ell) \propto p(\mathbf{f}_d|\mathbf{X}_{:,d}, \ell_d, \tau_d)p(\ell_d|\mu_\ell, v_\ell), \tag{4.23}$$

$$p(\tau_d|\mathbf{f}_d, \mathbf{X}_{:,d}, \ell_d, \alpha_\tau, \beta_\tau) \propto p(\mathbf{f}_d|\mathbf{X}_{:,d}, \ell_d, \tau_d)p(\tau_d|\alpha_\tau, \beta_\tau), \tag{4.24}$$

$$p(\tau_n|\mathbf{f}_1, \ldots, \mathbf{f}_D, \mathbf{y}, \alpha_n, \beta_n) \propto p(\mathbf{y}|\mathbf{f}_1, \ldots, \mathbf{f}_D, \tau_n)p(\tau_n|\alpha_n, \beta_n), \tag{4.25}$$

where $\mathbf{f}_{\neg d} \equiv \{\mathbf{f}_j\}_{j \neq d}$. Equation 4.22 can be implemented by sampling from the posterior GP over $\mathbf{X}_{:,d}$ with targets $\mathbf{t} = \mathbf{y} - \sum_{j \neq d} \mathbf{f}_j$ and noise σ_n^2, since

$$p(\mathbf{y}|\mathbf{f}_d, \mathbf{f}_{\neg d}, X, \theta) \propto \exp\left(-\frac{1}{2\sigma_n^2}\left(\mathbf{y} - \sum_{d=1}^{D} \mathbf{f}_d\right)^\top \left(\mathbf{y} - \sum_{d=1}^{D} \mathbf{f}_d\right)\right) \tag{4.26}$$

$$= \exp\left(-\frac{1}{2\sigma_n^2}(\mathbf{f}_d - \mathbf{t})^\top(\mathbf{f}_d - \mathbf{t})\right). \tag{4.27}$$

Somewhat unsurprisingly, this is similar to the central update in the Gauss-Seidel iteration used to compute the posterior means, as also noted in Hastie and Tibshirani [1998]. Ordinarily, sampling from a GP over N input locations is an $\mathcal{O}(N^3)$ operation with quadratic memory usage, however, using the SSM representation for a univariate GP, this operation can be run in $\mathcal{O}(N \log N)$ time and linear memory usage. The algorithm used to sample from the latent Markov chain in a SSM is known as the *forward-filtering, backward sampling* algorithm (FFBS), used by, for example Douc et al. [2009]. In FFBS, the forward filtering step is run much as in Algorithm 2 of Chapter 2. In the smoothing/backward recursion, instead of computing the smoothed density $p(\mathbf{z}_k|\mathbf{y}, \mathbf{X}_{:,d}, \theta_d)$, for $k = K, \ldots, 1$, we *sample* sequentially, in the backwards direction, using the conditionals $p(\mathbf{z}_k|\mathbf{z}_{k+1}^{\text{sample}}, \mathbf{y}, \mathbf{X}_{:,d}, \theta_d)$. The sampling is initialized by sampling from $p(\mathbf{z}_K|\mathbf{y}, \mathbf{X}_{:,d}, \theta_d)$, which is computed in the final step of the forward filtering run, to produce $\mathbf{z}_K^{\text{sample}}$. The FFBS algorithm, as applied to produce a sample from a univariate posterior GP (over training and test input locations), is outlined in detail in Algorithm 8. Note that Algorithm 8 generates a sample of the entire state vector. The sample of function values are obtained by reading out the first element of each of the $\mathbf{z}_k^{\text{sample}}$.

When sampling the hyperparameters, we make use of samples of $\{\mathbf{f}_d\}$, since these

Algorithm 8: Sampling a GP using FFBS

inputs : Jointly sorted training and test input locations $\mathbf{x}$. Targets $\mathbf{y}$ associated with training inputs. State transition function `stfunc` that returns $\boldsymbol{\Phi}$ *and* $\mathbf{Q}$ *matrices*. Hyperparameters θ.

outputs: Log-marginal likelihood $\log Z(\theta)$. Sample from the posterior: $\{\mathbf{z}_1^{\text{sample}}, \ldots, \mathbf{z}_N^{\text{sample}}\}$.

`//Forward Filtering as in Algorithm 2`

$\mathbf{z}_K^{\text{sample}} \sim \mathcal{N}(\boldsymbol{\mu}_K^{(f)}, V_K^{(f)})$;
for $k \leftarrow K-1$ **to** 1 **do**
    $\mathbf{L}_k \leftarrow \mathbf{V}_k \boldsymbol{\Phi}_k^\top \mathbf{P}_k^{-1}$;
    $\boldsymbol{\mu}_k \leftarrow \boldsymbol{\mu}_k^{(f)} + \mathbf{L}_t \left(\mathbf{z}_{k+1}^{\text{sample}} - \boldsymbol{\Phi}_t \boldsymbol{\mu}_t^{(f)} \right)$;
    $\mathbf{V}_k \leftarrow \mathbf{V}_k^{(f)} + \mathbf{L}_k \mathbf{P}_k \mathbf{L}_k^\top$;
    $\mathbf{z}_k^{\text{sample}} \sim \mathcal{N}(\boldsymbol{\mu}_k, V_k)$;
end

can be obtained by repeatedly using Equation 4.22 to cycle through sampling the additive components. Indeed, it is not necessary to have a "burn-in" period when obtaining these samples, since we can initialize this sampling process at the *mode* (i.e., mean) of the posterior over the $\mathbf{f}_d$ using Algorithm 7. This process is outlined in further detail in Algorithm 10. In order to sample from the conditional given in Equation 4.23 we have to resort to MCMC methods, as $p(\ell_d|\mu_\ell, v_\ell)$ is log-Gaussian and $p(\mathbf{f}_d|\mathbf{X}_{:,d}, \ell_d, \tau_d) = \mathcal{N}(\mathbf{0}, K_d(\theta_d))$ is a nonlinear function of ℓ_d. Note that this step of the sampling procedure is over a *scalar* space, so a simple MCMC algorithm such as Metropolis-Hastings works well. The sampling is performed over $\log(\ell_d)$, with a symmetric univariate Gaussian proposal density whose width is adjusted in a dataset-specific manner so that the rejection rate is close to 0.5.

In contrast to the lengthscale hyperparameter, we can derive the posteriors over τ_d and τ_n analytically. For τ_d we have:

$$p(\tau_d|\mathbf{f}_d, \mathbf{X}_{:,d}, \ell_d, \alpha_\tau, \beta_\tau) \propto \mathcal{N}\left(\mathbf{f}_d; \mathbf{0}, \tilde{\mathbf{K}}_d/\tau_d\right) \Gamma(\tau_d; \alpha_\tau, \beta_\tau)$$

$$
\begin{aligned}
&\propto \det(\tilde{\mathbf{K}}_d/\tau_d)^{-1/2} \exp\left(-\frac{1}{2}\mathbf{f}_d^\top(\tilde{\mathbf{K}}_d/\tau_d)^{-1}\mathbf{f}_d\right) \\
&\quad \times \qquad \tau_d^{\alpha_\tau - 1}\exp(-\beta_\tau \tau_d) \\
&\propto \Gamma(\alpha, \beta),
\end{aligned} \tag{4.28}
$$

where $\tilde{K}$ is k_d evaluated at the training inputs, with amplitude set to 1, and:

$$
\alpha = \alpha_\tau + \frac{N}{2}. \tag{4.29}
$$

$$
\beta = \beta_\tau + \frac{1}{2}\mathbf{f}_d^\top \tilde{\mathbf{K}}_d^{-1}\mathbf{f}_d. \tag{4.30}
$$

On the face of it, Equation 4.30 looks like an operation which causes the cubic matrix inversion operation to creep back into the MCMC scheme. However, the quantity $\mathbf{f}_d^\top \tilde{\mathbf{K}}_d^{-1}\mathbf{f}_d$ can also be calculated straightforwardly using the Kalman filter. In fact, it is equal to the standardised mean-squared error (SMSE) of the one-step-head predictions made by the GP (on the noise-free targets $\mathbf{f}_d$). This can be shown using the chain rule of probability:

$$
\begin{aligned}
\mathcal{N}(\mathbf{y}; \mathbf{0}, \mathbf{K}_N) &= \prod_{i=1}^{N} \mathcal{N}(y_i; m_i, v_i), \\
\text{where } m_i &= \mathbf{k}_i^\top \mathbf{K}_{i-1}^{-1}\mathbf{y}_{1:i-1}, \\
\text{and } v_i &= k_{i,i} - \mathbf{k}_i^\top \mathbf{K}_{i-1}^{-1}\mathbf{k}_i.
\end{aligned}
$$

Therefore we can write

$$
\left(\prod_{i=1}^{N}(2\pi v_i)^{-\frac{1}{2}}\right)\exp\left(-\frac{1}{2}\sum_{i=1}^{N}\frac{(y_i - m_i)^2}{v_i}\right) = (2\pi)^{-\frac{N}{2}}\det(\mathbf{K}_N)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}\mathbf{y}^\top\mathbf{K}_N^{-1}\mathbf{y}\right). \tag{4.31}
$$

As $\det(\mathbf{K}_N) = \prod_{i=1}^{N} v_i$, by repeated application of matrix identity A.4, it must be that:

$$
\frac{1}{2}\mathbf{y}^\top\mathbf{K}_N^{-1}\mathbf{y} = \frac{1}{2}\sum_{i=1}^{N}\frac{(y_i - m_i)^2}{v_i}. \tag{4.32}
$$

Since it involves one-step-ahead predictions, the SMSE for a GP can be calculated quite naturally using SSMs, as illustrated in Algorithm 9.

Algorithm 9: Efficient Computation of Standardised Squared Error using SSMs

inputs : Sorted training input locations $\mathbf{x}$. Noise-free function values $\mathbf{f}$ associated with training inputs. State transition function `stfunc` that returns $\mathbf{\Phi}$ *and* $\mathbf{Q}$ *matrices*. Hyperparameters θ.

outputs: Log-marginal likelihood $\log Z(\theta)$. Standardised Squared Error $\text{SSE}(\theta)$.

$\boldsymbol{\mu}_0^{(f)} \leftarrow \boldsymbol{\mu}$; $\mathbf{V}_0^{(f)} \leftarrow \mathbf{V}$; $\log Z(\theta) = 0$;
for $t \leftarrow 1$ **to** K **do**
  if $t > 1$ **then** $[\mathbf{\Phi}_{t-1}, \mathbf{Q}_{t-1}] \leftarrow$ `stfunc(`θ, $\mathbf{x}(t) - \mathbf{x}(t-1)$`)`;
  else $[\mathbf{\Phi}_{t-1}, \mathbf{Q}_{t-1}] \leftarrow$`stfunc(`$\theta$, ∞`);` `//Prior process covariance`
  $\mathbf{P}_{t-1} \leftarrow \mathbf{\Phi}_{t-1}\mathbf{V}_{t-1}^{(f)}\mathbf{\Phi}_{t-1}^{\top} + \mathbf{Q}_{t-1}$;
  $m \leftarrow \mathbf{H}\mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)}$;
  $v \leftarrow \mathbf{H}\mathbf{P}_{t-1}\mathbf{H}^{\top}$; `//Add jitter in practice`
  $\log Z(\theta) \leftarrow \log Z(\theta) - \frac{1}{2}\left(\log(2\pi) + log(v) + \frac{(f(t)-m)^2}{v}\right)$;
  $\text{SSE}(\theta) \leftarrow \text{SSE}(\theta) + \frac{(\mathbf{f}(t)-m)^2}{v}$;
  $\boldsymbol{\mu}_t^{(f)} = \mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)} + \mathbf{G}_t[f(t) - \mathbf{H}\mathbf{\Phi}_{t-1}\boldsymbol{\mu}_{t-1}^{(f)}]$;
  $\mathbf{V}_t^{(f)} = \mathbf{P}_{t-1} - \mathbf{G}_t\mathbf{H}\mathbf{P}_{t-1}$;
end

Similarly, the posterior over the noise precision, given by

$$p(\tau_n|\{\mathbf{f}_1\},\ldots,\{\mathbf{f}_D\},\mathbf{y},\alpha_n,\beta_n) = \Gamma(\alpha,\beta),$$

can be calculated analytically, as the Gamma distribution is conjugate to the precision of a Gaussian likelihood model (in this case, with zero mean). We thus obtain:

$$\alpha = \alpha_n + \frac{N}{2}. \tag{4.33}$$

$$\beta = \beta_n + \frac{1}{2}\left(\mathbf{y} - \sum_{d=1}^{D}\mathbf{f}_d\right)^\top \left(\mathbf{y} - \sum_{d=1}^{D}\mathbf{f}_d\right). \tag{4.34}$$

The pseudo-code for the full Gibbs Sampler is given in Algorithm 10.

Variational Bayesian EM

As an alternative to MCMC, we will consider a deterministic approximate inference scheme based on the EM algorithm. In particular, we will approximate the E-step using variational Bayes (VB). We first outline the generic algorithm for combining VB with the EM algorithm and then apply it to the graphical model in Figure 4.1. Notice that, in this case, we are viewing the hyperparameters θ as parameters that need to be optimised, as opposed to treating them as random variables, as was the choice made in the previous section.

The Generic VBEM Algorithm [▷▷] Consider a graphical model with observed variables $\mathbf{Y}$, latent variables $\mathbf{Z}$ and parameters θ. The overall aim is to approximate the posterior distribution $p(\mathbf{Z}|\mathbf{Y},\theta)$ and to optimise the marginal likelihood of the model w.r.t. the parameters, i.e., optimise $p(\mathbf{Y}|\theta)$ as a function of θ. The standard EM algorithm attacks the latter problem by making use of the following identity:

$$\log p(\mathbf{Y}|\theta) = \int q(\mathbf{Z}) \log p(\mathbf{Y}|\theta) d\mathbf{Z} \qquad \text{for any p.d.f. } q(\mathbf{Z}) \tag{4.35}$$

$$= \int q(\mathbf{Z}) \log \left(\frac{p(\mathbf{Y},\mathbf{Z}|\theta)}{q(\mathbf{Z})}\frac{q(\mathbf{Z})}{p(\mathbf{Z}|\mathbf{Y},\theta)}\right) d\mathbf{Z} \tag{4.36}$$

$$\equiv \mathcal{L}(q(\mathbf{Z});\theta) + \mathrm{KL}\left(q(\mathbf{Z})||p(\mathbf{Z}|\mathbf{Y},\theta)\right), \tag{4.37}$$

Algorithm 10: Additive GP Regression using Gibbs Sampling

inputs : Hyper-prior parameters : $\{\mu_\ell, v_\ell, \alpha_\tau, \beta_\tau, \alpha_n, \beta_n\}$
Training set : $\{\mathbf{X}, \mathbf{y}\}$, Test inputs : $\mathbf{X}_\star$
outputs: Hyperparameter samples : $\{\ell_d^{(s)}, \tau_d^{(s)}, \tau_n^{(s)}\}$
$\mathbb{E}(\mathbf{f}_\star|\mathbf{X}_\star, \theta)$ and $\mathbb{V}(\mathbf{f}_\star|\mathbf{X}_\star, \theta)$ (marginal variances)

Zero-mean the targets $\mathbf{y}$;
`//Sample hyperparameters from their prior:`
$\log(\ell_d) \sim \mathcal{N}(\mu_l, v_v)$; $\tau_d \sim \Gamma(\alpha_\tau, \beta_\tau)$; $\tau_n \sim \Gamma(\alpha_n, \beta_n)$; $d = 1, \ldots, D$;
for *each Gibbs sampling iteration* **do**
`//Sample` $\{\mathbf{f}_d\}$ `and` $\{\mathbf{f}_{\star d}\}$
`//Burn-in using classical backfitting:`
Initialise the $[\mathbf{f}_d^{\text{sample}}; \mathbf{f}_{\star d}^{\text{sample}}]$ (e.g. to $\mathbf{0}$);
while *The change in* $[\mathbf{f}_d^{sample}; \mathbf{f}_{\star d}^{sample}]$ *is above a threshold* **do**
for $d = 1, \ldots, D$ **do**
$[\mathbf{f}_d^{\text{sample}}; \mathbf{f}_{\star d}^{\text{sample}}] \leftarrow \mathbb{E}([\mathbf{f}_d; \mathbf{f}_{\star d}]|\mathbf{y} - \sum_{j \neq d} \mathbf{f}_j^{\text{sample}}, \mathbf{X}_{:,d}, \theta_d, \sigma_n^2)$;
end
end
for $d = 1, \ldots, D$ **do**
$\mathbf{t} \leftarrow \mathbf{y} - \sum_{j \neq d} \mathbf{f}_j^{\text{sample}}$;
$[\mathbf{f}_d^{\text{sample}}; \mathbf{f}_{\star d}^{\text{sample}}] \leftarrow$ `gpr_ffbs(`$[\mathbf{X}_{:,d}, \mathbf{X}_{\star d}]$, $\mathbf{t}$, θ_d`)`;
end
`//Sample hyperparameters:`
$\log(\ell_d^{(s)}) \leftarrow$ `Metropolis(`$\{\mathbf{f}_d\}$, $\mathbf{X}_d$, θ_d, μ_ℓ, v_ℓ`)`;
$\tau_d^{(s)} \sim \Gamma\left(\alpha_\tau + \frac{N}{2}, \beta_\tau + \frac{1}{2}\mathbf{f}_d^\top \tilde{\mathbf{K}}_d^{-1} \mathbf{f}_d\right)$;
$\tau_n^{(s)} \sim \Gamma\left(\alpha_n + \frac{N}{2}, \beta_n + \frac{1}{2}\left(\mathbf{y} - \sum_{d=1}^D \mathbf{f}_d\right)^\top \left(\mathbf{y} - \sum_{d=1}^D \mathbf{f}_d\right)\right)$;
Update estimates of $\mathbb{E}(\mathbf{f}_\star|\mathbf{X}_\star, \theta)$ and $\mathbb{V}(\mathbf{f}_\star|\mathbf{X}_\star, \theta)$ using $\{\mathbf{f}_{\star d}\}$;
end

where

$$\mathcal{L}(q(\mathbf{Z});\theta) = \int q(\mathbf{Z}) \log \left(\frac{p(\mathbf{Y},\mathbf{Z}|\theta)}{q(\mathbf{Z})} \right) d\mathbf{Z} \tag{4.38}$$

$$= \mathbb{E}_q \left(\log p(\mathbf{Y},\mathbf{Z}|\theta) \right) + H(q(\mathbf{Z})). \tag{4.39}$$

As KL divergences are always non-negative, $\mathcal{L}(q(\mathbf{Z});\theta)$ is a lower-bound to $\log p(\mathbf{Y}|\theta)$. Maximizing $\mathcal{L}(q(\mathbf{Z});\theta)$ with respect to $q(\mathbf{Z})$ (the E-step) is equivalent to minimizing $\mathrm{KL}\left(q(\mathbf{Z})||p(\mathbf{Z}|\mathbf{Y},\theta)\right)$, as made clear by Equation 4.37. Maximization w.r.t. θ (the M-step) is equivalent to optimizing the expected complete data log-likelihood (the first term in Equation 4.39), since $H(q(\mathbf{Z}))$ is independent of θ. Of course, standard EM assumes that the E-step can be performed optimally by simply setting $q(\mathbf{Z})$ to be equal to the posterior $p(\mathbf{Z}|\mathbf{Y},\theta)$ (i.e., when the KL divergences vanishes). For many models, including the additive GP, the assumption that the full posterior over latent variables can be computed tractably is clearly unrealistic.

The key idea behind variational Bayesian EM (VBEM) is to minimize the KL divergence between $q(\mathbf{Z})$ and $p(\mathbf{Z}|\mathbf{Y},\theta)$ subject to the constraint that $q(\mathbf{Z})$ belongs to a family of distributions, with some specific factorization properties. By constraining $q(\mathbf{Z})$ to have a particular factorization over K disjoint subsets of latent variables, i.e.,

$$q(\mathbf{Z}) = \prod_{i=1}^{K} q(\mathbf{Z}_i), \tag{4.40}$$

we can often attain a tractable minimization of the KL divergence, although the minimum we can obtain now is usually positive (since $q(\mathbf{Z})$ has limited flexibility). As we would like to minimize over *all* the factors $q(\mathbf{Z}_i)$, it is necessary to run an iterative algorithm where we optimize each of the individual factors one by one (while keeping the others fixed). By plugging Equation 4.40 into Equation 4.39 it can be seen that:

$$\mathcal{L}(q(\mathbf{Z});\theta) = \int q(\mathbf{Z}_j) \underbrace{\left[\int \log p(\mathbf{Y},\mathbf{Z}|\theta) \prod_{i \neq j} q(\mathbf{Z}_i) d\mathbf{Z}_i \right]}_{\equiv \mathbb{E}_{i \neq j}(\log p(\mathbf{Y},\mathbf{Z}|\theta))} - \sum_i \int q(\mathbf{Z}_i) \log q(\mathbf{Z}_i) d\mathbf{Z}_i. \tag{4.41}$$

When viewed as a function of an individual subset $\mathbf{Z}_j$, Equation 4.41 can be seen to

be a negative KL divergence between $q(\mathbf{Z}_j)$ and $\tilde{p}(\mathbf{Z}_j)$ where

$$\tilde{p}(\mathbf{Z}_j) \propto \exp\left(\mathbb{E}_{i\neq j}(\log p(\mathbf{Y}, \mathbf{Z}|\theta))\right), \tag{4.42}$$

and some constant terms which do not depend on $\mathbf{Z}_j$. Therefore, optimizing $\mathcal{L}$ w.r.t. an individual $q(\mathbf{Z}_j)$ amounts to simply equating it to $\tilde{p}(\mathbf{Z}_j)$, or, equivalently:

$$\log q_\star(\mathbf{Z}_j) = \mathbb{E}_{i\neq j}(\log p(\mathbf{Y}, \mathbf{Z}|\theta)) + \text{const.} \tag{4.43}$$

Equation 4.43 is the central update equation for the VB approximation to the E-step. Convergence to an optimum is *guaranteed* because the lower-bound is convex w.r.t. each of the $q(\mathbf{Z}_i)$, see Bishop [2007].

Once the optimization for the E-step converges, the M-step updates the free parameters θ, while keeping the $q(\mathbf{Z}_i)$ fixed, much like standard EM. The M-step is also typically easier than in standard EM because the factorization of $q(\mathbf{Z})$ often simplifies the optimization of $\mathbb{E}_q\left(\log p(\mathbf{Y}, \mathbf{Z}|\theta)\right)$.

Additive GP Regression using VBEM We now apply the VBEM algorithm to the additive GP model. In this case, the observed variables are the targets $\mathbf{y}$, the latent variables $\mathbf{Z}$ consist of the D Markov chains as shown in Figure 4.3:

$$\mathbf{Z} \equiv \left(\underbrace{\mathbf{z}_1^1, \ldots, \mathbf{z}_1^N}_{\equiv \mathbf{Z}_1}, \underbrace{\mathbf{z}_2^1, \ldots, \mathbf{z}_2^N}_{\equiv \mathbf{Z}_2}, \ldots, \underbrace{\mathbf{z}_D^1, \ldots, \mathbf{z}_D^N}_{\equiv \mathbf{Z}_D}\right). \tag{4.44}$$

Notice that we have made the SSM representation for the univariate regression models explicit in this case. Indeed, the graphical model in Figure 4.3 is a more detailed illustration of all the variables involved for an efficient implementation of the additive GP model in general. In previous sections it was sufficient to keep track of the distribution over the latent function values only, however, for a tractable VBEM implementation it will be necessary to explicitly incorporate all the latent variables of the SSMs involved. Note also that we could have placed a prior over θ, much like in the MCMC setup, and then absorb θ into $\mathbf{Z}$. Indeed, this would be the full VB solution to additive GP regression. However, the updates for the hyperparameters (e.g., the lengthscales) cannot be computed analytically, using Equation 4.43. Thus,

for the sake of simplicity we stick with a VBEM-style approach and optimize θ w.r.t. the marginal likelihood.

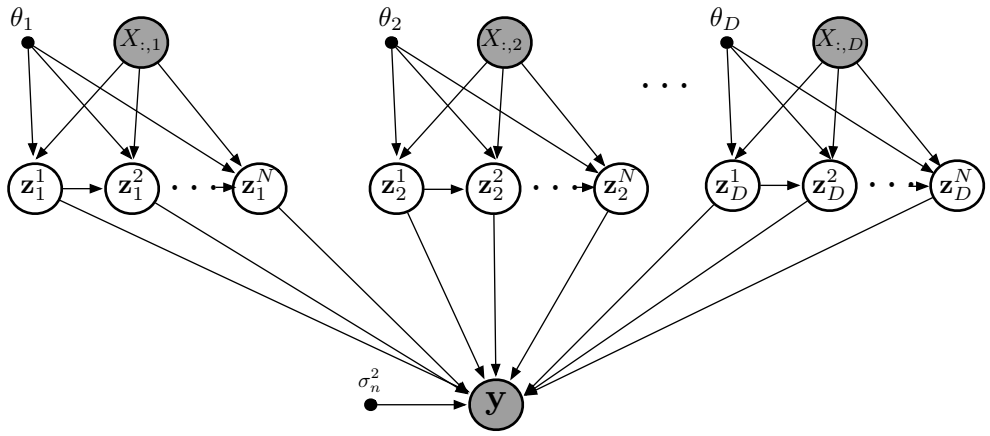


Figure 4.3: Graphical Model of Additive Regression, where we have made explicit the fact that each univariate GP model is implemented as a SSM. For each input location $\mathbf{x}$, we have D SSM states corresponding to the $\{x_d\}_{d=1}^D$. Each of these states emits an observation which contributes to the noise-free function value. We run VBEM on this graph to perform inference over the SSM states and optimise the values of $\{\theta_d\}_{d=1}^D$.

Referring to the definition in 4.44, the true posterior $p(\mathbf{Z}_1, \ldots, \mathbf{Z}_D|\mathbf{y}, \mathbf{X}, \theta)$ is hard to handle computationally because although the $\mathbf{Z}_i$ are independent a priori, once we condition on $\mathbf{y}$ they all become coupled. As a result, computing the expected sufficient statistics associated with $p(\mathbf{Z}_i|\mathbf{y}, \mathbf{X}, \theta)$, crucial to forming predictive distributions and learning θ, requires evaluation of $\int p(\mathbf{Z}_1, \ldots, \mathbf{Z}_D|\mathbf{y}, \mathbf{X}, \theta) d\mathbf{Z}_{\neg i}$. This is clearly intractable for large N. Thus we will consider an approximation to the true posterior which *does* factorise across the $\mathbf{Z}_i$, i.e.:

$$q(\mathbf{Z}) = \prod_{i=1}^{D} q(\mathbf{Z}_i). \tag{4.45}$$

In order to see what impact using such an approximation has in the context of using VBEM for the additive GP, we use Equations 4.45 and 4.43 to derive the iterative updates required. We first write down the log of the joint distribution over

all variables, given by:

$$\log(p(\mathbf{y}, \mathbf{Z}|\theta)) = \sum_{n=1}^{N} \log p\left(y_n|\mathbf{H}\sum_{d=1}^{D}\mathbf{z}_d^{t_d(n)}, \sigma_n^2\right) + \sum_{d=1}^{D}\sum_{t=1}^{N}\log p(\mathbf{z}_d^t|\mathbf{z}_d^{t-1}, \theta_d), \quad (4.46)$$

where we have defined $p(\mathbf{z}_d^t|\mathbf{z}_d^{t-1}, \theta_d) \equiv p(\mathbf{z}_d^1|\theta_d)$, for $t = 1$, and $\mathbf{Hz}$ gives the first element of $\mathbf{z}$ (as in Chapter 2). Note that it is also necessary to define the mapping $t_d(\cdot)$ which gives, for each dimension d, the SSM index associated with y_n. The index t iterates over the *sorted* input locations along axis d. Thus:

$$\begin{aligned}\log q_\star(\mathbf{Z}_j) &= \mathbb{E}_{i\neq j}(\log p(\mathbf{Y}, \mathbf{Z}|\theta)) + \text{const} \quad (4.47)\\ &= -\frac{1}{2\sigma_n^2}\sum_{i=n}^{N}\mathbb{E}_{i\neq j}\left[\left(\mathbf{Hz}_j^{t_j(n)} - \left(y_n - \sum_{i\neq j}\mathbf{Hz}_i^{t_d(n)}\right)\right)^2\right]\\ &+ \sum_{t=1}^{N}\log p(\mathbf{z}_j^t|\mathbf{z}_j^{t-1}, \theta_j) + \text{const.} \quad (4.48)\end{aligned}$$

where we have absorbed (almost) everything which does not depend on $\mathbf{Z}_j$ into the constant term. Note that $\mathbb{E}_{i\neq j}\left[\left(y_n - \sum_{i\neq j}\mathbf{Hz}_i^{t_d(n)}\right)^2\right]$ is also independent of $\mathbf{Z}_j$. By completing the square we can therefore write:

$$\begin{aligned}\log q_\star(\mathbf{Z}_j) &= \sum_{i=n}^{N}\log\mathcal{N}\left(\left(y_n - \sum_{i\neq j}\mathbf{H}\mathbb{E}\left[\mathbf{z}_i^{t_d(n)}\right]\right); \mathbf{Hz}_j^{t_j(n)}, \sigma_n^2\right)\\ &+ \sum_{t=1}^{N}\log p(\mathbf{z}_j^t|\mathbf{z}_j^{t-1}, \theta_j) + \text{const.} \quad (4.49)\end{aligned}$$

where

$$\mathbb{E}\left[\mathbf{z}_i^k\right] = \int \mathbf{z}_i^k q(\mathbf{Z}_i) d\mathbf{Z}_i. \quad (4.50)$$

In other words, in order to update the factor $q(\mathbf{Z}_j)$ it is sufficient to run the standard SSM inference procedure using the observations: $\left(y_n - \sum_{i\neq j}\mathbf{H}\mathbb{E}\left[\mathbf{z}_i^{t_d(n)}\right]\right)$. This is similar in nature to the classical backfitting update, and illustrates a novel connection between approximate Bayesian inference for additive models and classical estimation techniques. A number of conclusions can be drawn from this connection.

First, since VB iterations are guaranteed to converge, any moment computed using the factors $q(\mathbf{Z}_i)$ is also guaranteed to converge. Notice that convergence of these moments is important because they are used to learn the hyperparameters (see below). Second, since the *true* posterior $p(\mathbf{Z}_1, \ldots, \mathbf{Z}_D|\mathbf{y}, \theta)$ is a large joint Gaussian over all the latent variables we would expect $\mathbb{E}_q(\mathbf{Z})$ to be *equal* to the true posterior mean. This is because the Gaussian is unimodal and the VB approximation is *mode-seeking* Minka [2005]. This provides another proof of why backfitting computes the posterior mean over latent function values exactly and extends the result to the posterior mean over derivatives of the function values as well.

For the M-step of the VBEM algorithm it is necessary to optimise $\mathbb{E}_q\left(\log p(\mathbf{y}, \mathbf{Z}|\theta)\right)$. Using Equation 4.46 it is easy to show that the expected sufficient statistics required to compute derivatives w.r.t. θ are the set of expected sufficient statistics for the SSMs associated with each individual dimension. This is clearly another major advantage of using the factorised approximation to the posterior. Thus, for every dimension d, we use the Kalman filter and RTS smoother to compute $\left\{\mathbb{E}_{q(\mathbf{Z}_d)}(\mathbf{z}_d^n)\right\}_{n=1}^N$, $\left\{\mathbb{V}_{q(\mathbf{Z}_d)}(\mathbf{z}_d^n)\right\}_{n=1}^N$ and $\left\{\mathbb{E}_{q(\mathbf{Z}_d)}(\mathbf{z}_d^n\mathbf{z}_d^{n+1})\right\}_{n=1}^{N-1}$. We then use these expected statistics to compute derivatives of the expected complete data log-likelihood w.r.t. θ and use a standard minimizer (such as `minimize.m`) to complete the M step. The overall VBEM algorithm for training an additive GP is given in Algorithm 11.

4.3 Efficient Projected Additive GP Regression

So far, we have shown how the assumption of additivity can be exploited to derive non-sparse GP regression algorithms which scale as $\mathcal{O}(N \log N)$. The price to pay for such an improvement in complexity becomes apparent when applying the additive GP model to many regression tasks – the assumption of additivity often causes a decrease in accuracy and predictive power. We will see several illustrations of this phenomenon in Section 4.5.

Interestingly, it is actually possible to relax the assumption of additivity in the original space of covariates *without* sacrificing the $\mathcal{O}(N \log N)$ scaling, by simply considering an additive GP regression model in a feature space linearly related to original space of covariates. The graphical model illustrating this idea is given in Figure 4.4. We will restrict attention to the following generative process, which we

Algorithm 11: Additive GP Regression using VBEM

inputs : Initial values for the hyperparameters θ_0
Training set : $\{\mathbf{X}, \mathbf{y}\}$, Test inputs : $\mathbf{X}_\star$
outputs: Learned hyperparameters θ
$\mathbb{E}(\mathbf{f}_\star|\mathbf{X}_\star, \theta)$ and $\mathbb{V}(\mathbf{f}_\star|\mathbf{X}_\star, \theta)$ (test means and variances)

Zero-mean the targets $\mathbf{y}$; $\theta \leftarrow \theta_0$;
for *each VBEM iteration* **do**
    //E-step
    Initialise $q(\mathbf{Z}_d)$ such that $\mathbb{E}(\mathbf{Z}_d) = \mathbf{0}$ for $d = 1, \ldots, D$;
    while *The change in* $\{\mathbb{E}(q(\mathbf{Z}_d))\}_{d=1}^D$ *is above a threshold* **do**
        $//ESS_d \equiv \left[\{\mathbb{E}(\mathbf{z}_d^n)\}_{n=1}^N, \{\mathbb{V}(\mathbf{z}_d^n)\}_{n=1}^N, \left\{\mathbb{E}(\mathbf{z}_d^n \mathbf{z}_d^{n+1})\right\}_{n=1}^{N-1}\right]$
        $//\mathbf{y}_d \equiv \left\{y_n - \sum_{i \neq j} H\mathbb{E}\left[\mathbf{z}_i^{t_d(n)}\right]\right\}_{n=1}^N$
        for $d = 1, \ldots, D$ **do**
            $\left[ESS_d, \boldsymbol{\mu}_\star^d, \mathbf{v}_\star^d\right] \leftarrow$ `gpr_ssm(`$\mathbf{y}_d$, $[\mathbf{X}_{:,d}, \mathbf{X}_{\star d}]$, θ_d`);` `//See Alg.` 2
        end
    end
    //M-step
    Optimise θ using $\{ESS_d\}_{d=1}^D$; `//See Alg.` 3
end
$\mathbb{E}(\mathbf{f}_\star|\mathbf{X}_\star, \theta) \leftarrow \sum_{d=1}^D \boldsymbol{\mu}_\star^d$;
$\mathbb{V}(\mathbf{f}_\star|\mathbf{X}_\star, \theta) \leftarrow \sum_{d=1}^D \mathbf{v}_\star^d$;

will refer to as the *projected additive Gaussian process* (PAGP) prior:

$$y_i = \sum_{m=1}^{M} \mathbf{f}_m(\boldsymbol{\phi}_m(i)) + \epsilon \qquad i = 1, \ldots, N, \tag{4.51}$$

$$\boldsymbol{\phi}_m = \mathbf{w}_m^\top \mathbf{X} \qquad m = 1, \ldots, M, \tag{4.52}$$

$$\mathbf{f}_m(\cdot) \sim \mathcal{GP}\left(\mathbf{0}; k_m(\boldsymbol{\phi}_m, \boldsymbol{\phi}_m'; \theta_m)\right) \quad m = 1, \ldots, M, \tag{4.53}$$

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2).$$

Notice that the number of projections, M, can be less or greater than D. These cases are referred to as *undercomplete* and *overcomplete* PAGPs. Forming linear combinations of the inputs before feeding them into an additive GP model significantly enriches the flexibility of the functions supported by the prior above. It is straightforward to come up with example functions which would have support under the PAGP prior and not under the vanilla additive model: consider the function $(x_1 + x_2 + x_3)^3$. This function includes many terms which are formed by taking *products* of covariates, and thus can capture relationships where the covariates *jointly* affect the target variable. In fact, Equations 4.51 through to 4.53 are identical to the standard neural network model where the nonlinear activation functions (usually set in advance to be, for example, the sigmoid function) are modelled using GPs. Recall that, for neural networks, taking M to be arbitrarily large results in the ability to be able to approximate *any* continuous function in $\mathbb{R}^D$. This property certainly extends to PAGPs, since we are also learning the activation functions from data.

4.3.1 Inference and Learning

Naturally, learning and inference for the PAGP model can be implemented straightforwardly using the standard GP regression algorithm presented in Chapter 1. For example, assuming that $k_m(\cdot, \cdot)$ is set to be the squared-exponential kernel, we can simply define the covariance between two inputs to be:

$$k(\mathbf{x}, \mathbf{x}'; \theta, W) = \sum_{m=1}^{M} v_m \exp\left(-\frac{(\mathbf{w}_m^\top \mathbf{x} - \mathbf{w}_m^\top \mathbf{x}')^2}{2\ell_m^2}\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}'), \tag{4.54}$$

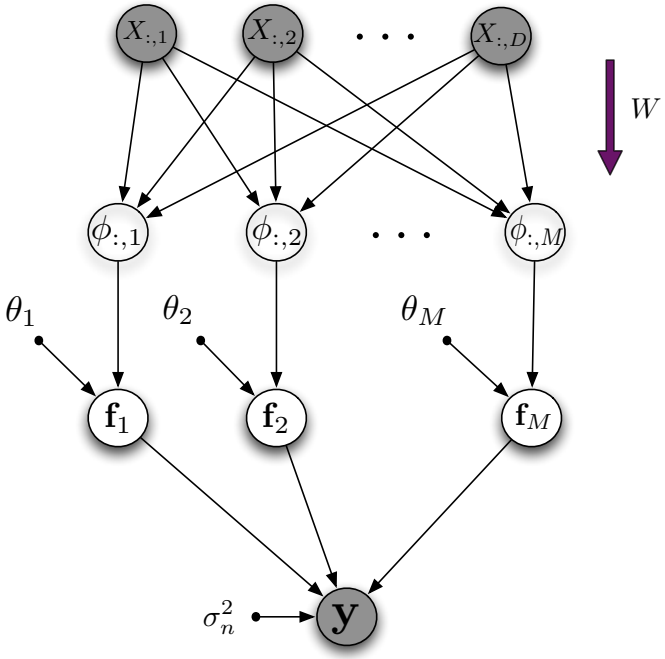


Figure 4.4: Graphical model for Projected Additive GP Regression. In general, $M \neq D$. We present a greedy algorithm to select M, and jointly optimise W and $\{\theta_d\}_{d=1}^{D}$.

where we have defined $W \equiv [\mathbf{w}_1, \ldots, \mathbf{w}_M]$ and $\theta \equiv [v_1, \ldots, v_M, \ell_1, \ldots, \ell_M, \sigma_n^2]$. Then, it is possible to jointly optimise the marginal likelihood w.r.t. θ and W. Note that no constraints are placed on the structure of the linear projection matrix W – it is simply optimised element-wise. In addition, a search is required to find which M is optimal for the problem at hand, the search terminating when a maximum quantity is reached.

Such an approach obviously suffers from the computational burden we are trying to avoid. In fact, it actually worsens it by also adding a search for a good value of M. Conceptually, the simplest way to improve efficiency is to treat the weight matrix W as part of the set of hyperparameters θ and use MCMC or VBEM, as was done in Section 4.2. When using MCMC one would need to add a prior over W, and select a suitable M using reversible-jump MCMC, as described in Green [1995]. When using VBEM, one would need to consider how to learn W using the appropriate (approximate) expected sufficient statistics. In this section, we will not consider these options (although they are perfectly valid choices). Instead, we will

focus on an elegant greedy learning strategy which is similar to another classical nonparametric regression algorithm known as *projection pursuit* regression. We will refer to it as projection-pursuit GP regression (PPGPR).

Consider the case where $M = 1$. Following the running example[1], the covariance matrix in Equation 4.54 becomes

$$k(\mathbf{x}, \mathbf{x}'; \theta, \mathbf{w}) = v \exp\left(-\frac{(\mathbf{w}^\top \mathbf{x} - \mathbf{w}^\top \mathbf{x}')^2}{2\ell^2}\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}'), \tag{4.55}$$

and the resulting PAGP regression model reduces to a *scalar* GP. Recall from Chapter 2 that, for a kernel that can be represented as a SSM, we can use the EM algorithm to optimise θ w.r.t. the marginal likelihood efficiently in Equation 4.55, for some fixed $\mathbf{w}$. It is possible to extend this idea and jointly optimise $\mathbf{w}$ and θ w.r.t. the marginal likelihood, although in this case we opt to optimise the marginal likelihood *directly* without the simplification of using the expected complete data log-likelihood, as is done in the EM algorithm. Notice that every step of this optimisation scales as $\mathcal{O}(N \log N)$, since at every step we need to compute the marginal likelihood of a scalar GP (and its derivatives). These quantities are computed using the Kalman filter by additionally (and laboriously) differentiating the Kalman filtering process w.r.t. $\mathbf{w}$ and θ. In the interest of reducing clutter, we present the details of this routine in Appendix B. In short, all that is required is the derivatives of the state transition and process covariance matrices ($\mathbf{\Phi}_t$ and $\mathbf{Q}_t$, for $t = 1, \ldots, N-1$) w.r.t. $\mathbf{w}$ and θ. In Algorithm 12 below, which gives pseudo-code for PPGPR, we will refer to the core Kalman filter-based routine as `ppgpr_1D`. Note how the greedy nature of the algorithm allows the learning of the dimensionality of the feature space, M, rather naturally – one keeps on adding new feature dimensions until there is no significant change in validation set performance (e.g., normalised mean-squared error). One important issue which arises in Algorithm 12 involves the initialisation of $\mathbf{w}^m$ at step m. Several alternatives come to mind, however we will limit ourselves to two. The first is random initialization where we sample each $\mathbf{w}_m(i)$ from a standard Gaussian, the second is where we initialise the weights as those obtained from a linear regression of X onto the target/residual vector $\mathbf{y}^m$. The initialisation of the

[1]Note that although we will introduce the concepts of PPGPR using the squared-exponential kernel, in practice we will always be using the Matérn kernel as the core univariate GP model.

other hyperparameters follow the usual "sensible" initialisation schemes present in the literature, reiterated in Section 4.5. Note that PPGPR is used for the purposes of learning W and θ only. Once this is complete, one can simply run the VBEM algorithm (without the M-step) to compute predictions.

Algorithm 12: Projection Pursuit GP Regression

inputs : Training set : $\{\mathbf{X}, \mathbf{y}\}$, Validation set : $\{\mathbf{X}_{\surd}, \mathbf{y}_{\surd}\}$
Test set : $\{\mathbf{X}_{\star}, \mathbf{y}_{\star}\}$
outputs: Learned hyperparameters $\boldsymbol{\theta}$, Projection W, M
$\mathbb{E}(\mathbf{f}_{\star}|\mathbf{X}_{\star}, \theta)$ and $\mathbb{V}(\mathbf{f}_{\star}|\mathbf{X}_{\star}, \theta)$ (test means and variances)

Zero-mean the targets $\mathbf{y}$;
$m \leftarrow 1$;
$\mathbf{y}^m \leftarrow \mathbf{y}$;
while *`validation_error` is above a threshold* **do**
    $\mathbf{w}_m^{(init)} \leftarrow$ `init_weights`; $\boldsymbol{\theta}_m^{(init)} \leftarrow$ `init_hypers` ; `// See text.`
    $[\mathbf{w}_m, \boldsymbol{\theta}_m] \leftarrow$ Optimise using `ppgpr_1D(`$\mathbf{w}_m^{(init)}$, $\boldsymbol{\theta}_m^{(init)}$, $\mathbf{X}$, $\mathbf{y}$`)`;
    $\left[\boldsymbol{\mu}^m, \boldsymbol{\mu}_{\surd}^m\right] \leftarrow$ `gpr_ssm(`$[\mathbf{y}, \mathbf{y}_{\surd}]$, $[\mathbf{X}, \mathbf{X}_{\surd}]$, $\boldsymbol{\theta}_m$, $\mathbf{w}_m$`)`;
    $\mathbf{y}^m \leftarrow \mathbf{y}^m - \boldsymbol{\mu}^m$; `//Form residual.`
    $\boldsymbol{\mu}_{\surd} \leftarrow \boldsymbol{\mu}_{\surd} + \boldsymbol{\mu}_{\surd}^m$;
    Update `validation_error` using $\boldsymbol{\mu}_{\surd}$ and $\mathbf{y}_{\surd}$;
    Add $\mathbf{w}_m$ to W; Add $\boldsymbol{\theta}_m$ to $\boldsymbol{\theta}$;
    $m \leftarrow m + 1$;
end
$\Phi \leftarrow W^\top \mathbf{X}$; $\Phi_{\star} \leftarrow W^\top \mathbf{X}_{\star}$;
Run Algorithm 10 with hypers fixed at $\boldsymbol{\theta}$ to compute $\mathbb{E}(\mathbf{f}_{\star})$ and $\mathbb{V}(\mathbf{f}_{\star})$;

4.4 Generalised Additive GP Regression

In Section 2.4 we extended the efficient state-space model implementation of GP regression to non-Gaussian likelihoods, in order to be able to handle tasks where the observed targets exhibit a non-Gaussian relationship to the latent GP. Of course, this was limited to problems where the input space was constrained to be scalar. The overwhelming majority of real-world generalised regression tasks (such as classification) use more than one feature to be able to model and predict the labels $\mathbf{y}$. In this Section, we demonstrate that for GP kernels with additive structure that

enables $\mathcal{O}(N \log N)$ scaling in the standard regression setting, it is possible to derive an $\mathcal{O}(N \log N)$ algorithm which performs MAP inference and hyperparameter learning using Laplace's approximation.

Recall that Laplace's approximation is based on Newton's method on the objective:

$$\Omega(\mathbf{f}) \equiv \log p(\mathbf{y}|\mathbf{f}) + \log p(\mathbf{f}|\mathbf{X}, \theta). \tag{4.56}$$

Let's now additionally assume that $\mathbf{f}$ is drawn from an additive GP. It is straighforward to show that:

$$\begin{aligned}
\nabla\Omega(\mathbf{f}) &= \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f}) - \mathbf{K}_{\text{add}}^{-1}\mathbf{f}, && (4.57)\\
\nabla\nabla\Omega(\mathbf{f}) &= \underbrace{\nabla\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})}_{\equiv -W} - \mathbf{K}_{\text{add}}^{-1}, && (4.58)\\
\mathbf{f}^{(k+1)} &\leftarrow \mathbf{f}^{(k)} + \left(\mathbf{K}_{\text{add}}^{-1} + W\right)^{-1} \left(\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}} - \mathbf{K}_{\text{add}}^{-1}\mathbf{f}^{(k)}\right) && (4.59)\\
&= \mathbf{K}_{\text{add}} \left(\mathbf{K}_{\text{add}} + W^{-1}\right)^{-1} \left[\mathbf{f}^{(k)} + W^{-1}\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}}\right]. && (4.60)
\end{aligned}$$

Note that since $p(\mathbf{y}|\mathbf{f}) = \prod_{n=1}^{N} p(y_i|f_i)$, $\mathbf{W}$ is a diagonal matrix and is easy to invert. Looking closer at Equation 4.60, we see that it is *precisely* the same as the expression to compute the posterior mean of a GP, where the target vector is given by $\left[\mathbf{f}^{(k)} + \mathbf{W}^{-1}\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}}\right]$ and where the diagonal "noise" term is given by $\mathbf{W}^{-1}$. Given an additive kernel corresponding to a sum of scalar GPs that can be represented using SSMs, we can therefore use Algorithm 7 to implement a single iteration of Newton's method! As a result, it is possible to compute $\hat{\mathbf{f}}$ in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space, since in practice only a handful of Newton iterations are required for convergence (as it is a second order optimisation method). Wrapping backfitting iterations inside a global Newton iteration is precisely how the local-scoring algorithm is run to fit a generalised additive model. Thus, we can view the development in this section as a novel Bayesian reinterpretation of local scoring.

We can also efficiently approximate the marginal likelihood using the Taylor expansion of the objective function $\Omega(\mathbf{F})$, as in Equation 1.54, although we will need to express it explicitly in terms of $\mathbf{F} \equiv [\mathbf{f}_1; \ldots; \mathbf{f}_D]$, as opposed to the sum $\mathbf{f}$.

$$\Omega(\mathbf{F}) = \log p(\mathbf{y}|\mathbf{F}) + \log p(\mathbf{F}|\mathbf{X}, \theta). \tag{4.61}$$

Note that the run which computes $\hat{\mathbf{f}}$ does so via the computation of $\hat{\mathbf{F}}$, so we obtain it for free after running Algorithm 7. Once $\hat{\mathbf{F}}$ is known, it can be used to compute the approximation to the marginal likelihood. Using Equation 1.55 we obtain:

$$\log p(\mathbf{y}|\mathbf{X}) \approx \Omega(\hat{\mathbf{F}}) - \frac{1}{2}\log\det\left(\tilde{\mathbf{W}} + \tilde{\mathbf{K}}^{-1}\right) + \frac{ND}{2}\log(2\pi) \tag{4.62}$$

$$= \log p(\mathbf{y}|\hat{\mathbf{F}}) - \frac{1}{2}\hat{\mathbf{F}}^{\top}\tilde{\mathbf{K}}^{-1}\hat{\mathbf{F}} - \frac{1}{2}\log\det\left(\tilde{\mathbf{K}} + \tilde{\mathbf{W}}^{-1}\right) - \frac{1}{2}\log\det(\tilde{\mathbf{W}}), \tag{4.63}$$

where we have defined:

$$\tilde{\mathbf{K}} = \begin{bmatrix} \mathbf{K}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_2 & \dots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{K}_D \end{bmatrix}, \qquad \tilde{\mathbf{W}} = \begin{bmatrix} \mathbf{W} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{W} & \dots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{W} \end{bmatrix}.$$

and have used the matrix determinant lemma (see Appendix A) to get from 4.62 to 4.63. Notice how every term in the marginal likelihood approximation can be computed in $\mathcal{O}(N \log N)$ complexity. This is trivially true for the terms $\log p(\mathbf{y}|\hat{\mathbf{F}})$ and $\frac{1}{2}\log\det(\tilde{\mathbf{W}})$. The term $\frac{1}{2}\hat{\mathbf{F}}^{\top}\tilde{\mathbf{K}}^{-1}\hat{\mathbf{F}}$ is the sum of SMSEs for D scalar GPs with noise-free targets $\hat{\mathbf{f}}_d$ (see Section 4.2.2). The term $\frac{1}{2}\log\det\left(\tilde{\mathbf{K}} + \tilde{\mathbf{W}}^{-1}\right)$ can be computed by summing the log predictive variances of D SSMs with diagonal noise matrices given by $\mathbf{W}^{-1}$ (for *any* target vector – see Equation 4.31). In Algorithm 13 we give the pseudo-code illustrating these ideas.

4.5 Results

4.5.1 Experiments on Regression Tasks

4.5.1.1 Performance Metrics

We will be concerned with three performance measures when comparing the additive GP regression algorithms introduced in this chapter, both to each other and to other algorithms introduced in Section 4.5.1.2. These measures have been chosen to be consistent with those commonly used in the sparse GP regression literature, in order

Algorithm 13: Generalised Additive GPR using Laplace's Approximation

inputs : Hyperparameters θ, Likelihood function : `likelihood` which computes log-likelihood of targets (λ) and its first (∇) and second ($-\mathbf{W}$) derivatives.
Training set : $\{\mathbf{X}, \mathbf{y}\}$, Test inputs : $\mathbf{X}_\star$

outputs: Predictive means : $\hat{\mathbf{f}}_\star$, Predictive probabilities : $\hat{\mathbf{p}}$, $\hat{\mathbf{p}}_\star$
Approximation of the log-marginal likelihood $\log Z(\theta)$.

$\mathbf{f}_d \leftarrow \mathbf{0}$ for $d = 1, \ldots, D$; $\mathbf{f} \leftarrow \sum_{d=1}^{D} \mathbf{f}_d$;
while *the change in* $\mathbf{f}$ *is above a threshold* **do**
    `// Newton iterations`
    $[\lambda,\ \nabla,\ \mathbf{W}] \leftarrow$ `likelihood(`$\mathbf{f}$, $\mathbf{y}$`)`;
    $\mathbf{z} \leftarrow \mathbf{f} + \mathbf{W}^{-1}\nabla$;
    while *the change in* $\mathbf{f}_d$ *is above a threshold* **do**
        `// Backfitting iterations`
        for $d = 1, \ldots, D$ **do**
            $\mathbf{z}_d \leftarrow \mathbf{z} - \sum_{j \neq d} \mathbf{f}_d$;
            $[\mathbf{f}_d, \mathbf{f}_{\star d}] \leftarrow$ `gpr_ssm(`$\mathbf{z}_d$, $\mathbf{W}^{-1}$, $[\mathbf{X}_{:,d}, \mathbf{X}_{\star d}]$, θ_d`)`; `//Diagonal noise`
        end
    end
    $\mathbf{f} \leftarrow \sum_{d=1}^{D} \mathbf{f}_d$; $\mathbf{f}_\star \leftarrow \sum_{d=1}^{D} \mathbf{f}_{\star d}$;
end
$\hat{\mathbf{f}} \leftarrow \mathbf{f}$; $\hat{\mathbf{f}_\star} \leftarrow \mathbf{f}_\star$;
$\hat{\mathbf{p}} \leftarrow$`likelihood(`$\hat{\mathbf{f}}$`)`; $\hat{\mathbf{p}}_\star \leftarrow$`likelihood(`$\hat{\mathbf{f}}_\star$`)`;
`// Marginal likelihood approximation`
$[\lambda,\ \nabla,\ \mathbf{W}] \leftarrow$ `likelihood(`$\hat{\mathbf{f}}$, $\mathbf{y}$`)`;
$Z(\theta) \leftarrow \lambda - \frac{1}{2}\log\det(\mathbf{W})$;
for $d = 1, \ldots, D$ **do**
    $smse_d \leftarrow$ `gpr_smse(`$\hat{\mathbf{f}}_d$, $\mathbf{0}$, $\mathbf{X}_{:,d}$, θ_d`)`;
    $logdet_d \leftarrow$ `gpr_logdet(`$\hat{\mathbf{f}}_d$, $\mathbf{W}^{-1}$, $\mathbf{X}_{:,d}$, θ_d`)`;
    $Z(\theta) \leftarrow Z(\theta) - \frac{1}{2}smse_d - \frac{1}{2}logdet_d$;
end

to ease comparison with other works.

- Runtime, measured in seconds. Fluctuations in all quantities presented pertaining to runtimes will be reduced by performing 10 independent runs.

- The test-set normalised mean square error (NMSE) :

$$\text{NMSE} = \frac{\sum_{i=1}^{N_\star}(\mathbf{y}_\star(i) - \boldsymbol{\mu}_\star(i))^2}{\sum_{i=1}^{N_\star}(\mathbf{y}_\star(i) - \bar{y})^2}, \tag{4.64}$$

  where $\boldsymbol{\mu}_\star \equiv \mathbb{E}(\mathbf{f}_\star|X, \mathbf{y}, X_\star, \theta)$ and $\bar{y}$ is the *training-set* average target value (which will always be zero in our case). This measure only assesses the quality of the posterior mean predictions and is suitable to cases where we only need mean predictions for the task at hand. However, for many real-world problems, error bars in the predictions are also very useful. The quality of error bars is measured using the metric below.

- The test-set Mean Negative Log Probability (MNLP) :

$$\text{MNLP} = \frac{1}{2N_\star}\sum_{i=1}^{N_\star}\left[\frac{(\mathbf{y}_\star(i) - \boldsymbol{\mu}_\star(i))^2}{\mathbf{v}_\star(i)} + \log \mathbf{v}_\star(i) + \log 2\pi\right], \tag{4.65}$$

  where $\mathbf{v}_\star \equiv \mathbb{V}(\mathbf{f}_\star|X, \mathbf{y}, X_\star, \theta)$. This metric correctly captures the intuitive notion of penalising overconfident predictions which are off, and underconfident predictions which are actually good. For both the NMSE and MNLP measures lower values indicate better performance.

4.5.1.2 Comparison Methodology

For each experiment presented, we will compare the performance, as measured by the three metrics above, of either all or a subset of algorithms listed in this section. If a particular algorithm has a stochastic component to it (e.g., if it involves MCMC) its performance will be averaged over 10 runs.

Additive-MCMC : This is the MCMC algorithm presented in Section 4.2.2. We will perform Gibbs sampling by cycling through Equations 4.22 to 4.25 100 times. Recall that since we run the backfitting algorithm within in each Gibbs sampling iteration a large majority of the random variables involved can be "burned-in"

rapidly to match the corresponding conditional *mode*. For all experiments involved, this reduces the number of Gibbs sampling iterations required to get satisfactory results. Referring to Equations 4.17 through 4.21 we set up the hyper-prior parameters as follows:

$$\mu_\ell = 0, v_\ell = 9; \alpha_\tau = 0.1, \beta_\tau = 0.1; \alpha_n = 1, \beta_n = 0.1.$$

When sampling hyperparameters, recall that we use 1-dimensional MH sampling for the lengthscales and that for all others we sample directly from the appropriate posterior. The MH step size is adjusted in a dataset-specific manner. The state-transition function used in the underlying SSMs is always set to be the one corresponding to the Matérn(7/2) kernel. This provides a very good approximation to the additive SE-ARD covariance.

Additive-VB : This is the VBEM algorithm presented in Section 4.2.2. It is computationally less-intensive than Additive-MCMC, since we optimise the hyperparameters instead of integrating them out via MCMC. The hyperparameters are initialised in a "sensible" way, unless otherwise stated. This usually has a particular meaning in the GP literature and we use the following protocol:

- Initialise the lengthscales such that $\ell_d = (\max(X_{:,d}) - \min(X_{:,d}))/2$,
- Initialise the signal variances such that $v_d = \text{var}(\mathbf{y})/D$,
- Initialise the noise variance to $\sigma_n^2 = \text{var}(\mathbf{y})/4$.

We run the VBEM algorithm (Algorithm 11) until the optimized value of θ in a given iteration has a squared distance less than 0.01 from the value of θ in the previous iteration. We also limit the number of VB iterations to 50, and this is rarely reached in practice.

Additive-Full : This corresponds to running the standard GP regression algorithm (Algorithm 1) with kernel:

$$k(\mathbf{x}, \mathbf{x}'; \theta) = \sum_{d=1}^{D} v_d \exp\left(-\frac{(\mathbf{x} - \mathbf{x}')^2}{2\ell_d^2}\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}'). \tag{4.66}$$

Note that we will only run the standard GPR algorithm if the training set size is less than 10000 points, as any larger experiment does not fit into memory. It would

also be reasonable to use a sum of Matérn(7/2) kernels here, in order to control for any variation in results due to differences in covariance function. However, as will become apparent, switching the Matérn(7/2) kernel with the squared-exponential creates a negligible difference in performance. In addition, it eases comparison to methods such as **SPGP**, as the software library for it supports squared-exponential kernels *only*.

PPGPR-greedy : This is Algorithm 12. We initialise the projection weights at every iteration iteration by means of a linear regression of X onto $\mathbf{y}^m$ in all cases except for one (the `kin40k` dataset – see Section 4.5.1.4) where we use random initialisation instead. These choices were driven by whichever options performed better. It appears that initialisation via linear regression works better more often. Once the weights are known we can initialise the scalar GP "sensibly" as described above (with $D = 1$). We stop the iterations when the change in validation-set NMSE is less than 1e-4. Many of the datasets we will use have standard train-test data splits. By default, we will use an 80-20 split of the original training set to form our training and validation sets.

PPGPR-MCMC : This algorithm refers to the case where we learn the projection weights W and the hyperparameters on a subset of data of size $\min(N, 1000)$ and use these to run the **Additive-MCMC** routine above using the projected inputs instead of the original inputs. Here, we learn the weights and hyperparameters in a non-greedy, yet computationally intensive way by simply using Algorithm 1 with kernel:

$$k(\mathbf{x}, \mathbf{x}'; \theta, \mathbf{w}) = v \exp\left(-\frac{(\mathbf{w}^\top \mathbf{x} - \mathbf{w}^\top \mathbf{x}')^2}{2\ell^2}\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}'), \tag{4.67}$$

on the subset. This algorithm is included mainly to portray the effect of the greedy, more efficient optimisation present in **PPGPR-greedy**. We initialise the weights randomly from standard IID Gaussians and initialise other hyperparameters according to the standard protocol.

GP-Full : This is the standard GP regression algorithm applied to the most

commonly used SE-ARD covariance, given by:

$$k(\mathbf{x}, \mathbf{x}'; \theta) = v \exp\left(-\sum_{d=1}^{D} \frac{(\mathbf{x} - \mathbf{x}')^2}{2\ell_d^2}\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}'). \tag{4.68}$$

Note that this kernel allows full interaction amongst all input variables, and will therefore be able to capture structure which an additive regression model cannot, making it an interesting choice for comparison.

SPGP : This algorithm is the sparse approximation method of Snelson and Ghahramani [2006], as applied to the **GP-Full** algorithm above. It was briefly introduced in Chapter 1. The key idea is to use a pseudo training-set $\{\bar{\mathbf{X}}, \bar{\mathbf{f}}\}$ with P noise-free input-output pairs, where $P < N$. Conceptually, this pseudo training-set is used to predict the original dataset $\{\mathbf{X}, \mathbf{y}\}$, giving rise to the following prior and likelihood model for sparse GP regression:

$$p(\bar{\mathbf{f}}|\bar{\mathbf{X}}, \theta) = \mathcal{N}(\mathbf{0}, \mathbf{K}_P), \tag{4.69}$$

$$p(\mathbf{y}|\mathbf{X}, \bar{\mathbf{X}}, \bar{\mathbf{f}}, \theta) = \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \bar{\mathbf{X}}, \bar{\mathbf{f}}) \tag{4.70}$$

$$= \mathcal{N}(\mathbf{y}|\mathbf{K}_{NP}\mathbf{K}_P^{-1}\bar{\mathbf{f}}, \mathbf{\Lambda} + \sigma_n^2 \mathbf{I}_N), \tag{4.71}$$

where $\mathbf{\Lambda}$ is a diagonal matrix of predictive variances, i.e. $\Lambda_{i,i} = K_{i,i} - \mathbf{k}_i^\top \mathbf{K}_P^{-1} \mathbf{k}_i$. Predictions can be made using the posterior over $\bar{\mathbf{f}}$ which can be obtained trivially using Equations 4.69 and 4.71. The marginal likelihood of SPGP is obtained by integrating out $\bar{\mathbf{f}}$ to obtain:

$$p(\mathbf{y}|\mathbf{X}, \bar{\mathbf{X}}, \theta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_{NP}\mathbf{K}_P^{-1}\mathbf{K}_{PN} + \mathbf{\Lambda} + \sigma_n^2). \tag{4.72}$$

This is optimised to learn the pseudo-input locations $\bar{\mathbf{X}}$ and θ. When using SPGP, we will always initialise the pseudo-input locations as a random subset of $\mathbf{X}$ and follow the usual protocol for initialising the rest of the hyperparameters. We will set $P = \min(N, 500)$ for all the problems analysed. We will use the implementation provided in Snelson and Ghahramani [2006] and limit ourselves to using the SE-ARD kernel, as this is the only kernel which is supported.

4.5.1.3 Experiment I : Runtimes on Synthetic Data

In this experiment, we illustrate how the runtimes of **Additive-VB**, **PPGPR-greedy**, **SPGP** and **Additive-Full** vary as a function of N and D. All the runs use synthetic data generated according to the following generative model:

$$y_i = \sum_{d=1}^{D} \mathbf{f}_d(X_{:,d}) + \epsilon \qquad i = 1, \ldots, N, \tag{4.73}$$

$$\mathbf{f}_d(\cdot) \sim \mathcal{GP}\left(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; [1,1])\right) \quad d = 1, \ldots, D, \tag{4.74}$$

$$\epsilon \sim \mathcal{N}(0, 0.01),$$

where $k_d(\mathbf{x}_d, \mathbf{x}'_d; [1,1])$ is given by the Matérn(7/2) kernel with unit lengthscale and amplitude. Recall that we can perform the sampling in Equation 4.74 in linear time and space using the FFBS algorithm given in Section 4.2.2. We will measure the runtime of the training phase (i.e., smoothing and hyperparameter learning given $\{\mathbf{X}, \mathbf{y}\}$) for all these algorithms. First, we fix D to 8 and collect runtimes for a set of values for N ranging from 200 to 50000. Note that we have not included the results for algorithms which use MCMC as these are always slower (although they still have similar asymptotic scaling). Furthermore, we use the MCMC-based algorithms mainly as a reference point for assessing the quality of the faster **Additive-VB** and **PPGPR-greedy** algorithms, as will be made clear in the following section. Also note that, for **Additive-VB** and **PPGPR-greedy** we have set the number of outer loop iterations (the number of VBEM iterations for the former, and the number of projections, M, for the latter) to be 10 for all N. In practice, the number of outer loop iterations will vary according to the dataset, causing fluctuations from the linear relationship to N clearly present in Figure 4.5. This figure clearly portrays the significant computational savings attained by exploiting the structure of the additive kernel. We obtain runtimes comparable to sparse GP algorithms despite using the entire training set during learning! In contrast, the **Additive-Full** algorithm cannot be run at all past 10000 points, as it doesn't fit into memory.

Secondly, we fix N to 1000 and vary D from 5 to 50, in intervals of 5. Figure 4.6 shows that the runtime also scales linearly as a function of D for all algorithms involved, though the constant factor for **Additive-VB** is higher than others due to the repeated calls made to the backfitting algorithm.

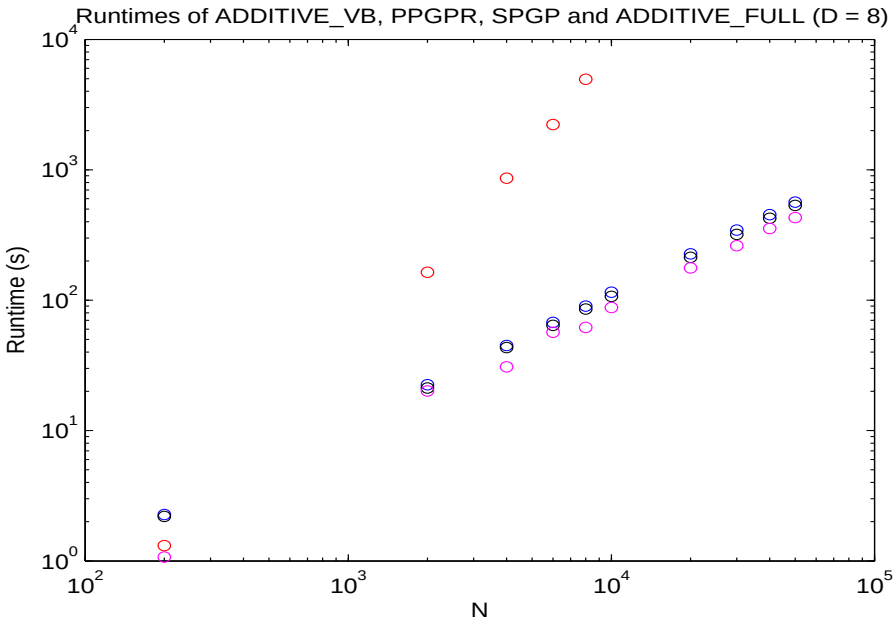


Figure 4.5: A comparison of runtimes for efficient Bayesian additive GP regression, projection-pursuit GP regression and generic techniques for full and sparse GP regression (with $N = [0.2, 2, 4, 6, 8, 10, 20, 30, 40, 50] \times 10^3$) presented as a log-log plot. The slope of Additive_Full (in red) is 2.8, and those of Additive_VB (blue), PPGPR (black) and SPGP (magenta) are between 1 and 1.05. This clearly illustrates the reduction from cubic runtime to $\mathcal{O}(N \log N)$.

4.5.1.4 Experiment II : Performance on Large Datasets

In this experiment, we assess the performance of all the algorithms above on a series of synthetic and large, real-world, benchmark datasets.

Synthetic Additive Data : We generate synthetic data from an additive model in exactly the same way as in Section 4.5.1.3. We have chosen a relatively smaller N and M in order to be able to obtain results for all the methods involved. In addition, we have kept D low in order to be able to effectively visualise the learned models. Recall that additive models are much easier to visualise than their non-additive counterparts since we can plot how each individual predictor relates to the target. Since the inherent additivity assumption is satisfied in this data, we expect all the **Additive-*** algorithms and those based on projection pursuit to perform well. Indeed, this expectation is verified in the top half of Table 4.1: both the additive model and the projection-pursuit algorithm have superior NMSE and MNLP when

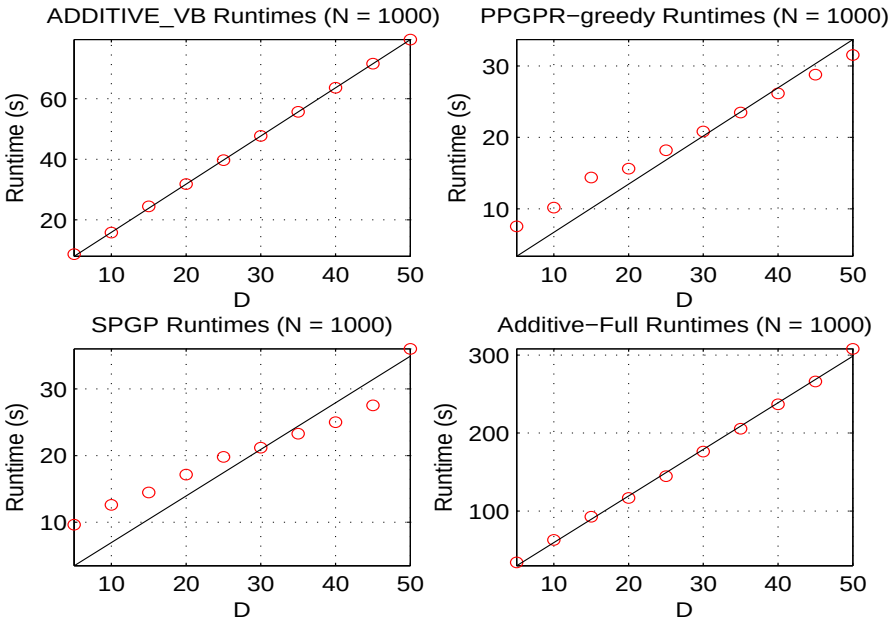


Figure 4.6: A comparison of runtimes for efficient Bayesian additive GP regression, projection-pursuit GP regression and generic techniques for full and sparse GP regression. We vary D from 5 to 50. As all methods clearly exhibit linear scaling with D, we use linear-linear plots.

compared with the GP models where full interaction is allowed. This superiority stems from the fact that an additive GP prior is simpler than a tensor GP prior and fits the data just as well (if not better). This results in a higher marginal likelihood (which is empirically verified, although not presented in the table below) which is reflected in better NMSE and MNLP scores. Figure 4.7 is an excellent illustration of this superior performance. We can see that **Additive-VB** has successfully recovered the 4 latent univariate functions which gave rise to the data (modulo constant offsets due to subtracting the mean). We can also see that **PPGPR-greedy** has learned automatically that there are 4 major additive components and has learned functional forms which are clearly consistent with the 4 true functions used to generate the data. For example, judging from the first row of the W matrix, we can see that the first component should be similar to the second function from the left; and from the fourth row of the matrix, we expect the fourth component to be the third function from the left but with the x-axis flipped (as the weight is negative). Both of these can clearly be verified visually!

Friedman's Synthetic Data : This is a standard synthetic dataset used commonly in the literature (e.g., Navone et al. [2001]). The seven input variables are sampled uniformly over the interval $[0, 1]$ and the targets are generated according to:

$$y_i = 10 \sin(\pi X_{i,1} X_{i,2}) + 20(X_{i,3} - 0.5)^2 + 10X_{i,4} + 5X_{i,5} + \epsilon, \qquad (4.75)$$
$$\epsilon \sim \mathcal{N}(0, \sigma_n^2), \quad i = 1, \ldots, N.$$

We set $N = 40000$ and $M = 5000$ in order to set up a regime where it would not be possible to use the full GP routines. In the bottom half of Table 4.1 we can see that the performance of **PPGPR-greedy**, **PPGPR-MCMC** and that of **SPGP** are comparable and that these two sets of algorithms fare better than the pure additive GP models. This is an unsurprising result because the first term in Equation 4.75 couples the first two input variables, and additive models cannot capture this type of structure. In contrast, the algorithms based on projection pursuit regression can improve accuracy considerably, thanks to the additional flexibility provided by being able to project the inputs. Note also that, in this case, **PPGPR-greedy** actually runs faster than **SPGP** as the number of dimensions of the learned input space was 6 (again, this is encouraging because it is close to the actual number of dimensions present in the generative model – 5).

Elevators : The `Elevators` dataset is the first of several large, real-world datasets we consider. It has been used in Lázaro-Gredilla [2010]. It is derived from a control problem pertaining to F16 aircraft where the 17 input variables represent the current state of the aircraft and the target relates to the action performed on its elevators. As can be seen in the top part of Table 4.2, for this particular dataset, **SPGP** performs comparably to **GP-Full** and both slightly outperform the additive and PPGPR methods, although not by a significant margin. This indicates that the additivity assumption is not a very strong one for this dataset although it does cause some degradation in performance.

Pumadyn-8fm : This dataset can be viewed as an example real-world regression task where the additivity assumption is perfectly reasonable. The `pumadyn` family of datasets are constructed from a simulation of a Puma 560 robot arm. The inputs include the angular positions, velocities and torques and the output variable is the

Table 4.1: Performance comparison of efficient Bayesian additive GP regression algorithms on synthetically-generated data. As expected, we see that additive GP models and **PPGPR-greedy** outperform **GP-Full** both in terms of runtime and predictive performance for additive data. We also observe that **SPGP**, while being the fastest method, gives the worst predictive performance in this case. Friedman's synthetic data adds some coupling between inputs, thus worsening the performance of additive models. It is interesting to see that **PPGPR-Greedy** is comparable in runtime and accuracy to **SPGP**.

Algorithm	NMSE	MNLP	Runtime (s)
Synthetic Additive Data ($N = 8000$, $M = 1000$, $D = 4$)			
Additive-MCMC	0.698	1.094	720
Additive-VB	0.703	1.112	125
Additive-Full	0.708	**1.070**	3097
PPGPR-Greedy	**0.669**	1.204	114
PPGPR-MCMC	0.714	1.223	846
GP-Full	0.738	1.534	2233
SPGP	0.792	1.702	**85**
Friedman's Synthetic Data ($N = 40000$, $M = 5000$, $D = 7$)			
Additive-MCMC	0.110	1.923	3134
Additive-VB	0.113	1.929	682
Additive-Full	N/A	N/A	N/A
PPGPR-Greedy	0.0424	1.426	**523**
PPGPR-MCMC	**0.0401**	1.438	4178
GP-Full	N/A	N/A	N/A
SPGP	0.0408	**1.410**	620

angular acceleration of one of the arm links. This particular dataset is classed "fairly linear", which justifies the assumption of additivity. Indeed, the **Additive-*** algorithms are comparable to **GP-Full** in terms of NMSE and MNLP, and there appears to be a slight edge in using PPGPR. The learned models are illustrated in Figure 4.8, where one can see exactly what "fairly linear" means in the context of this dataset. Furthermore, **Additive-VB** has discovered two linear components and has set the rest of the components to be "zero-signal, all-noise" (the errorbars plotted here do not include the noise component). Similarly, **PPGPR-greedy** assigns a linear component in its first iteration and rapidly converges to a setting where no more improvement in NMSE can be obtained (the fourth component is basically all noise).

Pumadyn-8nm : This dataset is similar to the above, except that its degree of nonlinearity is far greater. As a consequence, **GP-Full** does best, and is well-approximated by **SPGP**. The **Additive-*** algorithms perform significantly worse highlighting the damage the additivity assumption is causing in this particular case. Note, however, that **PPGPR-greedy** almost completely closes the performance gap.

kin40k : The `kin40k` dataset is a pathological case for any additive model, and it also forms a challenge to PPGPR-style algorithms. It is a highly nonlinear and low noise dataset. Therefore, kernels which support full interactions across all inputs are at a significant advantage, as can be seen in the final section of Table 4.2. Note the significant difference in performance between vanilla additive GP models (including **Additive-Full**) and **Full-GP** and **SPGP**. Again, the added flexibility present in **PPGPR** significantly boosts performance, although it still falls short. Indeed, the degree of nonlinearity is reflected by the fact that **PPGPR-greedy** converges after 20 iterations – indicating that it is preferable to have a 20-dimensional input space to explain the targets in an additive way, as opposed to the original 8.

4.5.2 Experiments on Classification Tasks

4.5.2.1 Performance Metrics

We will again use three performance measures when comparing the additive GP classification algorithm introduced Section 4.4 to other kernel classifiers (both Bayesian and non-Bayesian). These measures have been chosen to be consistent with those commonly used in the sparse GP classification literature, enabling straightforward comparison with other experimental results. We will focus on the task of binary classification. As the efficient implementation of Laplace's approximation can, in principle, be run with *any* likelihood function, extensions to tasks such as multi-class classification and Poisson regression can be performed without affecting asymptotic complexity.

- Runtime, measured in seconds. Jitter in all quantities presented pertaining to runtimes will be reduced by performing 10 independent runs.

- The test error rate :

$$\text{Error Rate} = \frac{\#(\text{incorrect classifications})}{\#(\text{test cases})}. \tag{4.76}$$

This measure is commonly used for the Support Vector Machine (SVM) classifier as its central goal is to return predicted class *labels*, not the probability of a new instance belonging to one class or the other. The other methods we use will always return probabilistic predictions, and when using this metric, we follow the protocol of assigning a test case to the class with highest probability. Recall that we had limited ourselves to the computation of $\hat{\mathbf{p}} = p(\mathbf{y}_\star|\hat{\mathbf{f}})$ and that the test error rate computed using these probabilities were, in fact, identical to that which is computed from the exact predictive probabilities given by Equation 1.44. A better measure for assessing the quality of *probabilistic* predictions is the *negative log-likelihood* (*cross-entropy*) of the test-set predictions, given below.

- The test-set Mean Negative Log-Likelihood (MNLL) :

$$\text{MNLP} = \frac{1}{N_\star} \sum_{i=1}^{N_\star} \left[y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i) \right]. \tag{4.77}$$

For both the test error rate and MNLL measures lower values indicate better performance.

4.5.2.2 Comparison Methodology

We will compare the following algorithms to each other using the metrics above. For datasets which are too large, it will not be possible to run the standard GP classification methods and the results corresponding to these cases will be omitted. For all the methods, we will rescale the inputs so that the training set inputs have zero mean and unit variance. We will also perform hyperparameter learning by *grid search* only. The reason for doing this is to make the comparison to Support Vector Machines (SVMs) fair, as this is the technique used for learning SVM hyperparameters. Note that for the SVM and the Informative Vector Machine (IVM) the lengthscale and amplitude parameters are tied across all dimensions and therefore

we will paramaterize the GP classifiers in a similar fashion. This limits the grid search to a 2-dimensional space, which is feasible. For SVMs and IVMs we will split the original training sets 80-20 to form a validation set, and optimise the validation set MNLL to select the hyperparameters. For GP-based classifiers we will use the marginal likelihood approximation. Note that using both gradient-based hyperparameter optimisation and different lengthscales and amplitudes per dimension ought to further improve performance although we will not consider these extensions.

Additive-LA : This is Algorithm 13 used to perform posterior inference for the following generative model:

$$
\begin{aligned}
y_i &\sim \text{Bernoulli}(p_i) \qquad i = 1, \ldots, N, \\
p_i &= g(f_i), \qquad (4.78) \\
\mathbf{f}(\cdot) &= \sum_d \mathbf{f}_d(\cdot), \qquad (4.79) \\
\mathbf{f}_d(\cdot) &\sim \mathcal{GP}\left(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; \theta_d)\right) \qquad d = 1, \ldots, D,
\end{aligned}
$$

where

$$
g(f_i) = \frac{1 - 2\epsilon}{1 + \exp(-f_i)} + \epsilon, \qquad (4.80)
$$

is the logistic link function, altered so that it tends to ϵ for large and negative values of f_i and to $1 - \epsilon$ for large and positive values of f_i. This greatly improves the numerical stability of the logistic likelihood function as it is common to find examples of datasets where one may observe a rare negative label in a region of high probability for the positive class (and vice versa). This causes very large noise variances to be introduced inside the underlying Kalman filter (as $W_{i,i}^{-1}$ is very high) which in turn runs the risk of causing numerical problems. Intuitively, the ϵ parameter plays the role of *label noise* which can, in fact, be learned from data, although we will fix it at 1e-5. We have the following expressions for $\boldsymbol{\nabla}$ and W for the link function in Equation 4.80.

$$
\nabla_i = \frac{(1 - 2\epsilon)(y_i \exp(-f_i) - (1 - y_i))}{1 + \exp(-f_i)} = (1 - 2\epsilon)(y_i - p_i), \qquad (4.81)
$$

$$
W_{i,i} = \frac{(1 - 2\epsilon)\exp(-f_i)}{(1 + \exp(-f_i))^2} = (1 - 2\epsilon)p_i(1 - p_i). \qquad (4.82)
$$

For the kernels $k_d(\cdot,\cdot)$, we will use the Matérn(7/2) covariance function. We will perform a grid search over the following space to find a good setting for the hyper-parameters: $\ell = [2^{-5}, 2^{-3}, \ldots, 2^{9}]$, $\sigma_f = [2^{-3}, 2^{-1}, \ldots, 2^{9}]$. This type of grid search is commonly used to select SVM free parameters and we have decided to keep this consistent across all the techniques being compared.

Additive-Full : This is the standard GP classification with Laplace's Approximation, as introduced in Rasmussen and Williams [2006]. The grid search over hyperparameters is identical to **Additive-LA** above.

Support Vector Machines (SVM) [▷▷]: The SVM is a popular algorithm, especially for classification problems, and was first introduced in Cortes and Vapnik [1995]. Its popularity stems from the fact parameter learning boils down to a convex optimisation problem, guaranteeing convergence to the global optimum. It can also be kernelized, allowing extensions to infinite dimensional feature spaces, such as the one provided by the commonly-used Gaussian Radial Basis Function (RBF) kernel. In addition, its loss function results in *sparse* solutions where the decision boundary depends only on a small subset of the training data, thus allowing fast predictions. It is not a probabilistic approach to classification. Thus the central goal is to optimise the test-set error rate, i.e., minimize training set classification error subject to model complexity constraints in order to avoid overfitting. This is achieved by minimizing the following objective function w.r.t. $\mathbf{w}$ and $\boldsymbol{\xi} \equiv \{\xi_n\}_{n=1}^N$:

$$\Omega_{\text{SVM}}(\mathbf{w}, \boldsymbol{\xi}) \equiv \underbrace{C \sum_{n=1}^{N} \xi_n}_{\text{Data fit penalty}} + \underbrace{\frac{1}{2}\mathbf{w}^\top \mathbf{w}}_{\text{Complexity penalty}} , \tag{4.83}$$

subject to

$$y_n \overbrace{\{\mathbf{w}^\top \phi(\mathbf{x}_n) + b\}}^{\equiv f(\mathbf{x}_n)} \geq 1 - \xi_n, \tag{4.84}$$

$$\xi_n \geq 0, \tag{4.85}$$

where $y_n \in \{-1, 1\}$. Notice how, if the sign of $f(\mathbf{x}_n)$ is consistent with the label then $0 \leq \xi_n < 1$. The parameter C controls the relative importance of the data fit term to the regularization term. This objective function can be optimized using Lagrange

multipliers $\{\alpha_n\}_{n=1}^N$ corresponding to the constraints in 4.84. The resulting dual Lagrangian problem is then to optimize the objective:

$$L(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \underbrace{\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m)}_{\equiv k(\mathbf{x}_n, \mathbf{x}_m)}, \tag{4.86}$$

subject to

$$0 \leq \alpha_n \leq C, \ \sum_{n=1}^{N} \alpha_n y_n = 0. \tag{4.87}$$

This optimisation task is convex (it is a quadratic programming problem). A popular algorithm to implement it is known as Sequential Minimal Optimisation (SMO) Platt [1998] and has scaling which is between linear and quadratic in N. Notice also that the SVM training scheme is easily kernelisable and in practice it is common to use:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \sum_{d=1}^{D} (\mathbf{x}_d - \mathbf{x}'_d)^2\right) \tag{4.88}$$

Once the optimal $\boldsymbol{\alpha}$ are computed, we can compute the optimal $\hat{\mathbf{w}}$ and $\hat{b}$ and use these to label unseen instances using $\text{sgn}(\hat{\mathbf{w}}^\top \phi(\mathbf{x}_\star) + \hat{b})$. Further details on SVMs can be found in Schölkopf and Smola [2001].

There exist many software libraries that implement SVMs. We have chosen to use LIBSVM (Chang and Lin [2011]) due to its ease of use and its support for MATLAB via an excellent MEX interface. It also implements an extension to the standard SVM algorithm which provides the user with *probabilistic* predictions, i.e. $p(y_\star = 1|\mathbf{x}_\star)$ as opposed to simply a label. This is important for our results because we would like to be able to compute the MNLL measure for SVMs also. The way probabilistic predictions are derived from the SVM is by using cross-validation (within the training set) to learn a logistic sigmoid map, using the cross-entropy metric (see Platt [1999]):

$$p(y = 1|\mathbf{x}) = \sigma(\beta_1 f(\mathbf{x}) + \beta_0) \tag{4.89}$$

We implement parameter learning for the SVM using a grid search over the joint space of C and γ, using $C = [2^{-5}, 2^{-3}, \ldots, 2^9]$, $\gamma = [2^{-9}, 2^{-7}, \ldots, 2^3]$.

Informative Vector Machine (IVM) [▷▷]: The IVM is a rare example of a sparse GP method which can be readily extended to handle non-Gaussian likelihoods such as the one for binary classification. The central idea is to find a Gaussian approximation to the true posterior as follows:

$$p(\mathbf{f}|\mathbf{y}, X, \theta) \propto \mathcal{N}(\mathbf{f}; 0, K) \prod_{n=1}^{N} \underbrace{p(y_n|f_n)}_{\text{Non-Gaussian}} \tag{4.90}$$

$$\approxeq \mathcal{N}(\mathbf{f}; 0, K) \prod_{i \in I} \mathcal{N}(m_i; f_i, v_i), \tag{4.91}$$

where $I \subset \{1, \ldots, N\}$ with fixed size (e.g. $\min(500, N)$). The set I is initialised to the empty set and elements are added incrementally by greedily adding whichever datapoint causes the biggest reduction in entropy for the Gaussian in Equation 4.91. The site parameters $\{m_i\}$ and $\{v_i\}$ are updated at each iteration k by minimizing $\mathrm{KL}(p_k(\mathbf{f})||q_k(\mathbf{f}))$, where:

$$p_k(\mathbf{f}) = \mathcal{N}(\mathbf{f}; 0, K) \prod_{i \in I_{k-1}} \mathcal{N}(m_i^{(k-1)}; f_i, v_i^{(k-1)}) p(y_{nk}|f_{nk}),$$

$$q_k(\mathbf{f}) = \mathcal{N}(\mathbf{f}; 0, K) \prod_{i \in I_k} \mathcal{N}(m_i^{(k)}; f_i, v_i^{(k)}).$$

Note the similarity of this approximation to EP, as introduced in 2.4. It is in fact a case of *Assumed Density Filtering* (ADF). For further details, see Minka [2001]. These updates are made tentatively when we are searching for the next example to add to the subset and are made permanently when the new example is added. The IVM algorithm runs in $\mathcal{O}(|I|^2 N)$ time and $\mathcal{O}(|I|N)$ space, and is thus comparable in terms of its scaling properties to Algorithm 13. For further details on IVM, consult Lawrence et al. [2003].

We use the IVM implementation given in Lawrence et al. [2003]. We use the RBF kernel where all the lengthscales and signal amplitudes are constrained to be equal. We set the subset size to be $\min(500, N)$. Note that, by plugging the approximation in Equation 4.91 into the expression for the marginal likelihood in Equation 1.43 we can optimise an approximation to the true marginal likelihood using conjugate gradients. However, we choose to use grid search much like the one

for **Additive-LA**.

Logistic Regression : This is the standard maximum-likelihood logistic regression model. We include it in our analysis as a performance baseline. It can only capture linear relationships between the inputs and the labels, although it runs extremely fast when compared with all the methods we consider here.

4.5.2.3 Experiment I : Runtimes on Synthetic Data

We perform the same experiment with runtimes as was done for the regression case. We sample sythetic data from the generative model of **Additive-LA** for the same set of values of N and D. The results are illustrated in Figures 4.10 and 4.11. Note, however, that in this case we report runtime values for *fixed* hyperparameter settings, as opposed to the regression case where we timed the hyperparameter learning process.

4.5.2.4 Experiment II : Performance on Large Datasets

In this experiment, we assess the performance of all the algorithms above on a set of synthetic or toy-sized datasets, and subsequently analyse performance on larger datasets.

Synthetic Additive Classification Data : We sample synthetic data from the generative model of **Additive-LA** with $N = 2000$ and $D = 4$. As illustrated in Figure 4.12, despite the limited amount of information available in binary labels, the recovery of the additive components is still quite accurate (modulo constant offsets). Unsurprisingly, high frequency components cannot be recovered very accurately. As the data is a sample from the generative model for additive classification we expect both **Additive-LA** and **Additive-Full** to dominate in accuracy – this is empirically verified in Table 4.3.

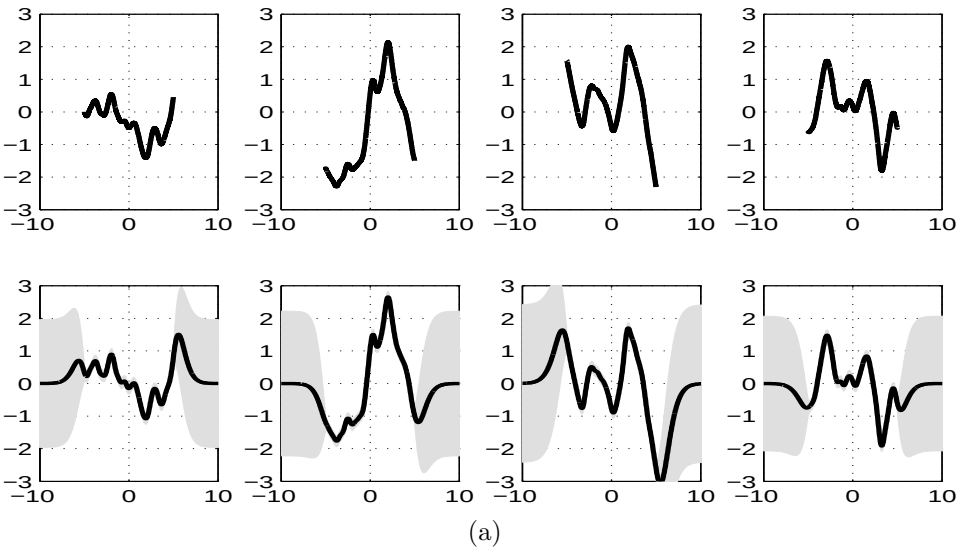
Breast Cancer : This is a small dataset where the aim is to diagnose a breast tumour as malignant or benign, using 9 discrete features. We see that the **Additive-LA** outperforms alternatives on this task.

Magic Gamma Telescope : This is a large binary classification dataset where the aim is to classify a *shower image* obtained via the collection of Cherenkov photons. Using certain features of the image, we would like to be able to predict if it

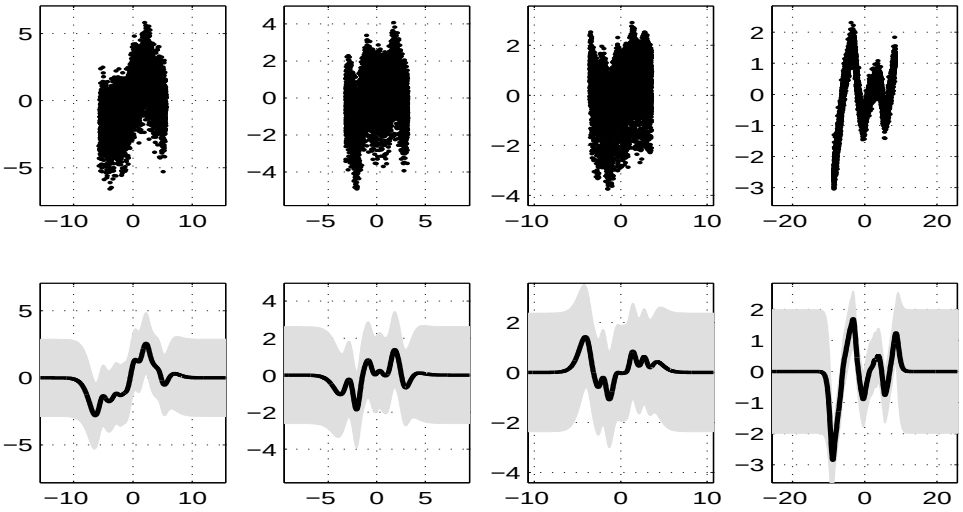
was the result of primary high energy gamma rays (positive class) or was the result of hadronic showers initiated by cosmic rays in the upper atmosphere (negative class). For further details see Bock et al. [2004]. We see that the **Additive-LA** outperforms alternatives for this task also.

IJCNN : This is from the IJCNN 2001 Neural Network Challenge. For further information see Chang and Lin [2001]. Although the SVM gives the best performance for this dataset, it is still worth noting that **Additive-LA** is a close second, consistently outperforming the IVM.

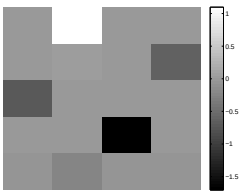
USPS Digit Classification : This is the US Postal Service Digit Classification dataset. It is a benchmark dataset for testing classification algorithms. The original problem has 10 classes, corresponding to handwritten digits 0-9. We focus on the binary classification of 0 versus not-0. For this problem, **SVM** and **IVM** perform comparably and both outperform **Additive-LA**. This is not very surprising, as for classification of images we would expect that pixel values jointly affect the class label.



(a)



(b)

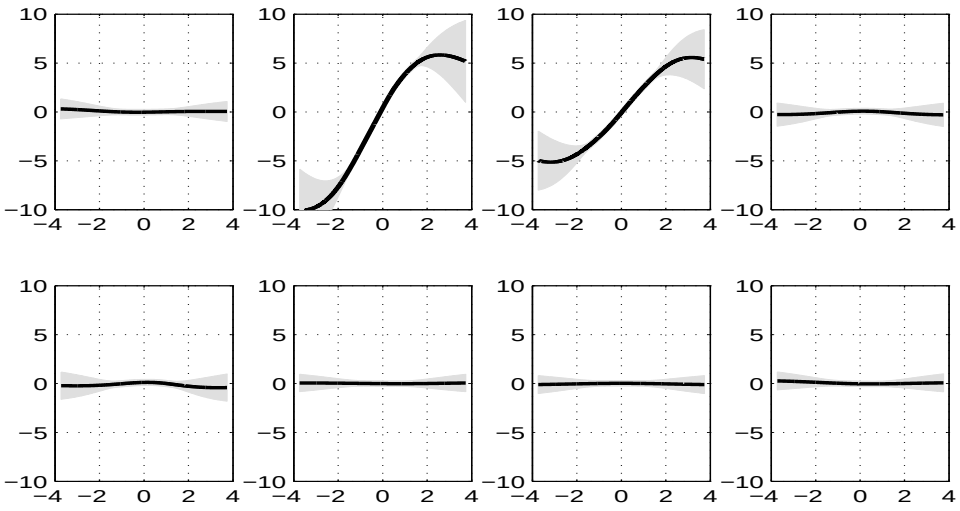


(c)

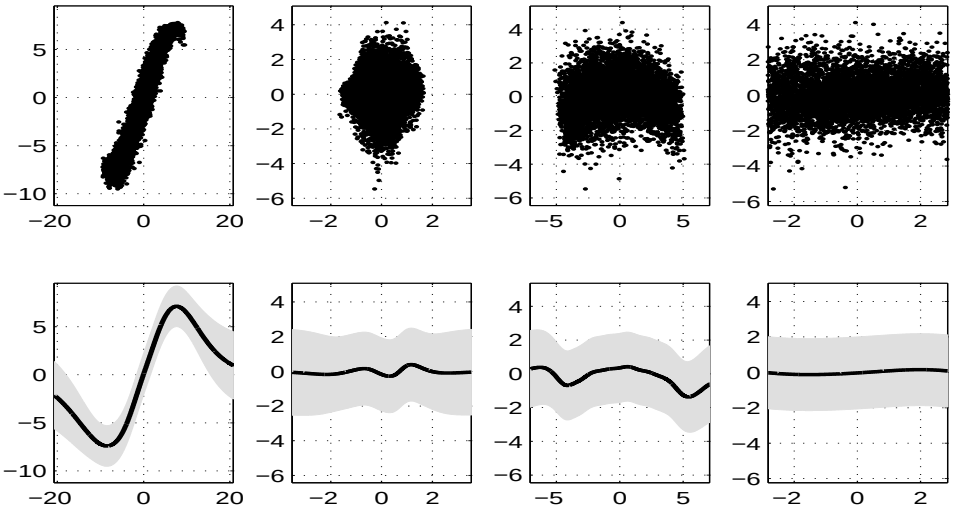
Figure 4.7: Inference results for the *Synthetic* dataset. (a) Top: The 4 *true* univariate functions used to generate the data. Bottom: Means and variances computed using the E-step of VBEM at learned hyperparameter settings. (b) Residuals (Top) and learned functions (Bottom) in the first 4 iterations of **PPGPR-greedy** (posterior over functions includes noise). (c) W matrix ($M = 5$).

Table 4.2: Performance comparison of efficient Bayesian additive GP regression algorithms with generic techniques for full and sparse GP regression on large, benchmark datasets. For datasets which do not exhibit significant nonlinearities and couplings between inputs (*Elevators* and *Pumadyn-8fm*) we see that additive GP models and **PPGPR-greedy** perform comparably to generic GP techniques, at a fraction of the computational cost. For more complex datasets, such as *kin40k*, we see that using **GP-Full** (or **SPGP**) is preferable. Nonetheless, it is worth noting how much **PPGPR-greedy** improves predictive performance in this case.

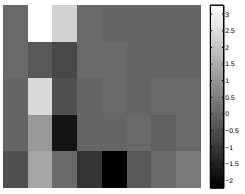
Algorithm	NMSE	MNLP	Runtime (s)
Elevators ($N = 8752$, $M = 7847$, $D = 17$)			
Additive-MCMC	0.145	−4.586	1150
Additive-VB	0.167	−4.521	178
Additive-Full	0.147	−4.587	8038
PPGPR-Greedy	0.128	−4.684	211
PPGPR-MCMC	0.222	−3.741	1244
GP-Full	**0.104**	−4.767	8219
SPGP	0.114	**−4.811**	**102**
Pumadyn-8fm ($N = 7168$, $M = 1024$, $D = 8$)			
Additive-MCMC	0.0705	1.608	611
Additive-VB	0.0716	1.627	92
Additive-Full	0.0714	1.617	3788
PPGPR-Greedy	0.0514	**1.437**	103
PPGPR-MCMC	**0.0502**	1.443	913
GP-Full	0.0545	1.484	2360
SPGP	0.0540	1.478	**63**
Pumadyn-8nm ($N = 7168$, $M = 1024$, $D = 8$)			
Additive-MCMC	0.374	2.640	633
Additive-VB	0.376	2.642	93
Additive-Full	0.372	2.641	3912
PPGPR-Greedy	0.0507	1.511	105
PPGPR-MCMC	0.0430	1.421	934
GP-Full	**0.0304**	**1.386**	2378
SPGP	0.0306	1.394	**71**
kin40k ($N = 10000$, $M = 30000$, $D = 8$)			
Additive-MCMC	0.945	1.391	2170
Additive-VB	0.945	1.391	438
Additive-Full	0.948	1.388	9415
PPGPR-Greedy	0.185	0.507	674
PPGPR-MCMC	0.138	0.428	2985
GP-Full	**0.0128**	**−0.897**	9302
SPGP	0.0502	−0.326	**214**



(a)

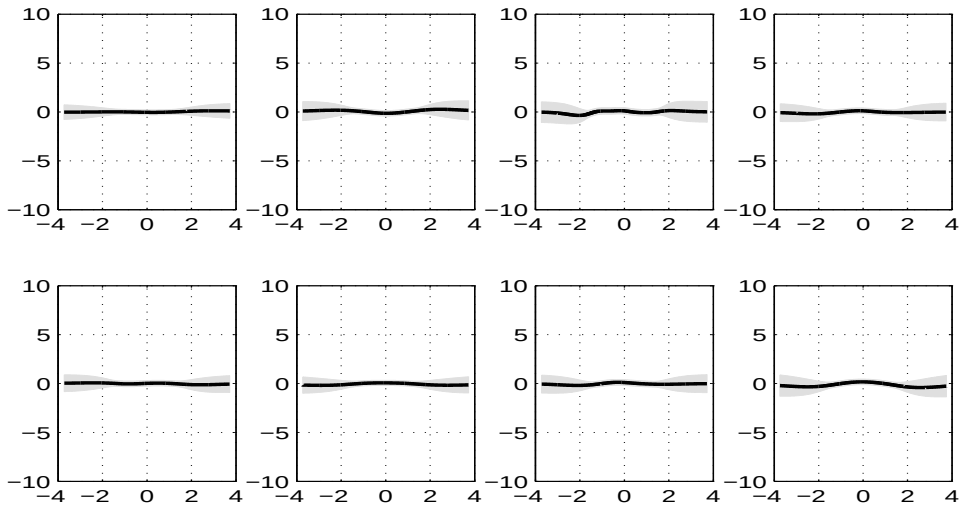


(b)

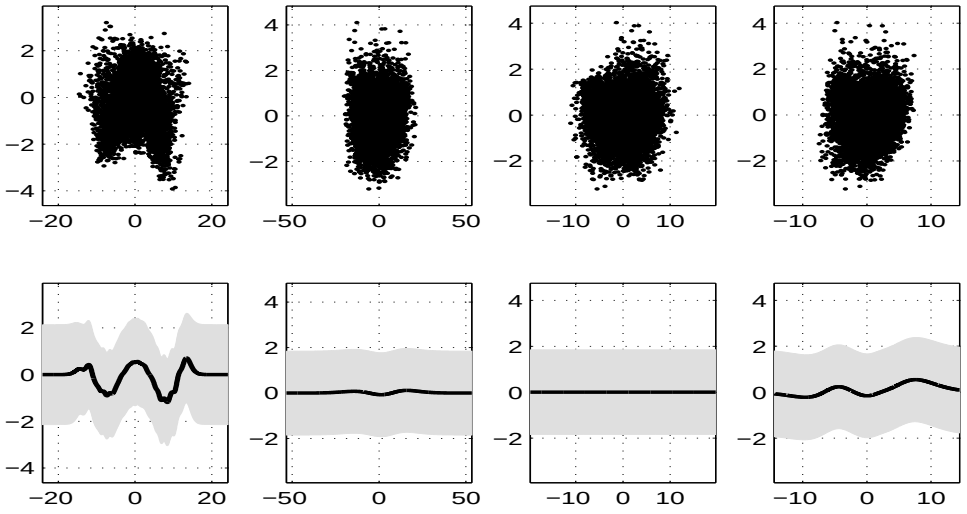


(c)

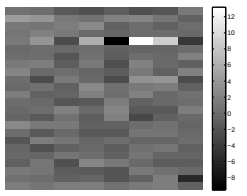
Figure 4.8: Inference results for the *Pumadyn-8fm* dataset. (a) The 8 additive components inferred by the E-step of VBEM at optimised hyperparameter settings. (b) Residuals (Top) and learned functions (Bottom) in the first 4 iterations of **PPGPR-greedy** (posterior over functions includes noise). (c) W matrix ($M = 5$).



(a)



(b)



(c)

Figure 4.9: Inference results for the *kin40k* dataset. (a) The 8 additive components inferred by the E-step of VBEM at optimised hyperparameter settings. (b) Residuals (Top) and learned functions (Bottom) in the first 4 iterations of **PPGPR-greedy** (posterior over functions includes noise). (c) W matrix ($M = 20$).

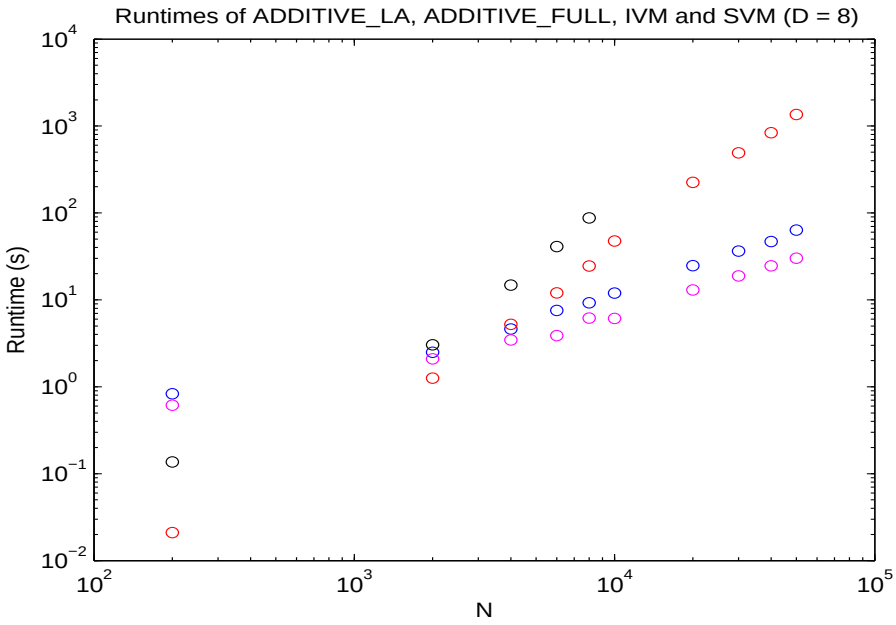


Figure 4.10: Classification runtimes for different N displayed as a log-log plot. The fitted slopes of **Additive-LA** (blue) and **IVM** (magenta) are 1.11 and 1.01 respectively. The slope of **SVM** (red) is 2.11, and that of **Additive-Full** (black) is 2.8.

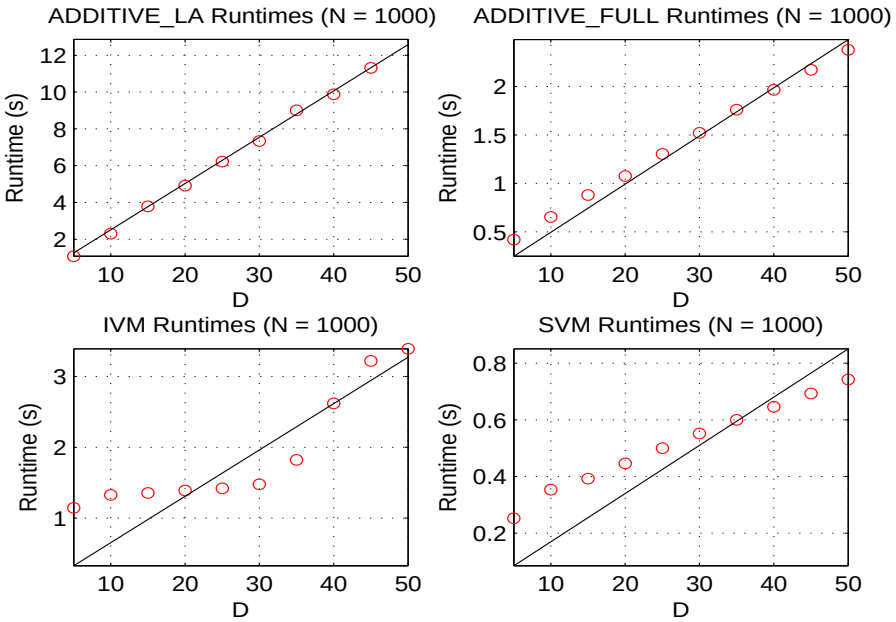


Figure 4.11: Classification runtimes for D ranging from 5 to 50, displayed as a linear-linear plot. We see that the **Additive-*** algorithms clearly have linear scaling with D, although the relationship of runtimes to D appears more complex for **IVM** and **SVM**.

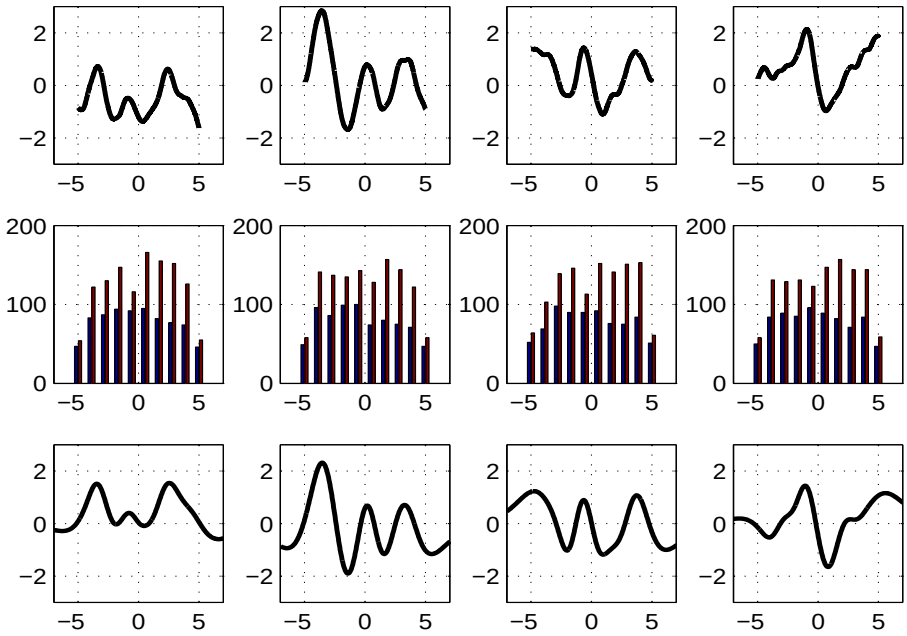


Figure 4.12: Top: The four univariate functions used to generate the class labels. Middle: Histogram of labels as a function of each input dimension. Bottom: The four additive components inferred by the Laplace Approximation.

Table 4.3: Performance on synthetic and small classification datasets. As expected, if the underlying function is truly additive, we see that **Additive-LA** and **Additive-Full** outperform alternatives in terms of accuracy. This warrants the use of **Additive-LA**, as it is also competitive in terms of computational complexity. We also see that **Additive-LA** is performing well for small datasets, as the amount of information in the data is not sufficient to create a preference for more complicated functions supported by **Full-GP**, **SVM** and **IVM**.

Algorithm	Error Rate	MNLL	Runtime (s)
Synthetic Additive Data ($N = 2000$, $M = 1000$, $D = 4$)			
Additive-LA	0.356	**0.557**	167
Additive-Full	**0.341**	0.580	305
Full-GP	0.437	0.624	335
SVM	0.437	0.737	124
IVM	0.479	0.709	128
Logistic	0.403	0.744	0
Breast Cancer ($N = 359$, $M = 90$, $D = 9$)			
Additive-LA	**0**	**0.0236**	110
Additive-Full	0.0333	0.0853	22
Full-GP	0	0.0612	23
SVM	0	0.0368	54
IVM	0.0111	0.586	327
Logistic	0.0556	0.0995	0

Table 4.4: Performance on large, real-world classification datasets. In contrast to the regression case, we observe that **Additive-LA** more frequently provides a competitive alternative to **SVM** and **IVM**. We conjecture that this phenomenon comes about due to the fact that binary labels do not give much information about the underlying function generating them. As a result, the additivity assumption becomes a weaker assumption for classiciation tasks.

Algorithm	Error Rate	MNLL	Runtime (s)
Magic Gamma Telescope ($N = 15216$, $M = 3804$, $D = 10$)			
Additive-LA	**0.142**	**0.348**	1580
Additive-Full	N/A	N/A	N/A
Full-GP	N/A	N/A	N/A
SVM	0.166	0.397	9833
IVM	0.341	0.642	1780
Logistic	0.209	0.463	1
IJCNN ($N = 49990$, $M = 91701$, $D = 13$)			
Additive-LA	0.0513	0.157	8281
Additive-Full	N/A	N/A	N/A
Full-GP	N/A	N/A	N/A
SVM	**0.0169**	**0.0482**	71083
IVM	0.0950	0.693	3880
Logistic	0.337	0.666	0
USPS ($N = 7291$, $M = 2007$, $D = 256$)			
Additive-LA	0.0164	0.0490	14686
Additive-Full	N/A	N/A	N/A
Full-GP	N/A	N/A	N/A
SVM	0.00897	**0.0317**	704
IVM	**0.00648**	0.0348	1580
Logistic	0.0204	0.0678	0

Chapter 5

Gaussian Processes on Multidimensional Grids

One of the better-known structured GP techniques involves the exploitation of *Toeplitz* structure in covariance matrices. Such matrices arise commonly when inputs are scalar and are on the integer lattice. Further details can be found in Golub and Van Loan [1996], Storkey [1999], Zhang et al. [2005] and Cunningham et al. [2008]. The fact that computations can be greatly simplified when scalar inputs live on a lattice begs the question of whether inputs living on a D-dimensional lattice can result in similar computational savings. The central contribution this chapter presents is the first full answer to this question, which, in a nutshell, turns out to be a "yes". We derive an algorithm which implements all operations in Algorithm 1 in $\mathcal{O}(N)$ time and space, for inputs arranged on a lattice. Note that the lattice does *not* have to be equispaced, as is the case for Toeplitz matrix methods, although it does have to be a *full* grid, thus constraining N to grow exponentially with D. It is helpful to consider this chapter as the study of how to generalize Toeplitz methods to higher dimensional input spaces, just as chapter 4 can be seen as a similar generalisation of the techniques in chapter 2.

5.1 Introduction

So far, we have illustrated several efficient GP regression algorithms derived as a consequence of making a range of assumptions about the covariance function or

about the properties we expect a dataset to have, such as nonstationarity. In this chapter, we explore the significant efficiency gains to be had by making assumption about *input locations*. A good example of this is found in Cunningham et al. [2008] where it is shown that, for uniformly-spaced input locations over a scalar input space, it is possible to reduce run-time complexity to $\mathcal{O}(N \log N)$ and memory requirements to $\mathcal{O}(N)$ through the exploitation of Toeplitz structure induced in the covariance matrix, for *any* stationary kernel. This is comparable to the efficiency of Gauss-Markov Processes, studied in Chapter 2. Indeed, although the techniques in Chapter 2 work with arbitrary input locations, they are restricted to kernels that can be represented using SSMs. The two ideas are therefore somewhat complementary. In this chapter, we demonstrate that significant efficiency gains can also be obtained in D-dimensional input spaces (with $D > 1$), as long as the input locations $\mathbf{X}$ lie on a *Cartesian grid*, i.e.,

$$\mathbf{X} = \mathbf{X}^{(1)} \times \cdots \times \mathbf{X}^{(D)}, \tag{5.1}$$

where $\mathbf{X}^{(i)}$ represents the vector of input locations along dimension i and the operator $\times$ represents the Cartesian product between vectors. Interestingly, the elements in each of the $\mathbf{X}^{(i)}$ can be *arbitrary*, i.e., $\mathbf{X}^{(i)} \in \mathbb{R}^{G_i}$ where G_i is the length of vector $\mathbf{X}^{(i)}$. Note that, by definition of the Cartesian product, $\mathbf{X}$ will therefore be of size $\left(\prod_{i=1}^{D} G_i\right)$-by-$D$, and in general will represent a non-uniform grid over D dimensions. In Section 5.2 we will present an algorithm to perform exact GP regression on such input spaces that runs in $\mathcal{O}(N)$ time, using $\mathcal{O}(N)$ memory! Note the disclaimer however: since $N = \prod_{i=1}^{D} G_i$ the computational and memory requirements both scale exponentially with dimension, limiting the applicability of the algorithms presented in this chapter to around 7 or 8 dimensions. Despite suffering badly from the "curse of dimensionality", one can imagine a number of example applications for which D is not too high and the inputs are often on a grid. Some of these applications are demonstrated in Section 5.3.

Firstly, there are many climate datasets which record measurements of quantities such as ocean surface temperatures, CO_2 concentrations at a grid of locations over the earth's surface. A typical example involving sea surface temperatures can be found in Rayner et al. [2006]. In this particular example, we have a time-series of "sensor" measurements arranged as a grid of geographic locations. Indeed, the time-series is also discretized, so one could view the entire dataset as living on a 3-D

Cartesian grid.

Secondly, consider the task of globally optimizing an unknown utility function with respect to a D-dimensional set of parameters. A good example could be optimizing the yield of a manufacturing process with respect to a set of parameters controlling it. As the utility function is unknown, it makes a lot of sense to model it with a GP, especially when evaluating an input-output pair is costly or derivatives of the utility function are not available. This idea was first introduced in O'Hagan and Kingman [1978], and has been the topic of many papers, such as Osborne [2010], Gramacy [2005], Srinivas et al. [2010]. The central question in GP-optimization (GPO) is where to evaluate the utility function in order to optimize a trade-off between exploration and exploitation. Often the first optimization iteration focusses on exploration and a sensible choice of initial input locations is a multidimensional grid[1], especially if little is know, a priori, about the region where the optima might live. After evaluating the utility function on the grid, we may update our belief about where we think the optimum is by optimizing the surrogate function given by $\mu_\star(\mathbf{x}_\star)$ or suggest new places to evaluate the utility function by optimizing $\mu_\star(\mathbf{x}_\star) + \lambda \left(\Sigma_\star(\mathbf{x}_\star)\right)^{1/2}$ where $\mu_\star$ and $\Sigma\star$ are given by Equations 1.29 and 1.30 (with $M = 1$). λ encodes the application-specific trade-off between exploration and exploitation (see Srinivas et al. [2010]). The results in this chapter imply that the first optimization iteration can run more efficiently than previously thought and larger grids can be used to boost initial exploration in GPO.

Another example arises in the regression setting where N is in the order of millions and D is comparatively much lower (e.g. $D < 8$). In such a scenario, it makes sense to bin the observations over a multidimensional grid and form a much smaller dataset $\{\mathbf{X}, \mathbf{y}\}$ where $\mathbf{X}$ are the grid centres and $\mathbf{y}$ represent the mean of the observations falling in each bin. Notice that the noise on the targets are now *input-dependent* leading to a diagonal noise covariance (as opposed to the usual spherical noise assumption). This is because the noise standard deviation in each bin is proportional to $\frac{1}{\sqrt{N_b}}$ where N_b is the number of observations that fell into bin b. Indeed, we will study precisely this type of application in Section 5.3.2.

Before delving into the description of the `GPR_GRID` algorithm we note that it is

[1] Assuming our parameter space is not too high dimensional – if it were, the global optimization problem would be a very tough one.

possible to enable the use of Toeplitz matrix inversion techniques in higher dimensions using the algorithm described in Storkey [1999]. In this algorithm one assumes that the inputs are on a *uniform* grid and Toeplitz structures in covariance matrices are obtained by mapping inputs onto a hyper-cylinder in input space. Additionally, one needs to assume that the covariance function has finite support, i.e., one for which long-distance correlations are zero. Hyperparameter learning also poses a problem as explained in Storkey [1999]. In this chapter, we attain superior complexity for arbitrary grids while making much milder assumptions about the covariance function and sidestepping any problems in hyperparameter optimization.

5.2 The `GPR_GRID` Algorithm

The algorithm presented in this section works with *any* covariance function which is a *tensor product* kernel and when the only marginals we are interested in live on a multi-dimensional grid. A covariance function $k(\cdot, \cdot)$ is a tensor product kernel if it computes covariances which can be written as a separable product over dimensions $d = 1, \ldots, D$. This means that for any two D-dimensional inputs $\mathbf{x}_i$, $\mathbf{x}_j \in \mathbf{X}$ we can write:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \prod_{d=1}^{D} k_d \left(\mathbf{x}_i^{(d)}, \mathbf{x}_j^{(d)} \right), \tag{5.2}$$

where $\mathbf{x}_i^{(d)} \in \mathbf{X}^{(d)}$ is simply the d-th element of input $\mathbf{x}_i$ and $k_d(\cdot, \cdot)$ is *any* positive definite kernel defined over a *scalar* input space. It can be shown that the positive definiteness of the individual k_d is a necessary and sufficient condition for $k(\cdot, \cdot)$ to be a positive kernel over a D-dimensional input space. As a consequence of this result, most standard covariance functions living on $\mathbb{R}^D$ are tensor product kernels, since this is the most intuitive and straightforward way to generalize a kernel (or set of kernels) defined over scalar inputs to multiple dimensions. Indeed, all the covariance functions we introduced in Chapter 1 and most of the ones presented in the standard reference in Rasmussen and Williams [2006] are tensor product kernels.

For example, for the squared-exponential kernel we may write:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(\frac{(\mathbf{x} - \mathbf{x}')^\top \Lambda (\mathbf{x} - \mathbf{x}')}{2}\right) \tag{5.3}$$

$$= \sigma_f^2 \exp\left(-\sum_{d=1}^{D} \frac{\left(\mathbf{x}_i^{(d)} - \mathbf{x}_j^{(d)}\right)^2}{2\ell_d^2}\right)$$

$$= \prod_{d=1}^{D} \sigma_f^{2/D} \exp\left(-\frac{\left(\mathbf{x}_i^{(d)} - \mathbf{x}_j^{(d)}\right)^2}{2\ell_d^2}\right). \tag{5.4}$$

Equation 5.4 indicates that the individual k_d are squared-exponential kernels over scalar inputs with amplitude $\sigma_f^{2/D}$. Despite the abundance of tensor product kernels in the literature there are several examples which do *not* fall in this category, including dot-product kernels, the neural network kernel and the "factor analysis" kernel (see Rasmussen and Williams [2006]). Nonetheless, we see that the restriction to tensor product kernels does not severely restrict the generality of GP priors we can work with.

The Kronecker Product

It is straightforward to show that a tensor product covariance function evaluated over a Cartesian grid of input locations will give rise to a covariance matrix that can be written as a *Kronecker product* of D smaller covariance matrices which are each formed by evaluating the axis-aligned kernel k_d over the inputs in $\mathbf{X}^{(d)}$. This result is a direct consequence of the definition of the Kronecker product – it is the generalization of the tensor product to matrices.

Definition 3. *If* $\mathbf{A}$ *is an* m*-by-*n *matrix and* $\mathbf{B}$ *is a* p*-by-*q *matrix, then the Kronecker product* $\mathbf{A} \otimes \mathbf{B}$ *is the* mp*-by-*nq *matrix*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

More generally,

Let $\mathbf{A} = \mathbf{A_1} \otimes \cdots \otimes \mathbf{A_D} = \bigotimes_{d=1}^{D} \mathbf{A_d}$*, where each of the* $\mathbf{A_d}$ *is a* G_d*-by-*G_d *matrix. Then,*

$$\mathbf{A}(i,j) = \mathbf{A}_1(i^{(1)}, j^{(1)})\mathbf{A}_2(i^{(2)}, j^{(2)}) \ldots \mathbf{A}_D(i^{(D)}, j^{(D)}), \tag{5.5}$$

- *where* $0 \leq i^{(d)}, j^{(d)} < G_d$ *and* $0 \leq i < N$, $0 \leq j < N$,
- $i = \sum_{d=1}^{D} G_d^d i^{(d)}$, *and* $j = \sum_{d=1}^{D} G_d^d j^{(d)}$,
- $\mathbf{A}$ *is thus a* $\left(\prod_{d=1}^{D} G_d\right)$*-by-*$D$ *matrix.*

Given that we have ordered the inputs correctly in every matrix involved, Equation 5.2 implies that for every $i, j \in \{1, \ldots, N \equiv \prod_{d=1}^{D} G_d\}$ we can write

$$\mathbf{K}(i,j) = \mathbf{K}_1(i^{(1)}, j^{(1)})\mathbf{K}_2(i^{(2)}, j^{(2)}) \ldots \mathbf{K}_D(i^{(D)}, j^{(D)}), \tag{5.6}$$

where $\mathbf{K}$ is the N-by$N$ (noise-free) covariance matrix over our training set which lives on the Cartesian grid, and $\mathbf{K}_d$ is the G_d-by-G_d covariance matrix defined over the vector of scalar input locations in $\mathbf{X}^{(d)}$. Therefore, using Equation 5.5, we arrive at the result fundamental to the `GPR_GRID` algorithm:

$$\mathbf{K} = \bigotimes_{d=1}^{D} \mathbf{K}_d. \tag{5.7}$$

Properties of the Kronecker Product

Before we can show how the identity in Equation 5.7 leads to a linear-time, linear-memory GP regression algorithm we list a few of the basic properties of the Kronecker product in Equations 5.8 through to 5.15. In these equations, N_X gives the size of the square matrix $\mathbf{X}$, and operator vec($\mathbf{X}$) corresponds to a column-wise stacking of the columns of $\mathbf{X}$. We then continue to prove several important theorems which apply to a covariance matrix for which Equation 5.7 holds.

Basic properties of the Kronecker Product (for square matrices)

$$\begin{aligned}
\text{Bilinearity}: &\quad \mathbf{A}\otimes(\mathbf{B}+\mathbf{C}) = \mathbf{A}\otimes\mathbf{B}+\mathbf{A}\otimes\mathbf{C} && (5.8)\\
\text{Associativity}: &\quad (\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C} = \mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}) && (5.9)\\
\text{Mixed-product property}: &\quad (\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) = \mathbf{AC}\otimes\mathbf{BD} && (5.10)\\
\text{Inverse}: &\quad (\mathbf{A}\otimes\mathbf{B})^{-1} = \mathbf{A}^{-1}\otimes\mathbf{B}^{-1} && (5.11)\\
\text{Transpose}: &\quad (\mathbf{A}\otimes\mathbf{B})^{\top} = \mathbf{A}^{\top}\otimes\mathbf{B}^{\top} && (5.12)\\
\text{Trace}: &\quad \operatorname{tr}(\mathbf{A}\otimes\mathbf{B}) = \operatorname{tr}(\mathbf{A})\operatorname{tr}(\mathbf{B}) && (5.13)\\
\text{Determinant}: &\quad \det(\mathbf{A}\otimes\mathbf{B}) = (\det\mathbf{A})^{N_A}(\det\mathbf{B})^{N_B} && (5.14)\\
\text{Vec}: &\quad \operatorname{vec}(\mathbf{CXB}^{\top}) = (\mathbf{B}\otimes\mathbf{C})\operatorname{vec}(\mathbf{X}) && (5.15)
\end{aligned}$$

Theorem 5.1. *If* $\mathbf{K} = \bigotimes_{d=1}^{D}\mathbf{K}_d$, *then* $\mathbf{K}^{-1} = \bigotimes_{d=1}^{D}\mathbf{K}_d^{-1}$.

Proof.

$$\begin{aligned}
\mathbf{K}^{-1} &= \left(\bigotimes_{d=1}^{D}\mathbf{K}_d\right)^{-1}\\
&= \mathbf{K}_1^{-1}\otimes\left(\bigotimes_{d=2}^{D}\mathbf{K}_d\right)^{-1} && [5.11]\\
&\vdots\\
&= \bigotimes_{d=1}^{D}\mathbf{K}_d^{-1}.
\end{aligned}$$

□

Theorem 5.2. *If* $\mathbf{L}_d$ *is a matrix square-root of* $\mathbf{K}_d$ *such that* $\mathbf{K}_d = \mathbf{L}_d\mathbf{L}_d^{\top}$, *then*

$$\mathbf{L} \equiv \bigotimes_{d=1}^{D}\mathbf{L}_d, \tag{5.16}$$

gives the corresponding matrix square-root of $\mathbf{K}$, *with* $\mathbf{K} = \mathbf{L}\mathbf{L}^{\top}$.

Proof (By Induction). Trivially true for $d = 1$. Now assume truth for $d = i$. i.e.,

$$\bigotimes_{d=1}^{i} \mathbf{L}_d \mathbf{L}_d^\top = (\mathbf{L}_1 \otimes \cdots \otimes \mathbf{L}_i)(\mathbf{L}_1 \otimes \cdots \otimes \mathbf{L}_i)^\top.$$

Then,

$$\begin{aligned}
\bigotimes_{d=1}^{i+1} \mathbf{L}_d \mathbf{L}_d^\top &= \left(\bigotimes_{d=1}^{i} \mathbf{L}_d \mathbf{L}_d^\top\right) \otimes \left(\mathbf{L}_{i+1} \mathbf{L}_{i+1}^\top\right) \\
&= \left((\mathbf{L}_1 \otimes \cdots \otimes \mathbf{L}_i)(\mathbf{L}_1 \otimes \cdots \otimes \mathbf{L}_i)^\top\right) \otimes \left(\mathbf{L}_{i+1} \mathbf{L}_{i+1}^\top\right) \qquad [5.16] \\
&= (\mathbf{L}_1 \otimes \cdots \otimes \mathbf{L}_i \otimes \mathbf{L}_{i+1})(\mathbf{L}_1 \otimes \cdots \otimes \mathbf{L}_i \otimes \mathbf{L}_{i+1})^\top. \qquad [5.10, 5.12] \quad \square
\end{aligned}$$

A very similar result can be analogously obtained for *eigendecompositions*:

Theorem 5.3. *Let* $\mathbf{K}_d = \mathbf{Q}_d \mathbf{\Lambda_d} \mathbf{Q}_d^\top$ *be the eigendecomposition of* $\mathbf{K}_d$*. Then the eigendecomposition of* $\mathbf{K} = \bigotimes_{d=1}^{D} \mathbf{K}_d$ *is given by* $\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$ *where*

$$\mathbf{Q} \equiv \bigotimes_{d=1}^{D} \mathbf{Q}_d, \tag{5.17}$$

$$\mathbf{\Lambda} \equiv \bigotimes_{d=1}^{D} \mathbf{\Lambda}_d. \tag{5.18}$$

Proof. Similar to Proof of Theorem 5.2, noting that

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D})(\mathbf{E} \otimes \mathbf{F}) = (\mathbf{AC} \otimes \mathbf{BD})(\mathbf{E} \otimes \mathbf{F}) = \mathbf{ACE} \otimes \mathbf{BDF}.$$

as a result of the mixed-product property. $\square$

Other spectral properties which apply to $\mathbf{K}$ include (see 5.13 and 5.14):

$$\operatorname{tr}(\mathbf{K}) = \prod_{d=1}^{D} \operatorname{tr}(\mathbf{K}_d), \tag{5.19}$$

$$\log(\det(\mathbf{K})) = \sum_{d=1}^{D} G_d \log(\det(\mathbf{K}_d)). \tag{5.20}$$

A property in line with that in Equation 5.19 relates the diagonals:

$$\text{diag}(\mathbf{K}) = \bigotimes_{d=1}^{D} \text{diag}(\mathbf{K}_d). \tag{5.21}$$

Efficient GP Regression using Kronecker Products

Armed with the properties of the Kronecker product described in the previous section, we continue our derivation of the `GPR_GRID` algorithm. We first consider the case where the target vector $\mathbf{y}$ contains function values which are not corrupted by noise, turning the regression problem into one of interpolation. The algorithm for interpolation is of little practical interest firstly because we rarely have noise-free data, and secondly because adding a small quantity of "jitter" noise turns out to be essential to avoid numerical problems. Nevertheless, this algorithm forms the foundation for the techniques used when we do have noisy data, and is presented in Algorithm 14.

The `kron_mvprod` and `kron_mmprod` sub-routines

We now outline the details of the `kron_mvprod` $([\mathbf{A}_1, \ldots, \mathbf{A}_D], \mathbf{b})$ call (Algorithm 14, line 7) where it is possible to evaluate

$$\boldsymbol{\alpha} = \left(\bigotimes_{d=1}^{D} \mathbf{A}_d \right) \mathbf{b}, \tag{5.22}$$

in $\mathcal{O}(N)$ time and using $\mathcal{O}(N)$ memory.

Clearly, computing $\boldsymbol{\alpha}$ using standard matrix-vector multiplication is $\mathcal{O}(N^2)$ in runtime and memory, although one can avoid quadratic memory usage by evaluating each row of $\left(\bigotimes_{d=1}^{D} \mathbf{A}_d\right)$ individually using Equation 5.5. Somewhat surprisingly, however, it is possible to attain linear runtime using *tensor algebra* (for an excellent introduction, see Riley et al. [2006]). A brief introduction to tensor algebra is provided in Appendix A. The product in 5.22 can be viewed as a *tensor product* between the tensor $\mathbf{T}^{A}_{i_1,j_1,\ldots,i_D,j_D}$ representing the *outer product* over $[\mathbf{A}_1, \ldots, \mathbf{A}_D]$, and $\mathbf{T}^{B}_{j_D,\ldots,j_1}$ representing the length-N vector $\mathbf{b}$. Conceptually, the aim is to compute

Algorithm 14: Gaussian Process Interpolation on Grid

inputs : Data $\{\mathbf{X} \equiv \mathbf{X}_1 \times \cdots \times \mathbf{X}_D, \mathbf{y}\}$, Test locations $\mathbf{X}_\star$, Covariance Function [`covfunc`, `dcovfunc`], hypers $\theta = [\sigma_f^2; \ell_1 \dots \ell_D]$

outputs: Log-marginal likelihood $\log Z(\theta)$, Derivatives $\frac{\partial \log Z(\theta)}{\partial \theta_i}$, predictive mean $\boldsymbol{\mu}_\star$ and covariance $\boldsymbol{\Sigma}_\star$

for $d \leftarrow 1$ **to** D **do**
  $\mathbf{K}_d \leftarrow$ `covfunc(`$\mathbf{X}_d$, $[\sigma_f^2, \ell_d]$`);` `//Evaluate covariance along dim` d
  $\mathbf{K}_d^{-1} \leftarrow$ `inv(`$\mathbf{K}_d$`);` `//Inverse & determinant`
  $\gamma_d \leftarrow \log(\det(\mathbf{K}_d))$;
  $G_d \leftarrow$ `size(`$\mathbf{K}_d$`);`
end
$\boldsymbol{\alpha} \leftarrow$ `kron_mvprod(`$[\mathbf{K}_1^{-1}, \dots, \mathbf{K}_D^{-1}], \mathbf{y}$`);` `//See text and Algorithm 15`
$\log Z(\theta) \leftarrow -\frac{1}{2}\left(\mathbf{y}^\top \boldsymbol{\alpha} + \sum_{d=1}^D G_d \gamma_d + \frac{N}{2}\log(2\pi)\right)$; `//[5.20]`
for $i \leftarrow 1$ **to** `length(`θ`)` **do**
  for $d \leftarrow 1$ **to** D **do**
    $\frac{\partial \mathbf{K}_d}{\partial \theta_i} \leftarrow$ `dcovfunc(`$\mathbf{X}_d$, $[\sigma_f^2, \ell_d]$, i`);` `//Consistent with 5.29`
    $\delta_d \leftarrow$ `tr(`$\mathbf{K}_d^{-1} \frac{\partial \mathbf{K}_d^\top}{\partial \theta_i}$`);`
  end
  $\boldsymbol{\kappa} \leftarrow$ `kron_mvprod(`$[\frac{\partial \mathbf{K}_1}{\partial \theta_i}, \dots, \frac{\partial \mathbf{K}_D}{\partial \theta_i}], \boldsymbol{\alpha}$`);`
  $\frac{\partial \log Z(\theta)}{\partial \theta_i} \leftarrow \frac{1}{2}\left(\boldsymbol{\alpha}^\top \boldsymbol{\kappa} - \prod_{d=1}^D \delta_d\right)$; `//See text`
end
$[\mathbf{K}_M, \mathbf{K}_{MN}] \leftarrow$ `covfunc(`$\mathbf{X}$, $\mathbf{X}_\star$, θ`);`
$\boldsymbol{\mu}_\star \leftarrow \mathbf{K}_{MN}\boldsymbol{\alpha}$;
$\mathbf{A} \leftarrow$ `kron_mmprod(`$[\mathbf{K}_1^{-1}, \dots, \mathbf{K}_D^{-1}], \mathbf{K}_{NM}$`);` `//See text`
$\boldsymbol{\Sigma}_\star \leftarrow \mathbf{K}_M - \mathbf{K}_{MN}\mathbf{A}$;

a *contraction* over the indices $j_1, \ldots, j_D$, namely:

$$\mathbf{T}^{\alpha} = \sum_{j_1} \cdots \sum_{j_D} \mathbf{T}^{A}_{i_1, j_1, \ldots, i_D, j_D} \mathbf{T}^{B}_{j_D, \ldots, j_1}, \tag{5.23}$$

where $\mathbf{T}^{\alpha}$ is the tensor representing the solution vector $\boldsymbol{\alpha}$. As the sum in Equation 5.23 is over N elements, it will clearly run in $\mathcal{O}(N)$ time! Equivalently, we can express this operation as a sequence of matrix-tensor products and tensor *transpose* operations.

$$\boldsymbol{\alpha} = \text{vec}\left(\left(\mathbf{A}_1 \ldots \left(\mathbf{A}_{D-1}\left(\mathbf{A}_D \mathbf{T}^{B}\right)^{\top}\right)^{\top}\right)^{\top}\right), \tag{5.24}$$

where we define matrix-tensor products of the form $\mathbf{Z} = \mathbf{X}\mathbf{T}$ as:

$$\mathbf{Z}_{i_1 \ldots i_D} = \sum_{k}^{\text{size}(\mathbf{T},1)} \mathbf{X}_{i_1 k} \mathbf{T}_{k i_2 \ldots i_D}. \tag{5.25}$$

The operator $\top$ is assumed to perform a cyclic permutation of the indices of a tensor, namely

$$\mathbf{Y}^{\top}_{i_D i_1 \ldots i_{D-1}} = \mathbf{Y}_{i_1 \ldots i_D}. \tag{5.26}$$

Furthermore, when implementing the expression in Equation 5.24 it is possible to represent the tensors involved using matrices where the first dimension is retained and all other dimensions are collapsed into the second. Thus we can represent $\mathbf{B}$ using a G_d-by-$\prod_{j \neq d} G_j$ matrix. Algorithm 15 gives pseudo-code illustrating these ideas.

`kron_mmprod` $([\mathbf{A}_1, \ldots, \mathbf{A}_D], \mathbf{B}) = \left(\bigotimes_{d=1}^{D} \mathbf{A}_d\right)\mathbf{B}$ is simply an extension of `kron_mvprod` where the right-hand-side is an N-by-M matrix instead of a column vector (we perform `kron_mvprod` M times). As it runs in $\mathcal{O}(NM)$-time and $\mathcal{O}(NM)$-space it is limited to cases where $M \ll N$.

Algorithm 15: `kron_mvprod`

inputs : D matrices $[\mathbf{A}_1 \dots \mathbf{A}_D]$, length-$N$ vector $\mathbf{b}$
outputs: $\boldsymbol{\alpha}$, where $\boldsymbol{\alpha} = \left(\bigotimes_{d=1}^{D} \mathbf{A}_d\right) \mathbf{b}$

$\mathbf{x} \leftarrow \mathbf{b}$;
for $d \leftarrow D$ **to** 1 **do**
    $G_d \leftarrow$ `size(`$\mathbf{K}_d$`)`;
    $\mathbf{X} \leftarrow$ `reshape(`$\mathbf{x}$, G_d, N/G_d`)`;
    $\mathbf{Z} \leftarrow \mathbf{A}_d\mathbf{X}$; `//Matrix-tensor product using matrices only`
    $\mathbf{Z} \leftarrow \mathbf{Z}^\top$; `//Tensor rotation using matrix transpose`
    $\mathbf{x} \leftarrow$ `vec(`$\mathbf{Z}$`)`;
end
$\boldsymbol{\alpha} \leftarrow \mathbf{x}$;

Derivatives

Recall the standard result for the derivative of the log-marginal likelihood with respect to each covariance function hyperparameter:

$$\frac{\partial \log Z(\theta)}{\partial \theta_i} = \frac{1}{2}\left(\boldsymbol{\alpha}^\top \frac{\partial \mathbf{K}}{\partial \theta_i}\boldsymbol{\alpha} - \operatorname{tr}\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}^\top}{\partial \theta_i}\right)\right), \tag{5.27}$$

where $\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y}$. For many tensor product kernels evaluated on a grid, it is also possible to write the derivative matrix $\frac{\partial \mathbf{K}}{\partial \theta_i}$ in Kronecker product form. As a direct result of the product rule for derivatives we have:

$$\frac{\partial \mathbf{K}}{\partial \theta_i} = \sum_{d=1}^{D} \frac{\partial \mathbf{K}_d}{\partial \theta_i} \otimes \left(\bigotimes_{j \neq d} \mathbf{K}_j\right). \tag{5.28}$$

Usually, we have that $\frac{\partial \mathbf{K}_d}{\partial \theta_i} = \mathbf{0}$ for $d \neq i$, as many hyperparameters are specific to a particular dimension d. Thus, the above simplifies to the following Kronecker product:

$$\frac{\partial \mathbf{K}}{\partial \theta_i} = \frac{\partial \mathbf{K}_i}{\partial \theta_i} \otimes \left(\bigotimes_{j \neq i} \mathbf{K}_j\right). \tag{5.29}$$

For example, consider the squared-exponential covariance function for which we had:

$$k(\mathbf{x}_a, \mathbf{x}_b) = \prod_{d=1}^{D} \sigma_f^{2/D} \exp\left(-\frac{\left(\mathbf{x}_a^{(d)} - \mathbf{x}_b^{(d)}\right)^2}{2\ell_d^2} \right). \tag{5.30}$$

It can be shown that[1] :

$$\frac{\partial k(\mathbf{x}_a, \mathbf{x}_b)}{\partial \log(\sigma_f)} = \prod_{d=1}^{D} 2^{1/D} \sigma_f^{2/D} \exp\left(-\frac{\left(\mathbf{x}_a^{(d)} - \mathbf{x}_b^{(d)}\right)^2}{2\ell_d^2} \right), \tag{5.31}$$

and

$$\frac{\partial k(\mathbf{x}_a, \mathbf{x}_b)}{\partial \log(\ell_i)} = \prod_{d=1}^{D} \sigma_f^{2/D} \exp\left(-\frac{\left(\mathbf{x}_a^{(d)} - \mathbf{x}_b^{(d)}\right)^2}{2\ell_d^2} \right) \left(\frac{\left(\mathbf{x}_a^{(d)} - \mathbf{x}_b^{(d)}\right)^2}{\ell_d^2} \right)^{\delta(i,d)}. \tag{5.32}$$

Using Equation 5.29, 5.19 and the mixed-product property we can write:

$$\operatorname{tr}\left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}^\top}{\partial \theta_i} \right) = \operatorname{tr}\left(\mathbf{K}_i^{-1} \frac{\partial \mathbf{K}_i^\top}{\partial \theta_i} \right) \prod_{j \neq i} \operatorname{tr}\left(\mathbf{K}_j^{-1} \mathbf{K}_j^\top \right). \tag{5.33}$$

As we can evaluate the term $\boldsymbol{\alpha}^\top \frac{\partial \mathbf{K}}{\partial \theta_i} \boldsymbol{\alpha}$ efficiently using `kron_mvprod` it is clear that, given the derivative matrix is such that Equation 5.29 holds, we can compute all necessary derivatives in linear-time and linear-memory.

Adding Spherical Noise

Although we can write $\mathbf{K}^{-1}$ as a Kronecker product, the same cannot be said for $(\mathbf{K} + \sigma_n^2 I_N)^{-1}$. This is because $(\mathbf{K} + \sigma_n^2 I_N)$ cannot itself be written as Kronecker product, due to the perturbation on the main diagonal. Nevertheless, it is possible to sidestep this problem using the eigendecomposition properties presented in Theorem 5.3. Furthermore, since we can write $\mathbf{K} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top$, with $\mathbf{Q}$ and $\boldsymbol{\Lambda}$ given by Equations 5.17 and 5.18 respectively, it is possible to solve the linear system $(\mathbf{K} + \sigma_n^2 I_N)^{-1} \mathbf{y}$

[1] Computing derivatives with respect to the *log* of the hyperparameter avoids the need to deal with the hyperparameter positivity constraint.

using the identity:

$$\left(\mathbf{K} + \sigma_n^2 I_N\right)^{-1} \mathbf{y} = \mathbf{Q} \left(\mathbf{\Lambda} + \sigma_n^2 I_N\right)^{-1} \mathbf{Q}^\top \mathbf{y}. \tag{5.34}$$

Algorithm 16 shows how Equation 5.34 is used to perform exact GP regression over a grid of inputs. On line 8 we use the fact that $\det(\mathbf{K} + \sigma_n^2 I_N)$ is the product of the eigenvalues on the main diagonal of $\mathbf{\Lambda} + \sigma_n^2$. On lines 12 and 16 we compute the tricky quantity $\operatorname{tr}\left(\left(\mathbf{K} + \sigma_n^2 I_N\right)^{-1} \frac{\partial \mathbf{K}^\top}{\partial \theta_i}\right)$ by appealing to the *cyclic property* of the trace operator which maintains that the trace is invariant under cyclic permutations, i.e., for appropriately-sized matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$:

$$\operatorname{tr}\left(\mathbf{ABCD}\right) = \operatorname{tr}\left(\mathbf{DABC}\right) = \operatorname{tr}\left(\mathbf{CDAB}\right) = \operatorname{tr}\left(\mathbf{BCDA}\right). \tag{5.35}$$

We can therefore write the following:

$$\begin{aligned}
\operatorname{tr}\left(\left(\mathbf{K} + \sigma_n^2 I_N\right)^{-1} \frac{\partial \mathbf{K}^\top}{\partial \theta_i}\right) &= \operatorname{tr}\left(\mathbf{Q}\left(\mathbf{\Lambda} + \sigma_n^2 I_N\right)^{-1} \mathbf{Q}^\top \frac{\partial \mathbf{K}^\top}{\partial \theta_i}\right) \\
&= \operatorname{tr}\left(\left(\mathbf{\Lambda} + \sigma_n^2 I_N\right)^{-1} \mathbf{Q}^\top \frac{\partial \mathbf{K}^\top}{\partial \theta_i} \mathbf{Q}\right) \\
&= \operatorname{diag}\left(\left(\mathbf{\Lambda} + \sigma_n^2 I_N\right)^{-1}\right)^\top \operatorname{diag}\left(\mathbf{Q}^\top \frac{\partial \mathbf{K}^\top}{\partial \theta_i} \mathbf{Q}\right),
\end{aligned}$$

where the last step is valid because the matrix $\left(\mathbf{\Lambda} + \sigma_n^2 I_N\right)^{-1}$ is diagonal. The term $\operatorname{diag}\left(\mathbf{Q}^\top \frac{\partial \mathbf{K}^\top}{\partial \theta_i} \mathbf{Q}\right)$ can be computed in $\mathcal{O}(N)$ runtime and $\mathcal{O}(N)$ memory using the property in Equation 5.21 and noting that:

$$\mathbf{Q}^\top \frac{\partial \mathbf{K}^\top}{\partial \theta_i} \mathbf{Q} = \mathbf{Q}_i^\top \frac{\partial \mathbf{K}_i^\top}{\partial \theta_i} \mathbf{Q}_i \otimes \left(\bigotimes_{j \neq i}^{D} \mathbf{Q}_j^\top \mathbf{K}_j \mathbf{Q}_j\right), \tag{5.36}$$

as a result of Equation 5.29.

Algorithm 16 gives pseudo-code of GP regression with spherical noise. It implements exactly the same operations as those given in Algorithm 1, but in $\mathcal{O}(N)$ runtime and space!

Algorithm 16: Gaussian Processs Regression on Grid

inputs : Data $\{\mathbf{X} \equiv \mathbf{X}_1 \times \cdots \times \mathbf{X}_D, \mathbf{y}\}$, Test locations $\mathbf{X}_\star$, Covariance Function [`covfunc`, `dcovfunc`], hypers $\theta = [\ell_1 \dots \ell_D; \sigma_f^2; \sigma_n^2]$

outputs: Log-marginal likelihood $\log Z(\theta)$, Derivatives $\frac{\partial \log Z(\theta)}{\partial \theta_i}$, predictive mean $\boldsymbol{\mu}_\star$ and covariance $\boldsymbol{\Sigma}_\star$

for $d \leftarrow 1$ **to** D **do**
  $\mathbf{K}_d \leftarrow$ `covfunc`$(\mathbf{X}_d, [\sigma_f^2, \ell_d])$; `//Evaluate covariance along dim` d
  $[\mathbf{Q}_d, \boldsymbol{\Lambda}_d] \leftarrow$ `eig`$(\mathbf{K}_d)$; `//Eigendecomposition`
end
$\boldsymbol{\alpha} \leftarrow$ `kron_mvprod`$([\mathbf{Q}_1^\top, \dots, \mathbf{Q}_D^\top], \mathbf{y})$;
$\boldsymbol{\alpha} \leftarrow (\boldsymbol{\Lambda} + \sigma_n^2 I_N)^{-1} \boldsymbol{\alpha}$;
$\boldsymbol{\alpha} \leftarrow$ `kron_mvprod`$([\mathbf{Q}_1, \dots, \mathbf{Q}_D], \boldsymbol{\alpha})$; `//`$\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y}$
$\log Z(\theta) \leftarrow -\frac{1}{2}\left(\mathbf{y}^\top \boldsymbol{\alpha} + \sum_{i=1}^N \log(\boldsymbol{\Lambda}(i,i) + \sigma_n^2) + \frac{N}{2}\log(2\pi)\right)$;

`//Derivatives`
for $i \leftarrow 1$ **to** `length`(θ) **do**
  for $d \leftarrow 1$ **to** D **do**
    $\frac{\partial \mathbf{K}_d}{\partial \theta_i} \leftarrow$ `dcovfunc`$(\mathbf{X}_d, [\sigma_f^2, \ell_d], i)$; `//Consistent with Eq.` 5.36
    $\boldsymbol{\gamma}_d \leftarrow$ `diag`$(\mathbf{Q}_d^\top \frac{\partial \mathbf{K}_d^\top}{\partial \theta_i} \mathbf{Q}_d)$;
  end
  $\boldsymbol{\gamma} \leftarrow \bigotimes_{d=1}^D \boldsymbol{\gamma}_d$;
  $\boldsymbol{\kappa} \leftarrow$ `kron_mvprod`$([\frac{\partial \mathbf{K}_1}{\partial \theta_i}, \dots, \frac{\partial \mathbf{K}_D}{\partial \theta_i}], \boldsymbol{\alpha})$;
  $\frac{\partial \log Z(\theta)}{\partial \theta_i} \leftarrow \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{\kappa} - \frac{1}{2}$ `sum`$((\boldsymbol{\Lambda} + \sigma_n^2 I_N)^{-1} \boldsymbol{\gamma})$;
end

`//Test set predictions`
$[\mathbf{K}_M, \mathbf{K}_{MN}] \leftarrow$ `covfunc`$(\mathbf{X}, \mathbf{X}_\star, \theta)$;
$\boldsymbol{\mu}_\star \leftarrow \mathbf{K}_{MN}\boldsymbol{\alpha}$;
$\mathbf{A} \leftarrow$ `kron_mmprod`$([\mathbf{Q}_1^\top, \dots, \mathbf{Q}_D^\top], \mathbf{K}_{NM})$;
$\mathbf{A} \leftarrow (\boldsymbol{\Lambda} + \sigma_n^2 I_N)^{-1} \mathbf{A}$;
$\mathbf{A} \leftarrow$ `kron_mmprod`$([\mathbf{Q}_1, \dots, \mathbf{Q}_D])\mathbf{A}$;
$\boldsymbol{\Sigma}_\star \leftarrow \mathbf{K}_M - \mathbf{K}_{MN}\mathbf{A}$;

5.3 Results

5.3.1 Runtimes

Figure 5.1 illustrates the magnitude of improvement in runtime obtained by exploiting the structure present in a covariance matrix that can be written as a Kronecker product. The runtime of Algorithm 16 for N greater than a million points is *much less* than the runtime of Algorithm 1 for $N = 4096$! These runtimes were obtained by running both these algorithms for a fixed set of hyperparameters given by setting all the lengthscales and the signal amplitude to 1 and the noise variance to 0.01. The input locations were given by $[-1, 1]^D$, thus $N = 2^D$. The noise-free function values were synthetically generated by using the `kron_mvprod` routine with $\{L_1, \ldots, L_D\}$ and an N-vector of standard Gaussian random variables, where L_d is the (2-by-2) Cholesky factor of the axis-aligned covariance matrix for dimension d (recall Theorem 5.2). These draws were then corrupted by adding IID Gaussian noise. The hyperparameters used for synthetic data generation were set to be equal to the hyperparameters used for runtime calculations.

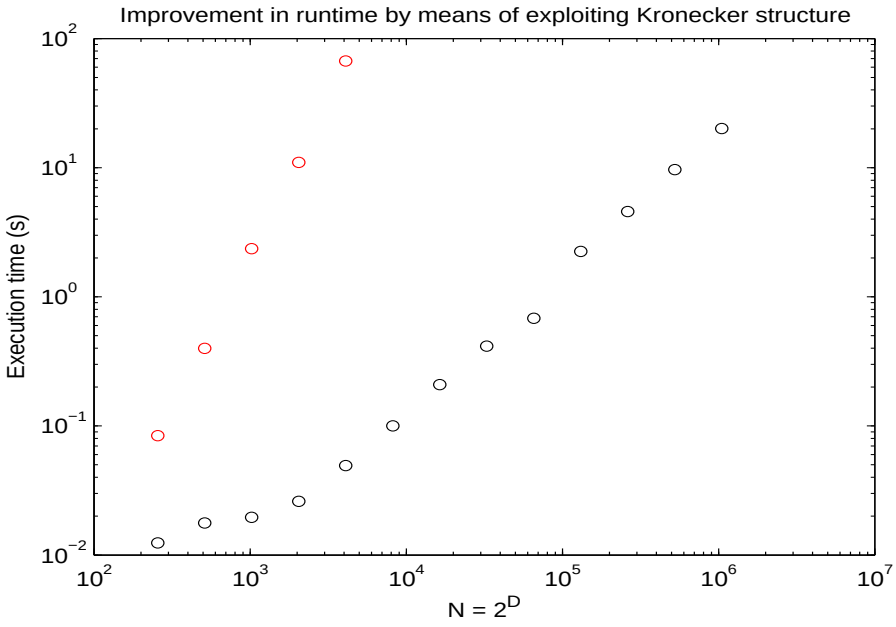


Figure 5.1: Runtimes of Algorithm 1 (in red) and Algorithm 16 (in black) for $N = [2^8, 2^9, \ldots, 2^{20}]$. The slope for Algorithm 1 is 2.6 and that of Algorithm 16 is 0.97. This empirically verifies the improvement to linear scaling.

5.3.2 Example Application: Record Linkage Dataset

An excellent example of a dataset where the inputs lie on a Cartesian grid is given by the Record Linkage Dataset, which can be found at: http://archive.ics.uci.edu/ml/datasets/Record+Linkage+Comparison+Patterns. This dataset, introduced in Sariyar et al. [2011], consists of 5,749,132 record pairs of which 20,931 are *matches*, i.e. they are two separate records belonging to the same person. Each record pair is described by a set of features encoding phonetic equality of typical personal data, such as first name, family name, date of birth, gender and so on. These give rise to 11 features of which 9 are predictive of a match. Of these 9 features, 2 features are rarely present, so we only consider the 7 features which are rarely missing. Crucially, these features are usually very coarsely *discretized*. The feature corresponding to a match of the first names takes 54 unique values and the one corresponding to a match of family names takes 101 unique values. All other features are *binary* where 0 indicates complete mismatch and 1 indicates a full match. This gives a total grid size of 174,528. As the number of examples is far greater than the grid size it is obvious that many targets are observed for individual inputs. As a result, we can convert the original binary classification problem into a regression problem with *input-dependent* noise. For each grid location $\mathbf{x}_g$ we can construct the corresponding target and noise variance using a Beta-Bernoulli hierarchy with a Beta(1,1) prior, i.e.:

$$y(\mathbf{x}_g) = \frac{\#p + 1}{\#p + \#n + 2}, \tag{5.37}$$

$$\sigma_n^2(\mathbf{x}_g) = \frac{(\#p\#n)}{(((\#p + \#n)^2)(\#p + \#n + 1))}, \tag{5.38}$$

where $\#p$ and $\#n$ are the number of positive and negative examples falling into bin $\mathbf{x}_g$.

Adding Diagonal Noise

Dealing with non-spherical noise poses a significant challenge to GPR on a grid, since the effect of diagonal noise components on the eigendecomposition of the kernel

matrix is highly non-trivial[1], unlike the case of adding spherical noise which simply adds a constant to all eigenvalues and leaves eigenvectors unaltered. In fact, it is likely to add another factor of N to the runtime. Note, however, that for the purposes of this example application, the majority of elements of the noise vector (given by Equation 5.38) will correspond to a grid location with no observations, and therefore to a fixed (indeed, maximal) value. Therefore, it is sensible to first run Algorithm 16 with spherical noise set to the maximal noise variance and then correct for the entries with at least one observation. Notice that we would like to be able to solve the following linear system:

$$\left(\mathbf{Q}(\mathbf{\Lambda} + \max(\sigma_n^2(\cdot))\mathbf{I}_N)\mathbf{Q}^\top + \sum_{i \in C} \rho_i \mathbf{e}_i \mathbf{e}_i^\top\right) \boldsymbol{\alpha} = \mathbf{y}, \tag{5.39}$$

where C is the set of indices to be corrected, $\mathbf{e}_i$ is the i^{th} unit vector, and, by definition, $\rho_i < 0$. This can be accomplished by initialising $\boldsymbol{\alpha}$ to be the solution of the spherical noise system, i.e.,

$$\boldsymbol{\alpha} \leftarrow \underbrace{\mathbf{Q}(\mathbf{\Lambda} + \max(\sigma_n^2(\cdot))\mathbf{I}_N)^{-1}\mathbf{Q}^\top}_{\equiv \mathbf{K}^{-1}} \mathbf{y}, \tag{5.40}$$

and updating it in light of each correction, as follows (see Press et al. [2007] for a derivation):

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \frac{\mathbf{e}_i^\top \boldsymbol{\alpha}}{1 + \rho_i \mathbf{e}_i^\top \mathbf{K}^{-1} \mathbf{e}_i} \mathbf{K}^{-1} \mathbf{e}_i. \tag{5.41}$$

In a similar fashion we can compute the log-determinant λ of the matrix in Equation 5.39 by first assigning:

$$\lambda \leftarrow \log(\det(\mathbf{K})), \tag{5.42}$$

and updating it for each correction, using:

$$\lambda \leftarrow \lambda + \log\left(1 + \rho_i \mathbf{e}_i^\top \mathbf{K}^{-1} \mathbf{e}_i\right). \tag{5.43}$$

Using both the solution vector $\boldsymbol{\alpha}$ and λ we can compute the marginal likelihood. Every diagonal correction adds $\mathcal{O}(N)$ computation to the load, as is evident from

[1] In fact, this is an open research question, see Knutson and Tao [2000]

Equation 5.41. Fortunately, in a 7-dimensional space most input locations on the grid will not be observed, thus $|C| \ll N$ and the overall runtime complexity will still be closer to linear than it is to quadratic. For example, in this particular application $|C| = 12,186$. One could solve the system in Equation 5.39 using the *conjugate gradients* algorithm also, however, since the size of this linear system is so large, it is typical to find that more iterations of conjugate gradients (each with $\mathcal{O}(N)$ computation) is necessary to achieve the same accuracy as the correction-based approach.

Note that the computation of the derivatives of the marginal likelihood is also complicated by the addition of diagonal noise. For this particular dataset we will perform hyperparameter learning by constraining the lengthscales in each dimension to be equal and doing a grid search over the singleton lengthscale and signal amplitude parameters.

Performance Comparison

The *Record Linkage* dataset is so large that it is not possible to run many of the standard algorithms one would ordinarily run to perform comparisons, including the SVM, the IVM and SPGP (these were used extensively in Chapter 4). Even fitting the additive GP classifier of Chapter 4 proved troublesome – although certain changes to the code can be made to improve efficiency in this case.

As can be seen in Table 5.1, we have compared the algorithm described above (referred to as `GPR_GRID`) to logistic regression and to simple histogramming, which uses the targets in Equation 5.37 directly to make predictions for unseen points (which will land in the same set of grid locations). Thus, we would expect the `GPR_GRID` algorithm to perform better than **Histogram**, as it predicts in exactly the same way, except that it additionally smoothes the targets. We have also compared the performance of the nearest-neighbour algorithm. Note that, without the use of kd-trees (see Bentley [1975]), we would not be able to run nearest-neighbours in a reasonable amount of time.

For all these methods we train on a randomised 80% subset of the full dataset and compute the test error and MNLP (see Chapter 4) on the remaining 20%.

Table 5.1: Performance on the very large *Record Linkage* dataset. The runtime for `GPR_GRID` does not include the time it took to perform the grid search to find the optimal hyperparameters. These were found to be $\log(\ell) = -2$, $\log(\sigma_f) = 0$. The timing is for the hyperparameters set to these values. The size of the dataset becomes apparent from the runtime of the simple *nearest neighbour* algorithm, implemented efficiently using kd-trees.

Algorithm	Error Rate	MNLL	Runtime (s)
Histogram	5.327×10^{-4}	-17.4	**2**
Logistic Regression	0.148×10^{-4}	-21.6	9
GPR_GRID	$\mathbf{0.0889 \times 10^{-4}}$	$\mathbf{-23.1}$	246
Nearest Neighbour	0.122×10^{-4}	N/A	24,252

Chapter 6

Conclusions

A number of noteworthy conclusions can be drawn from the findings in this thesis.

Firstly, it is reasonable to expect that most real-world scalar GP regression tasks can be tackled using the techniques in chapter 2, perhaps in conjunction with Toeplitz matrix methods. Thus, one should no longer assume that generalised GP regression over a scalar input space suffers from the same complexity as the generic GP regression algorithm. Indeed, this can be viewed as one of the central messages this thesis has attempted to convey. The fact that inference and learning can be achieved so efficiently unlocks the potential for many exciting extensions. For example, we have only considered generalised regression where the likelihood has been assumed to factorise across the observations. There are many applications where a more complex likelihood would be appropriate, an example being inference for the *Cox process* where we model the underlying rate function with a GP. The Cox process, or *doubly-stochastic* Poisson process was first introduced in Cox [1955]. Inference for such a model was studied extensively in Adams [2009], where the runtime complexity is cubic in N, per MCMC iteration. We conjecture that chapter 2 can form the foundation of alternative techniques which scale better in the frequently-encountered case where time is the only input dimension. Another exciting avenue is inference and learning for (temporal) Markov processes which are *non-Gaussian*. Recall that when running EP on Gauss-Markov processes, the forward and backward messages did not require a projection. This would not be the case for Markov processes, which therefore pose an interesting challenge to the EP algorithm. Approximate inference based on variational methods for Markov pro-

cesses has already been studied in Archambeau et al. [2008] and Archambeau and Opper [2011]. Chapter 2 provides a solid foundation for construction of scalable alternatives to these techniques. Another useful extension involves the use of the state-space model representation of GPs for the purpose of improving the efficiency of algorithms which tackle the case where there are *multiple* temporal latent GPs giving rise to *multivariate* targets. This is the subject of many methods including, but not limited to, sets of dependent temporal GPs Boyle and Frean [2005], latent force models Alvarez et al. [2009] and the (temporal) semiparametric latent factor model Teh et al. [2005].

In chapter 3 we saw how to exploit nonstationarity in a sequential data stream for the purposes improving predictive accuracy and runtime. The improvement in runtime will usually become apparent only in the case where we cannot represent the underlying GP kernel as a SSM, and inputs are continuous. In contrast, improvements in predictive accuracy are generally to be expected, especially if it is known, a priori, that the time series does indeed exhibit nonstationarity. We restricted attention to online inference and prediction, so an obvious extension is to consider the case where we do not need an online technique and extend the basic BOCPD algorithm so that we can perform *smoothing* in addition to filtering, in order to calculate the exact posterior runlength distribution, in a manner similar to Chib [1998]. Another interesting extension would be to combine the methods of chapter 3 with those of chapter 4 to construct an additive GP model which is naturally formed of independent additive models in different partitions of the D-dimensional input space. Indeed, this approach would have similarities to that in Gramacy [2005], albeit where the GP prior is assumed to be additive.

The main message of chapter 4 is that one of the simplest ways to extend the efficiency of a scalar GP method to higher dimensional input spaces, is through the additivity assumption. As long as we are summing a set of scalar GPs, irrespective of whether we use the original inputs or features derived from these inputs, we can expect to see significant computational savings. From a modelling perspective, the most interesting algorithm presented in this chapter is projection-pursuit GP regression, as it can be seen as a feed-forward neural network (see Bishop [2007]) where each activation function is a scalar GP! Chapter 4 presented a greedy algorithm for learning network weights and the GP-based activation functions in $\mathcal{O}(N \log N)$

runtime and $\mathcal{O}(N)$ space, which is impressively efficient for such a flexible model. Of course, this begs the question of whether there are better (non-greedy) ways to optimise the PPGPR model, perhaps similar in spirit to the original backpropagation algorithm of Rumelhart et al. [1986]. In addition, for PPGPR to be more widely used, it is crucial to be able to extend it to handle non-Gaussian likelihoods. Thus, another interesting extension would involve attempts to combine it with approximate inference techniques such as EP or Laplace's approximation. Note that we actually derived the additive GP classification algorithm by means of the Laplace approximation. For the purposes of classication there is consensus (see Nickisch and Rasmussen [2008]) that EP gives superior performance. However, the EP updates required for the additive GP can be shown to be intractable due to the coupling between the univariate functions induced by observing the targets. It would, nonetheless, be interesting to see if EP itself can be adapted to produce an alternative framework for generalised additive GP regression. Finally, note that the additivity assumption may not be the only way to extend efficiency to the case where we have multivariate inputs. A promising alternative may involve the use of *space-time* Gauss-Markov processes, for which inference may be performed using stochastic *PDEs*. Whether or not this results in improved efficiency is an open problem, and is introduced in Särkkä [2011].

The final chapter demonstrates the rather surprising result that GPs defined on multidimensional grids are actually easy to handle computationally! The main caveat is that the grid blows up in size as D increases, thus constraining applications to live over lower-dimensional inputs spaces. The first way to sidestep this problem is to study the case where inputs live on a *"subgrid"*. As of writing, no efficient algorithm is known for GP regression on subgrids. Thus, this can be viewed as an interesting open problem. The second way is to assume that we have low-dimensional latent *pseudo-inputs* and run a GP model efficiently using these inputs instead. Recall from Snelson and Ghahramani [2006] that we can integrate out the pseudo-targets associated with these inputs, and optimise the grid locations with respect to the marginal likelihood.

Appendix A

Mathematical Background

A.1 Matrix Identities

Here, we list some specific matrix identities which are used frequently in this thesis and which may not appear in any standard linear algebra text book.

- Let $\mathbf{P}$ be a square matrix of size N and $\mathbf{R}$ be a square matrix of size M. Then,

$$\left(\mathbf{P}^{-1} + \mathbf{B}^\top \mathbf{R}^{-1} \mathbf{B}\right)^{-1} \mathbf{B}^\top \mathbf{R}^{-1} = \mathbf{P}\mathbf{B}^\top \left(\mathbf{B}\mathbf{P}\mathbf{B}^\top + \mathbf{R}\right)^{-1}, \tag{A.1}$$

  where $\mathbf{B}$ is $M \times N$. In the special case where $M = N$ and $\mathbf{B} = \mathbf{I}_N$, we obtain:

$$\left(\mathbf{P}^{-1} + \mathbf{R}^{-1}\right)^{-1} \mathbf{R}^{-1} = \mathbf{P} \left(\mathbf{P} + \mathbf{R}\right)^{-1}. \tag{A.2}$$

- Another useful identity is the Woodbury formula, which holds for arbitrary matrices $\mathbf{A}, \mathbf{U}, \mathbf{W}, \mathbf{V}$ of appropriate sizes:

$$\left(\mathbf{A} + \mathbf{U}\mathbf{W}\mathbf{V}^\top\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U} \left(\mathbf{W}^{-1} + \mathbf{V}^\top \mathbf{A}^{-1} \mathbf{U}\right)^{-1} \mathbf{V}\mathbf{A}^{-1}. \tag{A.3}$$

  This comes in handy if $\mathbf{A}$ is easy to invert, and $(\mathbf{W}^{-1} + \mathbf{V}^\top \mathbf{A}^{-1}\mathbf{U})$ is much smaller than the matrix on the left-hand-side. It is also useful if we already know $\mathbf{A}^{-1}$ and would like to compute the inverse of a low-rank update to $\mathbf{A}$ efficiently.

- The Woodbury formula has an analogue for determinants, namely:

$$\det\left(\mathbf{A}+\mathbf{U}\mathbf{W}\mathbf{V}^{\top}\right)=\det(\mathbf{W}^{-1}+\mathbf{V}^{\top}\mathbf{A}^{-1}\mathbf{U})\det(\mathbf{W})\det(\mathbf{A}). \tag{A.4}$$

- Blockwise matrix inversion can be performed analytically:

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1}=\begin{bmatrix}\mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\\ -(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\end{bmatrix}. \tag{A.5}$$

  for appropriately size matrices $\mathbf{A},\mathbf{B},\mathbf{C}$ and $\mathbf{D}$. Note that $\mathbf{A}$ and $\mathbf{D}$ need to be square matrices for this identity to be valid. This is useful when we have already computed $\mathbf{A}^{-1}$, and would like to know the inverse of the same matrix but with a couple of rows/columns added in. This situation arises for the online GP where we keep on adding new rows (and associated columns) to the kernel matrix, and require its inverse for next-step predictions. In practice, it is necessary to avoid explicity computing the inverse (due to high numerical instability), and instead work with "rank-one updates" of solution vectors of the form $\mathbf{K}^{-1}\mathbf{y}$ and with Cholesky decompositions.

- Using Equation A.5, we can derive the rank-one update equation for GP kernels:

$$\mathbf{K}_{N+1}^{-1}\equiv\begin{bmatrix}\mathbf{K}_N & \mathbf{k}_\star\\ \mathbf{k}_\star^{\top} & k_{\star,\star}\end{bmatrix}^{-1}=\begin{bmatrix}\mathbf{K}_N^{-1}+\eta^{-1}\mathbf{v}\mathbf{v}^{\top} & -\eta^{-1}\mathbf{v}\\ -\eta^{-1}\mathbf{v}^{\top} & \eta^{-1}\end{bmatrix}, \tag{A.6}$$

  where $\eta\equiv k_{\star,\star}-\mathbf{k}_\star^{\top}\mathbf{K}_N^{-1}\mathbf{k}_\star$ and $\mathbf{v}\equiv\mathbf{K}_N^{-1}\mathbf{k}_\star$. Thus, given that we already have computed the solution $\boldsymbol{\alpha}_N\equiv\mathbf{K}_N^{-1}\mathbf{y}_N$, $\boldsymbol{\alpha}_{N+1}$ can be computed straightforwardly using:

$$\boldsymbol{\alpha}_{N+1}=\begin{bmatrix}\boldsymbol{\alpha}_N+\eta^{-1}\left[\left(\mathbf{v}^{\top}\mathbf{y}_N-y_{N+1}\right)\mathbf{v}\right]\\ \eta^{-1}\left[y_{N+1}-\mathbf{v}^{\top}\mathbf{y}_N\right]\end{bmatrix}. \tag{A.7}$$

  Given $\mathbf{L}_N$ is the lower Cholesky factor of $\mathbf{K}_N$, the lower Cholesky factor of $\mathbf{K}_{N+1}$ is given by:

$$\mathbf{L}_{N+1}=\begin{bmatrix}\mathbf{L}_N & \mathbf{0}\\ \mathbf{t} & \tau\end{bmatrix}, \tag{A.8}$$

where $\mathbf{t} \equiv \mathbf{L}_N^{-1}\mathbf{k}_\star$, and $\tau \equiv \left(k_{\star,\star} - \mathbf{t}^\top\mathbf{t}\right)^{1/2}$.

- An analogue of Equation A.5 also exists for determinants:

$$\det\begin{pmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{pmatrix} = \det(\mathbf{A})\det(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}). \tag{A.9}$$

  In particular, for the rank-one updates used by the online GP, we have:

$$\det\begin{pmatrix}\mathbf{K}_N & \mathbf{k}_\star\\ \mathbf{k}_\star^\top & k_{\star,\star}\end{pmatrix} = \det(\mathbf{K}_N)\det(k_{\star,\star} - \mathbf{k}_\star^\top\mathbf{K}_N^{-1}\mathbf{k}_\star). \tag{A.10}$$

A.2 Gaussian Identities

A.2.1 Marginalisation and Conditioning

Let

$$p\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_\mathbf{x}\\ \boldsymbol{\mu}_\mathbf{y}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} & \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{y}}\\ \boldsymbol{\Sigma}_{\mathbf{y},\mathbf{x}} & \boldsymbol{\Sigma}_{\mathbf{y},\mathbf{y}}\end{bmatrix}\right), \tag{A.11}$$

where the cross-covariance term is symmetric, i.e., $\boldsymbol{\Sigma}_{\mathbf{y},\mathbf{x}} = \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{y}}^\top$. Then, the marginals of this joint Gaussian distribution are (trivially) given by:

$$\int p(\mathbf{x},\mathbf{y})d\mathbf{y} \equiv p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}}), \tag{A.12}$$

$$\int p(\mathbf{x},\mathbf{y})d\mathbf{x} \equiv p(\mathbf{y}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{y}, \boldsymbol{\Sigma}_{\mathbf{y},\mathbf{y}}). \tag{A.13}$$

The conditionals are also Gaussian:

$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\mu}_\mathbf{x} + \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y},\mathbf{y}}^{-1}\left(\mathbf{y} - \boldsymbol{\mu}_\mathbf{y}\right), \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} - \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y},\mathbf{y}}^{-1}\boldsymbol{\Sigma}_{\mathbf{y},\mathbf{x}}\right), \tag{A.14}$$

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}\left(\boldsymbol{\mu}_\mathbf{y} + \boldsymbol{\Sigma}_{\mathbf{y},\mathbf{x}}\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}}^{-1}\left(\mathbf{x} - \boldsymbol{\mu}_\mathbf{x}\right), \boldsymbol{\Sigma}_{\mathbf{y},\mathbf{y}} - \boldsymbol{\Sigma}_{\mathbf{y},\mathbf{x}}\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}}^{-1}\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{y}}\right). \tag{A.15}$$

Equations A.11 through to A.15 provide all one needs to know in order to perform inference in any linear-Gaussian model. Such models are abundant in machine learning, ranging from Bayesian linear regression to inference in linear dynamical

systems and probabilistic principal components analysis. In particular, $\mathbf{x}$ are assumed to represent latent variables (or parameters) and $\mathbf{y}$ the observed variables under the following generative model:

$$p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{A.16}$$

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{N}). \tag{A.17}$$

Thus, for this case, Equation A.11 becomes:

$$p\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \mathbf{A}\boldsymbol{\mu} + \mathbf{b}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma} & \boldsymbol{\Sigma}\mathbf{A}^\top\\ \mathbf{A}\boldsymbol{\Sigma} & \mathbf{N} + \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top\end{bmatrix}\right), \tag{A.18}$$

from which we can straightforwardly derive the posterior $p(\mathbf{x}|\mathbf{y})$ and marginal likelihood terms $p(\mathbf{y})$, using Equations A.13 and A.14. These form the key quantities for Bayesian inference and learning as applied to the linear-Gaussian model.

A.2.2 Multiplication and Division

The multiplication and division of multivariate Gaussian factors of the same dimensionality D is central to approximate inference algorithms such as EP. The multiplication of two Gaussian factors results in another, unnormalised Gaussian. This can be easily seen as a result of "completing the square" in the exponent of the product of Gaussians.

$$\mathcal{N}(\mathbf{a}, \mathbf{A})\mathcal{N}(\mathbf{b}, \mathbf{B}) \propto \mathcal{N}(\mathbf{c}, \mathbf{C}), \tag{A.19}$$

where

$$\mathbf{C} = \left(\mathbf{A}^{-1} + \mathbf{B}^{-1}\right)^{-1}, \tag{A.20}$$

$$\mathbf{c} = \mathbf{C}\mathbf{A}^{-1}\mathbf{a} + \mathbf{C}\mathbf{B}^{-1}\mathbf{b}. \tag{A.21}$$

The normalisation constant Z of the left-hand-side of Equation A.19 can be derived by calculating the constant required to equate both sides. It can further be shown

that it is a *Gaussian* function of either $\mathbf{a}$ or $\mathbf{b}$:

$$Z = (2\pi)^{-D/2}|\mathbf{C}|^{+1/2}|\mathbf{A}|^{-1/2}|\mathbf{B}|^{-1/2}\exp\left[-\frac{1}{2}\left(\mathbf{a}^\top\mathbf{A}^{-1}\mathbf{a} + \mathbf{b}^\top\mathbf{B}^{-1}\mathbf{b} - \mathbf{c}^\top\mathbf{C}^{-1}\mathbf{c}\right)\right]. \tag{A.22}$$

The division of a two Gaussian factors also results in an unnormalised Gaussian, given that the resulting covariance is positive definite. Note that the division of two Gaussians may result in a "Gaussian" function with negative-definite covariance.

$$\frac{\mathcal{N}(\mathbf{a}, \mathbf{A})}{\mathcal{N}(\mathbf{b}, \mathbf{B})} \propto \mathcal{N}(\mathbf{c}, \mathbf{C}), \tag{A.23}$$

where

$$\mathbf{C} = \left(\mathbf{A}^{-1} - \mathbf{B}^{-1}\right)^{-1}, \tag{A.24}$$

$$\mathbf{c} = \mathbf{C}\mathbf{A}^{-1}\mathbf{a} - \mathbf{C}\mathbf{B}^{-1}\mathbf{b}. \tag{A.25}$$

The normalisation constant of the left-hand-side is given by:

$$Z = (2\pi)^{+D/2}|\mathbf{C}|^{+1/2}|\mathbf{A}|^{-1/2}|\mathbf{B}|^{+1/2}\exp\left[-\frac{1}{2}\left(\mathbf{a}^\top\mathbf{A}^{-1}\mathbf{a} - \mathbf{b}^\top\mathbf{B}^{-1}\mathbf{b} - \mathbf{c}^\top\mathbf{C}^{-1}\mathbf{c}\right)\right]. \tag{A.26}$$

A.2.3 Gaussian Expectation Propagation

In this thesis, we are only concerned with EP approximations which are fully-factorised and Gaussian. For this particular version of EP, the central factor update equation, given in Equation 2.73 and reiterated below, is implemented using only Gaussian multiplications, divisions and projections.

$$\forall k \in \mathrm{ne}(\phi_i) \tag{A.27}$$

$$\tilde{\phi}_{ik}^{\mathrm{new}}(\mathbf{z}_k) \propto \frac{\mathrm{proj}\left[\overbrace{\prod_{\substack{j\in\mathrm{ne}(\mathbf{z}_k)\\ j\neq i}} \tilde{\phi}_{jk}^{\mathrm{old}}(\mathbf{z}_k)}^{\propto\mathcal{N}(\mathbf{z}_k;\boldsymbol{\mu}_{\mathrm{cav}},\mathbf{V}_{\mathrm{cav}})} \overbrace{\int_{\mathbf{Z}_{\neg k}} \phi_i(\mathbf{Z}_{\mathrm{ne}(\phi_i)}) \prod_{\ell\in\mathrm{ne}(\mathbf{Z}_{\neg k})} \prod_{m\in\neg k} \tilde{\phi}_{\ell m}^{\mathrm{old}}(\mathbf{z}_m)\, d\mathbf{Z}_{\neg k}}^{\equiv f(\mathbf{z}_k)}\right]}{\underbrace{\prod_{\substack{j\in\mathrm{ne}(\mathbf{z}_k)\\ j\neq i}} \tilde{\phi}_{jk}^{\mathrm{old}}(\mathbf{z}_k)}_{\propto\mathcal{N}(\mathbf{z}_k;\boldsymbol{\mu}_{\mathrm{cav}},\mathbf{V}_{\mathrm{cav}})}}.$$

The "cavity distribution", given by $\mathcal{N}(\mathbf{z}_k; \boldsymbol{\mu}_{\mathrm{cav}}, \mathbf{V}_{\mathrm{cav}})$ can be computed by multiplying all the constituent Gaussians using Equations A.20 and A.21, and normalising the result. As one usually maintains the (approximated) marginals over $\mathbf{z}_k$, i.e., $\prod_{j\in\mathrm{ne}(\mathbf{z}_k)} \tilde{\phi}_{jk}^{\mathrm{old}}(\mathbf{z}_k)$, the cavity distribution can also be computed via *dividing* the marginal by $\tilde{\phi}_{ik}^{\mathrm{old}}(\mathbf{z}_k)$, using Equations A.24 and A.25 (and normalising). For most applications of EP, it should be possible to find an analytic expression for $f(\mathbf{z}_k)$. This term, will, more often than not, be a non-Gaussian function of $\mathbf{z}_k$. The proj[·] operator dictates that we find the mean, $\boldsymbol{\mu}_{\mathrm{proj}}$, and covariance, $\mathbf{V}_{\mathrm{proj}}$, of $\mathbf{z}_k$, for the product of factors given as an argument. As shown in Minka [2001] and Seeger [2005], general formulae exist for computing these quantities, provided that we can compute:

$$Z_{\mathrm{proj}}(\boldsymbol{\mu}_{\mathrm{cav}}, \mathbf{V}_{\mathrm{cav}}) \equiv \int \prod_{\substack{j\in\mathrm{ne}(\mathbf{z}_k)\\ j\neq i}} \tilde{\phi}_{jk}^{\mathrm{old}}(\mathbf{z}_k) f(\mathbf{z}_k) d\mathbf{z}_k. \tag{A.28}$$

with

$$\boldsymbol{\mu}_{\mathrm{proj}} = \boldsymbol{\mu}_{\mathrm{cav}} + \mathbf{V}_{\mathrm{cav}} \left[\nabla_{\boldsymbol{\mu}_{\mathrm{cav}}} \log Z_{\mathrm{proj}}\right], \tag{A.29}$$

$$\mathbf{V}_{\mathrm{proj}} = \mathbf{V}_{\mathrm{cav}} - \mathbf{V}_{\mathrm{cav}} \left[\left(\nabla_{\boldsymbol{\mu}_{\mathrm{cav}}} \nabla_{\boldsymbol{\mu}_{\mathrm{cav}}}^{\top} - 2\nabla_{\mathbf{V}_{\mathrm{cav}}}\right) \log Z_{\mathrm{proj}}\right]. \tag{A.30}$$

Finally, the new (normalised) factor $\tilde{\phi}_{ik}^{\mathrm{new}}(\mathbf{z}_k)$ is computed using the identity:

$$\tilde{\phi}_{ik}^{\mathrm{new}}(\mathbf{z}_k) = Z_{\mathrm{proj}} \frac{\mathcal{N}(\boldsymbol{\mu}_{\mathrm{proj}}, \mathbf{V}_{\mathrm{proj}})}{\prod_{\substack{j\in\mathrm{ne}(\mathbf{z}_k)\\ j\neq i}} \tilde{\phi}_{jk}^{\mathrm{old}}(\mathbf{z}_k)}. \tag{A.31}$$

A.3 Tensor Algebra

In this section, we will use the term "tensor" loosely to mean any object whose components are indexed using M indices. M will be referred to as the *order* of the tensor. We will present the basics of tensor algebra in a manner similar to the introduction in Riley et al. [2006].

- The *outer product* of a tensor $Q_{i_1,\ldots,i_M}$ of order M and that of a tensor $R_{j_1,\ldots,j_N}$ of order N is another tensor of order $M+N$, with entries given by:

$$P_{i_1,\ldots,i_M,j_1,\ldots,j_N} = Q_{i_1,\ldots,i_M} R_{j_1,\ldots,j_N}. \tag{A.32}$$

- The process of *contraction* is the generalisation of the *trace* operation to tensors. The same index is used for two subscripts and we sum over this index. An example contraction over the first two indices of a tensor $\mathbf{Q}$ is shown below:

$$C_{i_3,\ldots,i_M} = \sum_i Q_{i,i,i_3,\ldots,i_M}. \tag{A.33}$$

  Notice that a contraction over a tensor of order M produces a tensor of order $M-2$.

- A tensor product over indices i_k and j_l for two tensors $Q_{i_1,\ldots,i_k,\ldots,i_M}$ and $R_{j_1,\ldots,j_l,\ldots,j_N}$ can be performed by forming the outer product of Q and R followed by a contraction over indices i_k and j_l. This means that a tensor product of a vector of order M and one of order N is another tensor of order $M+N-2$.

$$P_{i_1,\ldots,i_{M-1},j_1,\ldots,j_{N-1}} = \sum_c Q_{i_1,\ldots,c,\ldots,i_M} R_{j_1,\ldots,c,\ldots,j_N}. \tag{A.34}$$

  As a simple example, one can view the matrix vector product

$$A_{ij} x_j = b_i, \tag{A.35}$$

  as the contraction of the third order tensor Q_{ijk} (over j and k) formed by the outer product of $\mathbf{A}$ and $\mathbf{b}$.

Appendix B

MATLAB Code Snippets

We present MATLAB code snippets for routines which are hard to present as pseudo-code, as has been the usual protocol:

- The `likfunc` routine in Algorithm 4, Line 10, which implements Equation 2.73 for targets which are binary labels ($y_i \in \{-1, +1\}$).

```
function [mu, var] = likfunc(y, mu_c, var_c)
```

  The cavity mean and variance are given by `mu_c` and `var_c` respectively. The target for which we want to compute the continuous pseudo-target `mu` and the pseudo-noise `var` is represented as `y`. Note that we do not have to compute the normalising constant, as it is handled rather naturally as part of the Kalman filtering pass.

- The `ppgpr_1D` routine in Algorithm 12, Line 6. It has the interface:

```
function [nlml, dnlml] = pp_gpr_1d(phi, stfunc, X, y)
```

  `phi` includes the projection weights $\mathbf{w}_m$ and the scalar GP hyperparameters $\boldsymbol{\theta}_m$. `stfunc` computes the state-transition matrices for the underlying Kalman filter. The purpose of this routine is to compute the marginal likelihood of the GP placed on the training set $\{\mathbf{w}_m^\top \mathbf{X}, \mathbf{y}\}$, and its derivatives w.r.t. $\mathbf{w}_m$ and

$\boldsymbol{\theta}_m$ efficiently using Kalman filtering. The variable names used in the code are consistent with the names used in Chapter 2. The derivatives labelled with the suffixes `_dl`, `_ds`, `_dn` and `_dW` correspond to derivatives w.r.t. the hyperparameters $\{\lambda, \sigma_f^2, \sigma_n^2\}$ and $\mathbf{w}_m$ respectively.

- We also present the script that uses the MATLAB Symbolic Math Toolbox to generate analytic expressions for the $\mathbf{\Phi}$ and $\mathbf{Q}$ matrices (see chapter 2) for Matérn(ν), for any ν. This script uses $p \equiv \nu - 1/2$.

```
function [mu, var] = likfunc(y, mu_c, var_c)

%Implementation of likfunc for the probit likelihood...

%Form product between cavity and non-Gaussian factor
z = (y*mu_c)/sqrt(1 + var_c);
mu_proj = mu_c + ...
(y*var_c*std_normal(z))/(Phi(z)*sqrt(1 + var_c));
aa = (var_c^2*std_normal(z))/((1 + var_c)*Phi(z));
bb = z + std_normal(z)/Phi(z);
var_proj = var_c - aa*bb;

%Divide the product by the cavity
var = 1/(1/var_proj - 1/var_c);
mu = var*(mu_proj/var_proj - mu_c/var_c);

function n = std_normal(x)
n = (1/(sqrt(2*pi)))*exp(-0.5*x*x);

function phi = Phi(x)
phi = 0.5*erfc(-x/sqrt(2));
```

```
function [nlml, dnlml] = pp_gpr_1d(phi, stfunc, X, y)

%...INITIALIZATION OF ALL NECESSARY VARIABLES BEFORE FILTERING LOOP

%FORWARD PASS : sufficient to compute marginal likelihood and derivatives...
for i = 2:T

    [Phi, Q, deriv] = feval(stfunc, lambda, signal_var, wgt, X(i,:)', X(i-1,:)');
    P = Phi*V*Phi' + Q;
    PhiMu = Phi*mu;
    pm = PhiMu(1);
    pv = P(1,1) + noise_var;

    dP_dl = Phi*V*deriv.dPhidlambda' + (Phi*dV_dl + deriv.dPhidlambda*V)*Phi' + ...
        deriv.dQdlambda;
    dP_ds = Phi*dV_ds*Phi' + deriv.dQdSigvar; dP_dn = Phi*dV_dn*Phi';
    for d = 1:D
        dP_dW(:,:,d) = Phi*V*deriv.dPhidW(:,:,d)' + ...
            (Phi*dV_dW(:,:,d) + deriv.dPhidW(:,:,d)*V)*Phi' + deriv.dQdW(:,:,d);
    end
    dPhiMu_dl = deriv.dPhidlambda*mu + Phi*dmu_dl;
    dPhiMu_ds = Phi*dmu_ds; dPhiMu_dn = Phi*dmu_dn;
    for d = 1:D
        dPhiMu_dW(:,d) = deriv.dPhidW(:,:,d)*mu + Phi*dmu_dW(:,d);
    end
    dpm_dl = dPhiMu_dl(1); dpm_ds = dPhiMu_ds(1);
    dpm_dn = dPhiMu_dn(1); dpm_dW = dPhiMu_dW(1,:)';
    dpv_dl = dP_dl(1,1); dpv_ds = dP_ds(1,1);
    dpv_dn = dP_dn(1,1) + 1; dpv_dW = squeeze(dP_dW(1,1,:));

    nlml = nlml + 0.5*(log(2*pi) + log(pv) + ((y(i) - pm)^2)/pv);
    dnlml_dpv = 0.5*(1/pv - ((y(i) - pm)^2)/(pv*pv));
    dnlml_dpm = -0.5*(2*(y(i) - pm)/pv);
    dnlml_dl = dnlml_dl + dnlml_dpv*dpv_dl + dnlml_dpm*dpm_dl;
    dnlml_ds = dnlml_ds + dnlml_dpv*dpv_ds + dnlml_dpm*dpm_ds;
    dnlml_dn = dnlml_dn + dnlml_dpv*dpv_dn + dnlml_dpm*dpm_dn;
    dnlml_dW = dnlml_dW + dnlml_dpv*dpv_dW + dnlml_dpm*dpm_dW;

    kalman_gain = (P*H')/pv;
    dg_dl = (pv*(dP_dl*H') - dpv_dl*(P*H'))/(pv*pv);
    dg_ds = (pv*(dP_ds*H') - dpv_ds*(P*H'))/(pv*pv);
    dg_dn = (pv*(dP_dn*H') - dpv_dn*(P*H'))/(pv*pv);
    for d = 1:D
        dg_dW(:,d) = (pv*(dP_dW(:,:,d)*H') - dpv_dW(d)*(P*H'))/(pv*pv);
    end
    mu = PhiMu + kalman_gain*(y(i) - pm);
    dmu_dl = dPhiMu_dl + dg_dl*(y(i) - pm) - kalman_gain*dpm_dl;
    dmu_ds = dPhiMu_ds + dg_ds*(y(i) - pm) - kalman_gain*dpm_ds;
    dmu_dn = dPhiMu_dn + dg_dn*(y(i) - pm) - kalman_gain*dpm_dn;
```

```
    for d = 1:D
        dmu_dW(:,d) = dPhiMu_dW(:,d) + dg_dW(:,d)*(y(i) - pm) - kalman_gain*dpm_dW(d);
    end
    V = (eye(p) - kalman_gain*H)*P;
    dV_dl = dP_dl - ((kalman_gain*H)*dP_dl + (dg_dl*H)*P);
    dV_ds = dP_ds - ((kalman_gain*H)*dP_ds + (dg_ds*H)*P);
    dV_dn = dP_dn - ((kalman_gain*H)*dP_dn + (dg_dn*H)*P);
    for d = 1:D
        dV_dW(:,:,d) = dP_dW(:,:,d) - ((kalman_gain*H)*dP_dW(:,:,d) + (dg_dW(:,d)*H)*P);
    end

end
dnlml = [dnlml_dW; -lambda*dnlml_dl; 2*signal_var*dnlml_ds; 2*noise_var*dnlml_dn];
```

```
function [Phi, Q, derivs] = get_Phi_Q_matern(p)

%Construct analytic expressions for SSM transition matrices ...
   using Symbolic toolbox!

lambda = sym('lambda', 'positive');
t = sym('t', 'positive');
syms s;
for i = 1:p
    M(i,i) = s;
    M(i,i+1) = -1;
end
M(p+1,p+1) = s;
poly_str = char(expand((lambda + 1)^(p+1)));
idx = findstr('+', poly_str);
idx = [0, idx];
for i = 1:length(idx)-1
    N(p+1,i) = sym(poly_str(idx(i)+1:idx(i+1)-1));
end
B = M + N;
expA = ilaplace(inv(B));
Phi = simplify(expA);
Outer = Phi(:,end)*Phi(:,end)';
delta = sym('delta', 'positive');
Q = int(Outer, t, 0, delta);
simplify(Q);
%also return derivs w.r.t. lambda & delta
if nargout > 2
    derivs.dQ_dl = diff(Q, lambda);
    derivs.dPhi_dl = diff(Phi, lambda);
    derivs.dQ_dd = diff(Q, delta);
    derivs.dPhi_dd = diff(Phi, t);
end
```

References

Abramowitz, M. and Stegun, I. A. (1964). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, Ninth edition. 29

Adams, R. and Stegle, O. (2008). Gaussian process product models for nonparametric nonstationarity. In *Proceedings of the 25th international conference on Machine learning*, pages 1–8. ACM. 20

Adams, R. P. (2009). *Kernel Methods for Nonparametric Bayesian Inference of Probability Densities and Point Processes*. PhD thesis, Department of Physics, University of Cambridge, Cambridge, UK. 146

Adams, R. P. and MacKay, D. J. (2007). Bayesian online changepoint detection. Technical report, University of Cambridge, Cambridge, UK. 55, 56, 57

Alvarez, M., Luengo, D., and Lawrence, N. (2009). Latent force models. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5, pages 9–16. 147

Archambeau, C. and Opper, M. (2011). Approximate inference for continuous-time Markov processes. *Inference and Estimation in Probabilistic Time-Series Models. Cambridge University Press*. 147

Archambeau, C., Opper, M., Shen, Y., Cornford, D., and Shawe-Taylor, J. (2008). Variational inference for diffusion processes. *Advances in Neural Information Processing Systems*, 20:17–24. 147

Arnold, L. (1992). *Stochastic Differential Equations: Theory and Practice*. Krieger Publishing Corporation. 37

Bar-Shalom, Y., Li, X. R., and Kirubarajan, T. (2001). *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons. 22, 40

Barry, D. and Hartigan, J. A. (1992). Product partition models for change point problems. *The Annals of Statistics*, 20(1):260–279. 20, 56

Barry, D. and Hartigan, J. A. (1993). A Bayesian analysis of change point problems. *Journal of the Americal Statistical Association*, 88(421):309–319. 56

Bentley, J. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517. 144

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer. i, 47, 89, 147

Bock, K. et al. (2004). Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope. *Nuclear Instruments and Methods in Physics Research A*, 516:511–528. 117

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, Cambridge, UK. 15

Boyle, P. and Frean, M. (2005). Dependent Gaussian processes. In *Advances in neural information processing systems 17: proceedings of the 2004 conference*, volume 17, page 217. The MIT Press. 147

Breiman, L. and Friedman, J. H. (1985). Estimating optimal transformations for multiple regression. In *Computer Science and Statistics: Proceedings of the 16th Symposium on the Interface*, pages 121–134. 75

Buja, A., Hastie, T., and Tibshirani, R. (1989). Linear smoothers and additive models. *The Annals of Statistics*, 17:453–510. 73

Chang, C. and Lin, C. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. 114

Chang, C. and Lin, C.-J. (2001). Ijcnn 2001 challenge: Generalization ability and text decoding. In *In Proceedings of IJCNN. IEEE*, pages 1031–1036. 117

Chib, S. (1998). Estimation and comparison of multiple change-point models. *Journal of Econometrics*, 86(2):221–241. 56, 147

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297. 113

Cox, D. (1955). Some statistical methods connected with series of events. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 129–164. 146

Cunningham, J. P., Shenoy, K. V., and Sahani, M. (2008). Fast Gaussian process methods for point process intensity estimation. In McCallum, A. and Roweis, S., editors, *25th International Conference on Machine Learning*, pages 192–199, Helsinki, Finland. ACM. 21, 55, 65, 126, 127

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38. 42

Doob, J. L. (1953). *Stochastic Processes*. John Wiley & Sons, Cambridge, MA, USA. 7

Douc, R., Garivier, A., Moulines, E., and Olsson, J. (2009). On the Forward Filtering Backward Smoothing Particle Approximations of the Smoothing Distribution in General State Space Models. 82

Doucet, A., De Freitas, N., and Gordon, N. (2001). *Sequential Monte Carlo methods in practice*. Springer Verlag. 44

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222. 63

Fearnhead, P. (2006). Exact and efficient Bayesian inference for multiple changepoint problems. *Statistics and computing*, 16(2):203–213. 56

Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression. *Journal of the American Statistical Association*, 76:817–823. 72, 73

Friedman, J. H., Stuetzle, W., and Schroeder, A. (1984). Projection pursuit density estimation. *Journal of the American Statistical Association*, 79:599–608. 73

Garnett, R., Osborne, M. A., and Roberts, S. J. (2009). Sequential Bayesian prediction in the presence of changepoints. In *26th International Conference on Machine Learning*, pages 345–352, Montreal, QC, Canada. ACM. 61, 68

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. The Johns Hopkins University Press, Third edition. 79, 126

Gramacy, R. C. (2005). *Bayesian treed Gaussian process models*. PhD thesis, UC Santa Cruz, Santa Cruz, California, USA. 20, 128, 147

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732. 95

Green, P. J. and Silverman, B. W. (1994). *Nonparametric Regression and Generalised Linear Models*. Chapman & Hall/CRC. 33

Grewal, M. S. and Andrews, A. P. (2001). *Kalman Filtering: Theory and Practice using MATLAB*. John Wiley & Sons, Second edition. 22

Hartikainen, J. and Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal Gaussian process regression models. In *Machine Learning for Signal Processing (MLSP)*, pages 379–384, Kittilä, Finland. IEEE. 20, 22, 31, 53

Hastie, T. and Tibshirani, R. (1998). Bayesian backfitting. *Statistical Science*, 15:193–223. 82

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer-Verlag, Second edition. i, 31, 72, 73

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized additive models*. London: Chapman & Hall. 72, 73, 76, 80

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. 4

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME — Journal of Basic Engineering*, 82(Series D):35–45. 22, 24, 41

Karklin, Y. and Lewicki, M. (2005). A hierarchical Bayesian model for learning nonlinear statistical regularities in nonstationary natural signals. *Neural Computation*, 17(2):397–423. 20

Kimeldorf, G. S. and Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95. 33

Knutson, A. and Tao, T. (2000). Honeycombs and sums of Hermitian matrices. 143

Kreyszig, E. (1989). *Introductory Functional Analysis with Applications*. John Wiley & Sons, First edition. 32

Kuss, M. and Rasmussen, C. (2005). Assessing approximate inference for binary Gaussian process classification. *The Journal of Machine Learning Research*, 6:1679–1704. 46

Lawrence, N. D., Seeger, M., and Herbrich, R. (2003). Fast sparse Gaussian process methods: the informative vector machine. In *Advances in Neural Information Processing Systems 17*, pages 625–632, Cambridge, MA, USA. The MIT Press. 18, 115

Lázaro-Gredilla, M. (2010). *Sparse Gaussian Processes for Large-Scale Machine Learning*. PhD thesis, Universidad Carlos III de Madrid, Madrid, Spain. 17, 108

MacKay, D. J. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. i, 81

Minka, T. (2001). *A family of algorithms for approximate Bayesian inference*. Phd thesis, MIT. 14, 45, 47, 115, 154

Minka, T. (2005). Divergence measures and message passing. Technical report, Microsoft Research. 92

Murray, I. and Adams, R. P. (2010). Slice sampling covariance hyperparameters of latent Gaussian models. In Lafferty, J., Williams, C. K. I., Zemel, R., Shawe-Taylor, J., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1723–1731. 4

Naish-Guzman, A. and Holden, S. (2007). The generalized FITC approximation. In *Advances in Neural Information Processing Systems 21*, pages 534–542, Cambridge, MA, USA. The MIT Press. 18

Navone, H., Granitto, P., and Verdes, P. (2001). A Learning Algorithm for Neural Network Ensembles. *Journal of Artificial Intelligence*, 5(12):70–74. 108

Neal, R. (1993). Probabilistic inference using markov chain monte carlo methods. Technical report, University of Toronto. 81

Nickisch, H. and Rasmussen, C. E. (2008). Approximations for binary gaussian process classification. *Journal of Machine Learning Research*, 9:2035–2078. 14, 45, 148

Oh, S., Rehg, J., Balch, T., and Dellaert, F. (2008). Learning and inferring motion patterns using parametric segmental switching linear dynamic systems. *International Journal of Computer Vision*, 77(1):103–124. 69

O'Hagan, A. and Forster, J. (1994). *Kendall's Advanced Theory of Statistics: Volume 2B; Bayesian Inference*. Halsted Press. 4

O'Hagan, A. and Kingman, J. F. C. (1978). Curve Fitting and Optimal Design for Prediction. *Journal of the Royal Statistical Society. Series B (Methodological)*, 40(1):1–42. 128

Oppenheim, A., Willsky, A., and Young, I. (1996). *Signals and Systems*. Prentice-Hall Upper Saddle River, NJ, Second edition. 25

Ó Ruanaidh, J. J., Fitzgerald, W. J., and Pope, K. J. (1994). Recursive Bayesian location of a discontinuity in time series. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 4, pages 513–516, Adelaide, SA, Australia. IEEE. 56

Osborne, M. (2010). *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. PhD thesis, University of Oxford, Oxford, UK. 128

Papoulis, A., Pillai, S., and Unnikrishna, S. (2002). *Probability, random variables, and stochastic processes*. McGraw-Hill New York, Fourth edition. 11

Platt, J. C. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. 114

Platt, J. C. (1999). Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press. 114

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Programming*. Cambridge University Press, Third edition. 143

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959. 18

Rasmussen, C. and Ghahramani, Z. (2003). Bayesian Monte Carlo. *Advances in Neural Information Processing Systems*, pages 505–512. 61

Rasmussen, C. E. and Nickisch, H. (2010). Gaussian processes for machine learning (gpml) toolbox. *Journal of Machine Learning Research*, 11:3011–3015. 22, 44

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. The MIT Press. 12, 14, 16, 18, 29, 31, 113, 129, 130

Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamical systems. *AIAA Journal*, 3(8):1445–1450. 24

Rayner, N., Brohan, P., Parker, D., Folland, C., Kennedy, J., Vanicek, M., Ansell, T., and Tett, S. (2006). Improved analyses of changes and uncertainties in sea surface temperature measured in situ since the mid-nineteenth century: the HadSST2 dataset. *Journal of Climate*, 19(3):446–469. 127

Reinsch, C. H. (1967). Smoothing by spline functions. *Numerische Mathematik*, 10(3):177–183. 33

Riley, K. F., Hobson, M. P., and Bence, S. J. (2006). *Mathematical Methods for Physics and Engineering*. Cambridge University Press, Third edition. 37, 134, 155

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. 148

Saatçi, Y., Turner, R., and Rasmussen, C. E. (2010). Gaussian process change point models. In *27th International Conference on Machine Learning*, pages 927–934, Haifa, Israel. Omnipress. 55, 70

Sariyar, M., Borg, A., and Pommerening, K. (2011). Controlling false match rates in record linkage using extreme value theory. *Journal of Biomedical Informatics*. 142

Särkkä, S. (2006). *Recursive Bayesian Inference on Stochastic Differential Equations*. PhD thesis, Aalto University. 38

Särkkä, S. (2011). Linear Operators and Stochastic Partial Differential Equations in Gaussian Process Regression. *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 151–158. 148

Schölkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, Cambridge, MA, USA. 114

Seeger, M. (1999). Relationships between Gaussian Processes, Support Vector Machines and Smoothing Splines. Technical report, University of Edinburgh, Edinburgh, UK. 33

Seeger, M. (2005). Expectation propagation for exponential families. Technical report, University of California at Berkeley. 154

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264, Cambridge, MA, USA. The MIT Press. 17, 18, 104, 148

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In Fürnkranz, J. and Joachims, T., editors, *27th International Conference on Machine Learning*, pages 1015–1022, Haifa, Israel. Omnipress. 128

Storkey, A. J. (1999). Truncated covariance matrices and Toeplitz methods in Gaussian processes. In *9th International Conference on Artificial Neural Networks*, volume 1, pages 55–60, Edinburgh, UK. IET. 126, 129

Teh, Y., Seeger, M., and Jordan, M. (2005). Semiparametric latent factor models. In *Workshop on Artificial Intelligence and Statistics*, volume 10. 147

Van Gael, J., Saatçi, Y., Teh, Y., and Ghahramani, Z. (2008). Beam sampling for the infinite hidden Markov model. In *Proceedings of the 25th international conference on Machine learning*, pages 1088–1095. ACM. 64

Wahba, G. (1990). *Spline Models for Observational Data*. Capital City Press. 31, 35

Wecker, W. E. and Ansley, C. F. (1983). The Signal Extraction Approach to Nonlinear Regression and Spline Smoothing. *Journal of the American Statistical Association*, 78(381):81– 89. 22, 39

Willems, J. (1971). Least squares stationary optimal control and the algebraic Riccati equation. *Automatic Control, IEEE Transactions on*, 16(6):621–634. 36

Williams, C. and Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*. 17

Zhang, Y., Leithead, W. E., and Leith, D. J. (2005). Time-series Gaussian Process Regression Based on Toeplitz Computation. *Optimization*, pages 3711–3716. 21, 126