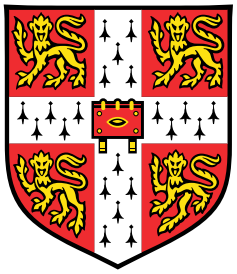
Advances in Software and Spatio-Temporal Modelling with Gaussian Processes



William Christopher Tebbutt

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Darwin College

September 2022

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

William Christopher Tebbutt
September 2022

Acknowledgements

I would first like to thank my supervisors – Richard E. Turner, Emily Shuckburgh, and Scott Hosking – for their guidance, patience, and advice. In particular Rich for many years of mentorship.

I would like to thank Arno Solin for hosting me for two highly fullfilling and educational months at Aalto in 2019, Mike Davey for his advice and guidance, and Carl Rasmussen and Simo Särkkä for an enjoyable and instructive viva.

I have been fortunate in my friends and colleagues in Cambridge: Wessel Bruinsma – our technical discussions have been formative – all past and current members of The Galapagogos, Helen, my office mates in BE4-40, and Will and Millie – I could not have asked for better people to be stuck in a house with for a year. I am grateful to Joel, Hannah, and Wessel for providing light relief during the last days of writing this thesis. Finally I thank my family – Mum, Dad, and Tom – for everything.

Abstract

This thesis concerns the use of Gaussian processes (GPs) as distributions over unknown functions in Machine Learning and probabilistic modeling. GPs have been found to have utility in a wide range of applications owing to their flexibility, interpretability, and tractability. I advance their use in three directions.

Firstly, the abstractions upon which software is built for their use in practice. In modern GP software libraries such as GPML, GPy, GPflow, and GPyTorch, the kernel is undoubtedly the dominant abstraction. While it remains highly successful it of course has limitations, and I propose to address some of these through a complementary abstraction: affine transformations of GPs. Specifically I show how a collection of GPs, and affine transformations thereof, can themselves be treated as a single GP. This in turn leads to a design for software, including exact and approximate inference algorithms. I demonstrate the utility of this software through a collection of worked examples, focussing on models which are more cleanly and easily expressed using this new software.

Secondly, I develop a new scalable approximate inference algorithm for a class of GPs commonly utilised in spatio-temporal problems. This is a setting in which GPs excel, for example enabling the incorporation of important inductive biases, and observations made at arbitrary points in time and space. However, the computation required to perform exact inference and learning in GPs scales cubically in the number of observations, necessitating approximation, to which end I combine two important complementary classes of approximation: pseudo-point and Markovian. The key contribution is the insight that a simple and useful way to combine them turns out to be well-justified. This resolves an open question in the literature, provides new insight into existing work, and a new family of approximations. The efficacy of an important member of this family is demonstrated empirically.

Finally I develop a GP model and associated approximate inference techniques for the prediction of sea surface temperatures (SSTs) on decadal time scales, which are relevant when taking planning decisions which consider resilience to climate change. There remains a

large degree of uncertainty as to the state of the climate on such time scales, but it is thought to be possible to reduce this by exploiting the predictability of natural variability in the climate. The developed GP-based model incorporates a key assumption used by the existing statistical models employed for decadal prediction, thus retaining a valuable inductive bias, while offering several advantages. Amongst these is the lack of need for spatial aggregation of data, which is especially relevant when data are sparse, as is the case with historical ocean SST data.

In summary, this thesis contributes to the practical use of GPs through a set of abstractions that are useful in the design of software, algorithms for approximate inference in spatial-temporal settings, and their use in decadal climate prediction.

Table of contents

List of figures

List of tables

Chapter 1

Introduction and Background

Gaussian processes are flexible nonparametric distributions over real-valued functions, and are utilised throughout machine learning, probabilistic modelling, and this thesis.

The simplest use of GPs is for non-linear regression with measurements made under Gaussian noise. Simple extensions of this model replace the Gaussian observation noise with something non-Gaussian. For example a Bernoulli distribution whose parameter is a function of the GP is often used in classification problems, or a student's-t distribution whose mean given by the GP in a non-linear regression problem when the noise is thought to be heavy-tailed. GPs have seen use in probabilistic treatments of numerics problems, such as approximate integration of a function (Ghahramani and Rasmussen, 2003; O'Hagan, 1987, 1991; Xi et al., 2018), line searches in optimisation (Mahsereci and Hennig, 2017), and differential equation solvers (Archambeau et al., 2007; Duffin et al., 2021; Schober et al., 2019). They are used extensively in the probabilistic treatments of global optimisation problems, known as Bayesian optimisation, as models on which search directions are based (Hernández-Lobato et al., 2017; Osborne et al., 2009; Snoek et al., 2012; Wu et al., 2017a,b). They have been used as models for natural sounds (Turner, 2010), in audio signal analysis (Turner and Sahani, 2011; Wilkinson et al., 2019a,b), to parametrise transition dynamics discrete-time (Frigola et al., 2014; Ialongo et al., 2019, 2018) and continuous-time (Duncker et al., 2019; Wilson et al., 2021) dynamical systems, in particular in model-based reinforcement learning (Deisenroth and Rasmussen, 2011). They have been applied to the problem of simultaneous localisation and mapping (SLAM) (Kok and Solin, 2018), automating work usually done by a human statistician (Duvenaud et al., 2013; Steinruecken et al., 2019), solar power forecasting (Dahl and Bonilla, 2019), as a model for a spatially- and temporally-varying effective reproduction number in a large model for Covid-19 transmission model (Nicholson

et al., 2021), as a model for dimensionality reduction and density estimation (Adams et al., 2008; Damianou et al., 2016; Lawrence and Hyvärinen, 2005; Lawrence, 2003; Titsias and Lawrence, 2010), as the component model in deep probabilistic models (Damianou and Lawrence, 2013; Duvenaud et al., 2014; Salimbeni et al., 2019), for multi-fidelity modelling (Perdikaris et al., 2017), and in multi-output (multi-task) regression problems (Alvarez et al., 2011; Bruinsma et al., 2020; Goovaerts, 1997; Requeima et al., 2019)

The above selection of applications is not intended to form an exhaustive list of uses of GPs, simply a collection of uses I have encountered and find interesting, and is intended to highlight the versatility of GPs through some of the many varied uses throughout the literature. The common thread tying them together is the need to infer some unknown real-valued function for which a concise parametric form is unavailable.

This short chapter provides a technical introduction to Gaussian processes and a family of approximate inference methods known as *pseudo-point approximations*, as both are utilised in each subsequent chapter. It concludes with an outline of the contributions of the remaining chapters of the thesis.

1.1 Standard GP Regression

The simplest example of this is as a prior distribution over an unobserved function in non-linear regression under noisy observations. Consider such a problem in which N real-valued observations $\mathbf{y} := (y_1, y_2, ..., y_N)$ are made of some unknown function at input locations $\mathbf{x} := (x_1, x_2, ..., x_N)$, under independent and identically-distributed (i.i.d.) Gaussian noise with zero-mean and variance σ^2. If a GP prior, f, is placed over the unknown function, then it is possible to perform exact inference to obtain the posterior, $f \mid \mathbf{y}, \mathbf{x}$. This is depicted in Fig. 1.1: the left panel depicts a particular GP prior distribution though a handful of samples (thin wiggly lines) and the marginal statistics of the prior, indicated by the mean (thick constant line at zero) ± 3 standard deviations from it (the filled region). The right panel depicts the posterior distribution over f, another GP, in the same manner. Notice how all of the samples from the posterior run close to the data, and the uncertainty is much lower nearer the data, reflecting the often desirable assumption that our model should be less uncertain about the value of the function near where data have been observed.

The above example is but the simplest use-case for GPs in machine learning and probabilistic modelling, but GPs are used in this manner in a surprisingly large number of situations.

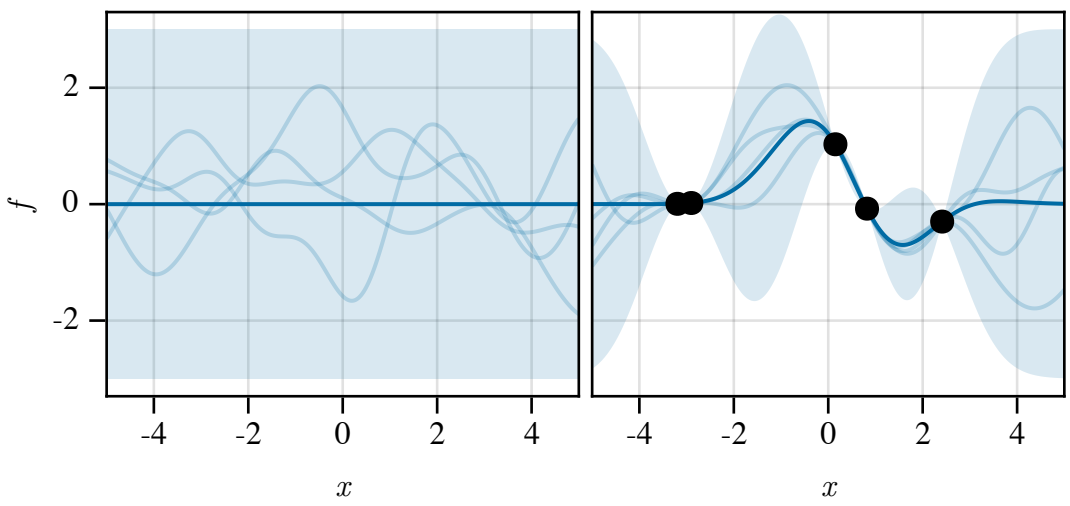


Fig. 1.1 Left: distribution over function before data is observed. Right: distribution over function after data is observed. Thin lines are sample paths, thick line is the mean function, the filled interval is the mean function ± 3 standard deviations, and black dots are observations.

1.2 Definition and Exact Inference

This section provides a more precise definition of a GP, introduces notation used throughout this thesis, and explains how exact inference is performed in simple settings.

A GP is defined as a collection of random variables, the marginal distribution over any subset of which is Gaussian (Rasmussen and Williams, 2006). This definition maps on to the above example by associating with each $x \in \mathcal{X} := \mathbb{R}$ a random variable $f(x)$, and specifying that the joint distribution over any subset $\mathbf{f} := [f(x_1), ..., f(x_N)]$ of the random variables is a multivariate Gaussian. I refer to $\mathcal{X}$ as the *domain* of f, as it is the domain of the functions sampled from it and need not be $\mathbb{R}$.[1] In the machine learning literature, a GP is usually explicitly parametrised by a *mean function*, $m : \mathcal{X} \to \mathbb{R}$, and *covariance function* or *kernel*, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The mean function specifies the mean vector of $\mathbf{f}$: $\mathbf{m_f} := [m(x_1), ..., m(x_N)]$, and the kernel the covariance matrix: $[\mathbf{C_f}]_{ij} := \kappa(x_i, x_j)$. By

$$f \sim \mathcal{GP}(m, \kappa)\,, \tag{1.1}$$

I mean that f is distributed according to a GP, which is to say that f is a stochastic process satisfying the above conditions.

[1] $\mathcal{X}$ is often called the *index set*.

Exact posterior inference is tractable under a Gaussian observation model, reducing to some standard linear algebra operations. Specifically, consider the simple hierarchical model for non-linear regression discussed above:

$$f \sim \mathcal{GP}(m, \kappa)\,, \quad y_n \mid f \sim \mathcal{N}\big(f(x_n), \sigma^2\big)\,, \quad n \in \{1, ..., N\}. \tag{1.2}$$

The posterior (conditional) distribution over f given $\mathbf{y}$, denoted $f' := f \mid \mathbf{y}$, is itself another Gaussian process with mean function

$$m'(x) := m(x) + \mathbf{c_f}(x)^\top \left(\mathbf{C_f} + \sigma^2\mathbf{I}\right)^{-1} (\mathbf{y} - \mathbf{m_f}), \tag{1.3}$$

where

$$\mathbf{c_f}(x) := \begin{bmatrix} \kappa(x, x_1) \\ \vdots \\ \kappa(x, x_N) \end{bmatrix}, \tag{1.4}$$

and kernel

$$\kappa'(x, x') := \kappa(x, x') - \mathbf{c_f}(x)^\top \left(\mathbf{C_f} + \sigma^2\mathbf{I}\right)^{-1} \mathbf{c_f}(x')\,. \tag{1.5}$$

Thus the posterior marginal distribution over the GP at any finite collection of inputs $\mathbf{x}_*$ is again a multivariate Gaussian distribution, whose mean vector and covariance matrix are given by applying Eq. (1.3) and Eq. (1.5) to $\mathbf{x}_*$.

Thus far it has been assumed that the mean function and kernel are known. In practice the mean function is typically set to $\mathbf{0}$ everywhere, and most of the hard work goes into selecting the kernel. There are a number of different families of kernel from which to choose, and they each tend to have a number of parameters θ that must be chosen. The most popular way to pick these parameters is by maximising the log marginal likelihood of these parameters given the observed data – this is known as Type-II maximum likelihood. That is, choosing θ such that $\log p(\mathbf{y} \mid \theta)$ is maximised. Noting that $p(\mathbf{y} \mid f) = p(\mathbf{y} \mid \mathbf{f})$ by definition (Eq. (1.2)), the log marginal likelihood is simply

$$\log p(\mathbf{y} \mid \theta) := \log \int p(\mathbf{y}, \mathbf{f} \mid \theta)\, \mathrm{d}\mathbf{f} = \log \mathcal{N}\big(\mathbf{y}; \mathbf{0}, \mathbf{C_f} + \sigma^2\mathbf{I}\big) \tag{1.6}$$

where $\mathbf{C_f}$ and σ^2 are functions of θ, but this dependence is suppressed in order to keep the notation clean. This optimisation problem does *not* have a closed-form solution in general, and is usually non-convex. Consequently, heuristic optimisation procedures are typically applied, such as gradient-based optimisation procedures in conjunction with some kind of

search over initialisations, in order to explore multiple modes of the log marginal likelihood. Approximate Bayesian inference is sometimes performed over the hyperparameters, for example using Hamiltonian Monte Carlo (Williams and Rasmussen, 1996). Recent work by Lalchand and Rasmussen (2020) and Simpson et al. (2020) highlights the substantial advantages that approximate inference over hyperparameters can provide relative to Type-II maximum likelihood when the hyperparameters are poorly constrained by the prior and observations.

Benefits and Limitations The above describes how exact inference in a Gaussian process is performed, however, it is worth noting two key limitations. Firstly, it is necessary that everything be jointly Gaussian – if the observation model for y_n is not a Gaussian centred on f, it would not have been possible to perform inference exactly. Tangentially, note that the expressions for the posterior mean function, posterior kernel, and log marginal likelihood, all involve the inverse of the matrix $\mathbf{C_f} + \sigma^2\mathbf{I}$, and the log marginal likelihood also requires the log determinant of this matrix. Since this matrix is positive definite, this is typically implemented in terms of the Cholesky factorisation. Computing this factorisation requires $\mathcal{O}(N^3)$ operations, meaning that exact inference is limited to approximately $N = 50,000$ observations in practice on modern hardware, and really not more than roughly $N = 10,000$ on the machines accessible to most people.

It is important to remember, however, that Gaussian processes are one of only very few distribution over functions in which exact Bayesian inference is tractable under any circumstances.[2] In this sense, they are truly remarkable.

1.3 Pseudo-Point Approximations

Pseudo-point (also known as *sparse*, or *inducing point*) approximations are one possible approach to approximating exact inference and learning in a GP when a large number of data are available.

Roughly speaking, they do this by summarising the exact posterior induced through a complete data set of N observations through a much smaller set of M carefully chosen (uncertain) pseudo-observations, and perform well when a specific kind of *over-sampling* structure is available in the problem. To understand this, consider Fig. 1.2. It depicts a situation in which many noisy observations of the underlying function are available, but it is

[2] Student-t processes (Shah et al., 2014) are also tractable, but they are essentially just Gaussian processes with a conjugate prior placed on the overall processes variance.

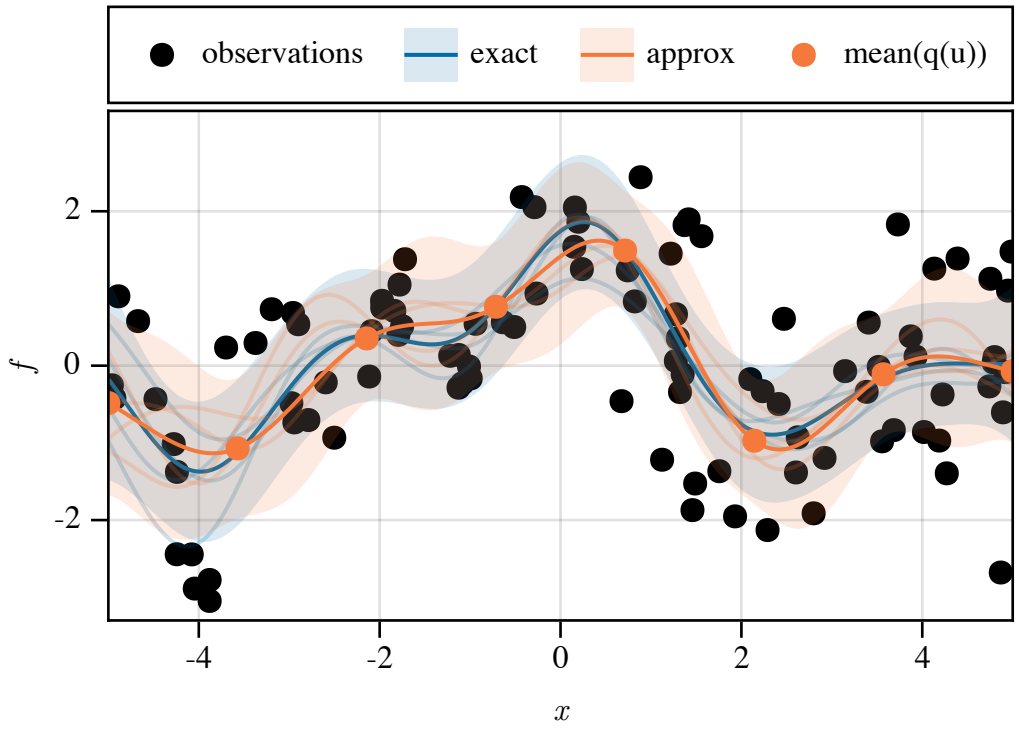


Fig. 1.2 Simple pseudo-point approximation to exact inference. Thin lines are sample paths, thick lines mean functions, filled regions ± 3 standard deviations from the mean, black dots are observations, and orange dots are the means of pseudo-points. The pseudo-point approximate works reasonably well in this problem because a small number of pseudo-points are able to summarise a large number of noisy observations.

clear that the same posterior could be induced using a small set of observations made under only a small amount of noise.

You might hope that by carefully constructing a small set of pseudo-observations it would be possible to avoid scaling cubically in N. This is indeed the case, and it is in situations like that described above in which pseudo-point approximations shine.

In this section I review in depth the most popular formulation of pseudo-point approximation, detailing in particular

- how it is formalised through variational inference,
- how is can be used to efficiently produce *marginal* predictions,
- how it can be used to construct an objective function which scales (sub-)linearly in N,
- a variety of different ways to implement the approximation in practice, which can effect how well the approximation works,
- other ways to formalise pseudo-point approximations,
- and a summary of the benefits that it affords, and where its limitations lie.

1.3.1 Pseudo-Point Approximation as Variational Inference

Consider a GP, $f \sim \mathcal{GP}(m, \kappa)$, of which N observations $\mathbf{y} \in \mathbb{R}^N$ are made at locations $\mathbf{x} \in \mathcal{X}^N$ through observation model

$$p(\mathbf{y} \mid \mathbf{f}) := \prod_{n=1}^{N} p(y_n \mid \mathbf{f}_n), \quad \mathbf{f}_n := f(\mathbf{x}_n). \tag{1.7}$$

Suppose (briefly) that $\mathcal{X}$ is finite. The seminal work of Titsias (2009), revisited by Matthews et al. (2016), can be viewed as proposing to approximate the exact posterior distribution over f given y with another GP of the form

$$q(f) = q(\mathbf{u})\, p(f_{\neq \mathbf{u}} \mid \mathbf{u}), \tag{1.8}$$

where $u_m := f(z_m)$ are the *pseudo-points* for a collection of M *pseudo-inputs* $\mathbf{z}$, and $f_{\neq \mathbf{u}} := f \setminus \mathbf{u}$ are all of the random variables in f except those used as pseudo-points. $p(f_{\neq \mathbf{u}} \mid \mathbf{u})$ is the exact conditional distribution of the GP given $\mathbf{u}$, and $q(\mathbf{u})$ is assumed to be an arbitrary multivariate Gaussian with mean $\mathbf{m}_{\mathbf{u}}^{\mathrm{q}}$ and covariance matrix $\mathbf{C}_{\mathbf{u}}^{\mathrm{q}}$.

More notation is required to explain how particular values for $\mathbf{m}_\mathbf{u}^\mathrm{q}$ an $\mathbf{C}_\mathbf{u}^\mathrm{q}$ are chosen. First parition the random variables in f into three disjoint subsets: $\mathbf{u}$, $\mathbf{f}$, $f_{\neq \mathbf{u},\mathbf{f}} := f \setminus \mathbf{u} \cup \mathbf{f}$ – in what follows f and $\{\mathbf{u}, \mathbf{f}, f_{\neq \mathbf{u},\mathbf{f}}\}$ are interchanged freely since they correspond to the same collection of random variables. This further implies that we may re-state Eq. (1.8) as

$$q(f) = q(\mathbf{u})\, p(\mathbf{f} \,|\, \mathbf{u})\, p(f_{\neq \mathbf{u},\mathbf{f}} \,|\, \mathbf{u}, \mathbf{f})\,, \tag{1.9}$$

and

$$p(f) = p(\mathbf{u})\, p(\mathbf{f} \,|\, \mathbf{u})\, p(f_{\neq \mathbf{u},\mathbf{f}} \,|\, \mathbf{u}, \mathbf{f})\,. \tag{1.10}$$

Noting that $\mathbf{y} \perp\!\!\!\perp \mathbf{u} \cup f_{\neq \mathbf{u},\mathbf{f}} \mid \mathbf{f}$ so $p(\mathbf{y} \,|\, f) = p(\mathbf{y} \,|\, \mathbf{f})$, particular values for $\mathbf{m}_\mathbf{u}^\mathrm{q}$ and $\mathbf{C}_\mathbf{u}^\mathrm{q}$ are now chosen by minimising the *Kullback-Leibler* (KL) divergence between q and the exact posterior:

$$\begin{aligned}\mathcal{KL}[q(f) \,\|\, p(f \,|\, \mathbf{y})] &= \log p(\mathbf{y}) + \mathbb{E}_q\left[\frac{q(f)}{p(f)\, p(\mathbf{y} \,|\, \mathbf{f})}\right] \\ &= \log p(\mathbf{y}) + \mathbb{E}_q\left[\log \frac{q(\mathbf{u})}{p(\mathbf{u})\, p(\mathbf{y} \,|\, \mathbf{f})} \frac{p(\mathbf{f} \,|\, \mathbf{u})\, p(f_{\neq \mathbf{u},\mathbf{f}} \,|\, \mathbf{u}, \mathbf{f})}{p(\mathbf{f} \,|\, \mathbf{u})\, p(f_{\neq \mathbf{u},\mathbf{f}} \,|\, \mathbf{u}, \mathbf{f})}\right]. \end{aligned} \tag{1.11}$$

Noting that the final ratio cancels, we see why the particular choice of $q(f)$ is crucial – the dimension of f, and therefore $f_{\neq \mathbf{u},\mathbf{f}}$, is typically uncountably infinite, so it is vital that they do not appear in any computations that are needed in practice. Given this, and further noting that $\mathbf{y} \perp\!\!\!\perp \mathbf{u} \mid \mathbf{f}$, the above simplifies to

$$\mathcal{KL}[q(f) \,\|\, p(f \,|\, \mathbf{y})] = \log p(\mathbf{y}) + \mathcal{KL}[q(\mathbf{u}) \,\|\, p(\mathbf{u})] - \mathbb{E}_q[\log p(\mathbf{y} \,|\, \mathbf{f})]\,. \tag{1.12}$$

This is an instance of Variational Inference (a.k.a. Variational Bayes – see e.g. (Murphy, 2012)).

The pseudo-inputs $\mathbf{z}$ are also variational parameters, and can be chosen to minimise Eq. (1.11) without risking over-fitting. In practice the resulting optimisation problem is quite hard, so the pseudo-input locations are often fixed if working in low-dimensional domains, or a simple heuristic employed in higher-dimensions (Burt et al., 2020a). In what follows the dependence of $q(\mathbf{u})$ on $\mathbf{z}$ will be suppressed.

Making Predictions Recall that the conditional distribution $\mathbf{f} \mid \mathbf{u}$ has density

$$p(\mathbf{f} \,|\, \mathbf{u}) = \mathcal{N}(\mathbf{f}; \mathbf{m}_\mathbf{f} + \mathbf{C}_\mathbf{fu}\Lambda_\mathbf{u}(\mathbf{u} - \mathbf{m}_\mathbf{u}), \mathbf{C}_\mathbf{f} - \mathbf{C}_\mathbf{fu}\Lambda_\mathbf{u}\mathbf{C}_\mathbf{uf})\,, \tag{1.13}$$

where $\Lambda_{\mathbf{u}} := \mathbf{C}_{\mathbf{u}}^{-1}$, $[\mathbf{C}_{\mathbf{u}}]_{ij} := \kappa(z_i, z_j)$, $[\mathbf{C}_{\mathbf{fu}}]_{ij} := \kappa(x_i, z_j)$, $\mathbf{C}_{\mathbf{uf}} := \mathbf{C}_{\mathbf{fu}}^{\top}$, and $[\mathbf{m}_{\mathbf{u}}]_i := m(z_i)$. Noting that $\mathbf{C}_{\mathbf{f}}$ appears in the expression for its covariance matrix, it is clear that this distribution is intractable in the same sense that the prior is – operations such as sampling, density calculation, and covariance calculation require $\mathcal{O}(N^3)$ operations. A key insight, however, is that many applications require only the marginals, not the entire joint distribution. This means that it is often sufficient to obtain only the diagonal of the covariance, requiring only $\mathcal{O}(NM^2 + M^3)$ operations which, crucially, is linear in N.

In practice, one typically also marginalises over $\mathbf{u}$ to obtain the marginals of $q(\mathbf{f})$, which has density

$$\mathcal{N}(\mathbf{f}; \mathbf{m}_{\mathbf{f}} + \mathbf{C}_{\mathbf{fu}}\Lambda_{\mathbf{u}}(\mathbf{m}_{\mathbf{u}}^{\mathrm{q}} - \mathbf{m}_{\mathbf{u}}), \mathbf{C}_{\mathbf{f}} - \mathbf{C}_{\mathbf{fu}}\Lambda_{\mathbf{u}}(\mathbf{C}_{\mathbf{u}} - \mathbf{C}_{\mathbf{u}}^{\mathrm{q}})\Lambda_{\mathbf{u}}\mathbf{C}_{\mathbf{uf}}) \,. \tag{1.14}$$

Operations on this have essentially the same computational properties as those on $\mathbf{f} \mid \mathbf{u}$.

Infinite Domains The preceeding manipulations assumed $\mathcal{X}$ to be finite in order to enable a simplified derivation of the key results in Eq. (1.12) and Eq. (1.14). Matthews et al. (2016) provide a measure-theoretic treatment of the above which handles the technical problems that arise when $\mathcal{X}$ is infinite, and produce precisely the same results. In the remainder of this work, $\mathcal{X}$ may therefore be assumed to be infinite.

1.3.2 The Unsaturated Bound

Through the application of Gibbs' inequality (see e.g. MacKay (2003)), which states that the KL divergence is non-negative, a lower bound on the $\log p(\mathbf{y})$ can be derived from Eq. (1.12):

$$\log p(\mathbf{y} \mid \theta) \geq \mathbb{E}_q[\log p(\mathbf{y} \mid \mathbf{f})] - \mathcal{KL}[q(\mathbf{u}) \,\|\, p(\mathbf{u})] =: \mathcal{L}_{\mathrm{u}}. \tag{1.15}$$

This kind of lower bound is typically referred to as the ELBO – Evidence Lower BOund – as $\log p(\mathbf{y})$ is also known as the *evidence*. It is common practice to optimise the ELBO with respect to (w.r.t.) both θ and the variational parameters which specify $q(\mathbf{u})$. Henceforth, this particular ELBO will be referred as the *unsaturated bound*. It was first considered by Hensman et al. (2013), and has the advantage that a simple unbiased estimator for it, and its gradient w.r.t. the parameters of q, can be constructed that requires evaluation of only a mini-batch of the elements in $\mathbf{y}$. To see how this is achieved, first apply the conditional independence assumption made previously in Eq. (1.7) to conclude that:

$$\mathbb{E}_q[\log p(\mathbf{y} \mid \mathbf{f})] = \sum_{n=1}^{N} \mathbb{E}_{q(\mathbf{f}_n)}[\log p(y_n \mid \mathbf{f}_n)] \,. \tag{1.16}$$

From here, the estimator for the unsaturated bound is constructed by selecting a minibatch comprising L elements of $\mathbf{y}$, chosen uniformly at random (with replacement), and computing

$$\frac{N}{L}\sum_{l=1}^{L}\mathbb{E}_{q\left(\mathbf{f}_{n(l)}\right)}\left[\log p\left(y_{n(l)}\,\middle|\,\mathbf{f}_{n(l)}\right)\right] - \mathcal{KL}[q(\mathbf{u})\,\|\,p(\mathbf{u})]\,, \quad n(l)\sim\text{Uniform}(1,...,N)\,. \tag{1.17}$$

The variance of this estimator can be controlled by choosing the value of L. A greater value of L reduces the variance, but increases the cost of computing the estimator. Conversely, a smaller value of L increases the variance while decreasing the cost of computing the estimator. Consequently, a trade off must be struck, and in practice L is often chosen to be on the order of 10^2 or 10^3.

The best way to optimise this bound in practice remains an open question, but substantial progress has been made. There is consensus that a gradient-based approach is best, but different studies take slightly different approaches. Hensman et al. (2013) performed *natural gradient ascent* (Amari, 1998) w.r.t. the parameters of q, and gradient ascent with momentum for the kernel parameters. Salimbeni et al. (2018) investigate a number of different schemes for optimising the unsaturated bound, noting in particular that the step size in the natural gradient ascent scheme must be *increased* throughout the course of learning in order to achieve optimal performance, and that combining ADAM (Kingma and Ba, 2015) with natural gradient ascent produces poor results. Recently, Adam et al. (2021) proposed an alternate parametrisation of $q(\mathbf{u})$ in which natural gradient ascent is performed. This parametrisation loosens the coupling between the optimal choice for $q(\mathbf{u})$ and the kernel parameters, and enabling faster and more stable learning in practice.

More generally, many authors do not employ natural gradients at all, and simply apply the ADAM optimiser jointly to the parameters of q and the kernel, such as Borovitskiy et al. (2020), and some place restrictions the covariance matrix of $q(\mathbf{u})$, for example Borovitskiy et al. (2021) require that it be diagonal. There are several important ways to parametrise $q(\mathbf{u})$, which I now detail.

The Centred and Non-Centred Parametrisations The *centred* parametrisation refers to choosing $q(\mathbf{u})$ to be

$$q(\mathbf{u}) := \mathcal{N}(\mathbf{u};\mathbf{m}_{\mathbf{u}}^{\mathrm{q}},\mathbf{C}_{\mathbf{u}}^{\mathrm{q}})\,, \tag{1.18}$$

where $\mathbf{m}$ and $\mathbf{C}$ are optimised directly. Conversely, the *non-centred* parametrisation refers to choosing

$$q(\mathbf{u}) := \mathcal{N}\left(\mathbf{u};\mathbf{m}_{\mathbf{u}}+\mathbf{U}^\top\mathbf{m}_{\varepsilon}^{\mathrm{q}},\mathbf{U}^\top\mathbf{C}_{\varepsilon}^{\mathrm{q}}\mathbf{U}\right)\,, \quad \mathbf{U}^\top\mathbf{U}=\mathbf{C}_{\mathbf{u}}. \tag{1.19}$$

The non-centred parametrisation can be interpreted as directly parametrising the approximate posterior distribution over $\varepsilon := \mathbf{U}^{-\top}(\mathbf{u} - \mathbf{m}_\mathbf{u})$ as $q(\varepsilon) := \mathcal{N}(\varepsilon; \mathbf{m}_\varepsilon^\mathrm{q}, \mathbf{C}_\varepsilon^\mathrm{q})$, whose prior distribution is $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

This terminology dates back to Gelfand et al. (1995) in a slightly different setting from ours. They would term the former parametrisation *centred* because the mass of the distribution is located around the parameter $\mathbf{m}$, whereas the mass in the latter is located around $\mathbf{m}_\mathbf{u} + \mathbf{U}^\top\mathbf{m}$ instead.[3] These parametrisations are alternatively known as the *unwhitened* and *whitened* respectively, owing to white noise being uncorrelated.

Which parametrisation yields an easier optimisation problem is context-specific, depending on the relative strength of the prior and observation model (Gorinova et al., 2020). There has been surprisingly little work specific to GPs conducted which investigates this trade off empirically – Adam et al. (2021) do however discuss it in the context of developing another parametrisation (discussed next), noting that they found it yielded better hyperparameter optimisation performance in their experiments. Some basic intuition can be gained here by observing that non-centred parametrisation for $q(\mathbf{u})$ is itself a function of the kernel. This means that $q(\mathbf{u})$ will change depending on the kernel for fixed $\mathbf{m}_\varepsilon^\mathrm{q}$ and $\mathbf{C}_\varepsilon^\mathrm{q}$. Clearly, given observations such as that made by Gorinova et al. (2020), this is not a uniformly beneficial property, but it *is* what separates the two approximations.

One unquestionable advantage of the non-centred parametrisation is simply that it requires fewer operations (in absolute terms) to compute the unsaturated bound than does the centred parametrisation. For example, consider the KL-divergence between $q(\mathbf{u})$ and $p(\mathbf{u})$ under each parametrisation:

$$\begin{aligned}\text{Centred: } &\frac{1}{2}\left[\mathrm{tr}\big(\mathbf{C}_\mathbf{u}^\mathrm{q}\mathbf{C}_\mathbf{u}^{-1}\big) + (\mathbf{m}_\mathbf{u}^\mathrm{q} - \mathbf{m}_\mathbf{u})\mathbf{C}_\mathbf{u}^{-1}(\mathbf{m}_\mathbf{u}^\mathrm{q} - \mathbf{m}_\mathbf{u}) - M - \log\det(\mathbf{C}_\mathbf{u}^\mathrm{q}\mathbf{C}_\mathbf{u}^{-1})\right], \\ \text{Non-Centred: } &\frac{1}{2}\left[\mathrm{tr}(\mathbf{C}_\varepsilon^\mathrm{q}) + \|\mathbf{m}_\mathbf{u}^\mathrm{q}\|_2^2 - M - \log\det\mathbf{C}_\varepsilon^\mathrm{q}\right].\end{aligned}$$

The simplification occurs here because $p(\varepsilon) = \mathcal{N}(\varepsilon; \mathbf{0}, \mathbf{I})$. Similarly, the approximate posterior marginal distributions over a vector of random variables $\mathbf{f}$ are

$$\begin{aligned}\text{Centred: } &\mathcal{N}(\mathbf{f}; \mathbf{m}_\mathbf{f} + \mathbf{C}_\mathbf{fu}\Lambda_\mathbf{u}(\mathbf{m}_\mathbf{u}^\mathrm{q} - \mathbf{m}_\mathbf{u}), \mathbf{C}_\mathbf{f} - \mathbf{C}_\mathbf{fu}\Lambda_\mathbf{u}(\mathbf{C}_\mathbf{u} - \mathbf{C}_\mathbf{u}^\mathrm{q})\Lambda_\mathbf{u}\mathbf{C}_\mathbf{uf}) \\ \text{Non-Centred: } &\mathcal{N}\big(\mathbf{f}; \mathbf{m}_\mathbf{f} + \mathbf{C}_\mathbf{fu}\mathbf{U}^{-1}\mathbf{m}_\varepsilon^\mathrm{q}, \mathbf{C}_\mathbf{f} - \mathbf{C}_\mathbf{fu}\mathbf{U}^{-1}(\mathbf{I} - \mathbf{C}_\varepsilon^\mathrm{q})\mathbf{U}^{-\top}\mathbf{C}_\mathbf{uf}\big).\end{aligned}$$

[3] More precisely, I refer to the former parametrisation as centred because $\mathbf{m}_\mathbf{u}^\mathrm{q}$ is located at the centre of the ellipse formed by any contour of constant probability density under $q(\mathbf{u})$. Gelfand et al. (1995) do not themselves provide this specific formalisation of the terminology, but it seems appropriate.

where $\mathbf{U}^\top\mathbf{U} = \mathbf{C_u}$ as before. The most important difference in this case is the need only to compute $\mathbf{C_{fu}}\mathbf{U}^{-1}$ under the non-centred parametrisation, as opposed to $\mathbf{C_{fu}}\Lambda_\mathbf{u} = \mathbf{C_{fu}}\mathbf{U}^{-1}\mathbf{U}^{-\top}$ under the centred parametrisation.

The Pseudo-Observation Parametrisation Panos et al. (2018) and Adam et al. (2021) both investigate parametrisations for $q(\mathbf{u})$ which are (morally speaking) of the form

$$q(\mathbf{u}) \propto \mathcal{N}(\mathbf{u}; \mathbf{m_u}, \mathbf{C_u})\, \mathcal{N}(\mathbf{y}^\mathrm{q}; \mathbf{u}, \mathbf{S}^\mathrm{q})\,, \tag{1.20}$$

where $\mathbf{y}^\mathrm{q} \in \mathbb{R}^M$, $\mathbf{S}^\mathrm{q} \in \mathbb{S}_+^M$, and $\mathbb{S}_+^M$ denotes the positive semi-definite matrices of size $M \times M$. Phrased differently, they parametrise the approximate posterior over the pseudo-points as the exact posterior of a surrogate model over $\mathbf{u}$, in which the prior is that of the actual model, and the observation model has density $\mathcal{N}(\mathbf{y}^\mathrm{q}; \mathbf{u}, \mathbf{S}^\mathrm{q})$. Consequently, I refer to $\mathbf{y}^\mathrm{q}$ and $\mathbf{S}^\mathrm{q}$ as *pseudo-observation parameters*.

Panos et al. (2018) and Adam et al. (2021) take somewhat different approaches to working with parametrisations of this form. Panos et al. (2018) constrain $\mathbf{S}^\mathrm{q}$ to be diagonal, and perform stochastic gradient ascent. This approach has the property that for $N = M$, the optimal approximate posterior is within the family. The hope is that when there are enough pseudo-points the approximation will be nearly optimal, but that this optimum will be easier to obtain because there are only $\mathcal{O}(M)$ variational parameters to optimise, rather than $\mathcal{O}(M^2)$.

Conversely, Adam et al. (2021) parametrise the observation model in terms of its natural parameters $([\mathbf{S}^\mathrm{q}]^{-1}\mathbf{y}^\mathrm{q}, -\frac{1}{2}[\mathbf{S}^\mathrm{q}]^{-1})$, and perform natural gradient ascent w.r.t. these. Crucially they allow $\mathbf{S}^\mathrm{q}$ to be dense, meaning that they can recover the optimal approximate posterior. As discussed before, the primary advantage of this parametrisation is the reduced coupling between the parameters of $q(\mathbf{u})$ and the kernel parameters, which can lead to faster convergence during learning.

I build on this family of parametrisations in Chapter 4.

1.3.3 The Saturated Bound

Thus far I have discussed what is known as the unsaturated bound, which is amenable to iterative gradient-based optimisation, can work with non-Gaussian observation models, and scales well to very large data sets through mini-batching. However, in some situations it is possible to derive a closed-form expression for the optimum, circumventing the need for iterative optimisation.

Specifically, subject to the constraint imposed in Eq. (1.8), if the observation model is of the form

$$p(\mathbf{y} \mid \mathbf{f}) := \prod_{n=1}^{N} p(\mathbf{y}_n \mid \mathbf{f}_n) = \mathcal{N}(\mathbf{y}; \mathbf{f}, \mathbf{S}) \tag{1.21}$$

where $\mathbf{S} \in \mathbb{R}^{N \times N}$ is a positive-definite diagonal matrix, then the $q(\mathbf{u})$ which maximises the unsaturated bound for any particular set of kernel parameters and pseudo-inputs is:

$$q(\mathbf{u}) \propto \mathcal{N}(\mathbf{y}; \mathbf{C_{fu}}\Lambda_\mathbf{u}\mathbf{u}, \mathbf{S})\,\mathcal{N}(\mathbf{u}; \mathbf{0}, \mathbf{C_u})\,. \tag{1.22}$$

Further, the value of the unsaturated bound at this optimum is

$$\mathcal{L} = \log \mathcal{N}(\mathbf{y}; \mathbf{m_f}, \mathbf{C_{fu}}\Lambda_\mathbf{u}\mathbf{C_{uf}} + \mathbf{S}) - \frac{1}{2}\mathrm{tr}\big(\mathbf{S}^{-1}(\mathbf{C_f} - \mathbf{C_{fu}}\Lambda_\mathbf{u}\mathbf{C_{uf}})\big)\,, \tag{1.23}$$

and is known as the *saturated bound*. It can be computed using only $\mathcal{O}(NM^2)$ operations using the matrix inversion and determinant lemmas, or through the *projection trick* described by Bruinsma et al. (2020) in a slightly different context.

The saturated bound requires a pass over the entire training data set each time it or its gradient w.r.t. the kernel parameters and pseudo-input locations is required. This means that its use is limited to situations in which only a moderate number of observations are available – for example in Chapter 3 I utilise it on a problem comprising $\mathcal{O}(10^5)$ training examples. The restriction that the observation model must be Gaussian limits application to non-linear regression under Gaussian noise. Regardless, the saturated bound remains interesting for a couple of reasons. Firstly, when the above criteria *are* fulfilled, textbook quasi-Newton optimisation algorithms such as BFGS (see e.g. Nocedal and Wright (1999)) can be employed to find good kernel parameters, generally leading to convergence in a hundred or so iterations with robust automatic procedures for assessing convergence. This contrasts to the tens or hundreds of thousands which may be required when optimising the unsaturated bound via a variant of stochastic gradient ascent, and the increased difficulty in assessing convergence.

Moreover, it is a useful object to study when developing parametrisations for $q(\mathbf{u})$ – it is important to know whether a given parametrisation contains the optimal approximate posterior or not. It is for these reasons that I study this version of the bound in Chapter 3.

1.3.4 Alternative Formulations of Pseudo-Point Approximations

The precise formulation for pseudo-point approximations described above is the most popular one currently in use, and widely considered a gold-standard method, but it took a while for

the community to arrive at this particular formulation. The excellent reviews of Quiñonero-Candela and Rasmussen (2005) and Bui et al. (2017) provide additional context on this front.

Eq. (1.22) coincides with the exact posterior distribution over $\mathbf{u}$ under an *approximate model* with observation density $\mathcal{N}(\mathbf{y}; \mathbf{C}_{\mathbf{fu}}\Lambda_{\mathbf{u}}\mathbf{u}, \mathbf{S})$, and that the first term in Eq. (1.23) is the log marginal likelihood under this approximate model. It is well known that this approximate model is precisely the approximation employed by Seeger et al. (2003), known as the *Deterministic Training Conditional* (DTC). Despite their similarities, the DTC log marginal likelihood and the ELBO typically yield quite different kernel parameters and pseudo-inputs when optimised for – while the pseudo-inputs $\mathbf{z}$ are variational parameters in the variational approximation, and therefore not subject to overfitting (see section 2. of Bui et al. (2017)), they are model parameters in the DTC. For this reason, the variational approximation is widely favoured over the DTC.

However, the fact that the variational approximation is so closely related to a low-rank approximate model remains useful computationally, and is utilised in Sec. 3.5 to obtain algorithms which combine pseudo-point and state-space approximations in a manner which is both efficient, and easy to implement.

1.3.5 Benefits and Limitations

As discussed above, pseudo-point approximations perform well when many more observations of a GP are made than are needed to accurately describe its posterior; in such settings the approximation enables training and inference in GP models on extremely large data sets. This is often the case for regression tasks where the inputs are sampled independently from a light-tailed distribution. This is because the value of M required to maintain an accurate approximation does not increase too quickly as N increases—indeed Burt et al. (2019) showed that if the inputs $\mathbf{x}_n$ are sampled i.i.d. from a Gaussian, then the value of M required scales sub-linearly in N. The consequence of poor approximation will be poor predictions, in particular due to the ELBO forming a poor hyperparameter optimisation objective when it is loose. Recently, Artemev et al. (2021) and Burt et al. (2021) have attempted to address the quality of the log marginal likelihood approximation offered by the ELBO through Krylov subspace methods employed by Gibbs and MacKay (1997) and Gardner et al. (2018b).

An especially pathological case for pseudo-point approximations are time series. Bui and Turner (2014) noted that because the interval in which the observations live typically grows linearly in N. Consequently the number of pseudo-points M required to maintain a good

approximation must grow linearly in N, so the cost of accurate approximate inference using pseudo-point methods is really $\mathcal{O}(N^3)$ in this case. Indeed Tobar (2019) formalised this, showing that the number of the pseudo-points per unit time must not drop below a rate, analogous to the Nyquist-Shannon rate, if an accurate posterior approximation is to be maintained as N grows. Appropriately, different classes of approximation have been developed for time series which exploit entirely different kinds of structure, and are discussed at length in Chapter 3.

The version of this approximation presented can only make marginal predictions efficiently, or joint predictions over the GP at small collections of inputs, as discussed above. This is primarily troublesome as it means that it is not possible to efficiently generate samples from the approximate posterior joint distribution at a large collection of test points. Wilson et al. (2021) recently proposed a technique for generating these samples efficiently, but it requires additional approximation. This technique is discussed further in Chapter 4, where it is utilised to perform approximate inference in a setting where access to sample paths is required.

1.4 Outline and Contributions

The rest of this thesis is divided into three chapters, and a summary.

Chapter Two concerns the abstractions used in the creation of software for working with GPs in practice. In particular it concerns software which enables the construction of new GPs from affine transformations of existing GPs. This is useful because many GPs in the literature are defined in this manner, but the software in use today does not reflect this. This chapter makes several contributions on this front:

- a mathematical formalism which describes the above, with a crucial composability property that ensures interoperability with existing algorithms and methods for exact and approximate inference and learning,
- a methodology and algorithm for implementing such software, and
- numerous case studies demonstrating the utility of the abstraction.

Chapter Three develops a pseudo-point approximation for spatio-temporal problems. It does this by showing how pseudo-point approximations can be combined with state-space approximations, which excel in time series problems. Its primary contribution is to show that the optimal approximate posterior has a Markov property, subject to some restrictions on the

locations of pseudo-inputs. Knowing that this is the best that is possible is of value in itself. It also means that one can simply apply a pseudo-point approximation and utilise standard algorithms for fast inference which exploit Markov structure, safe in the knowledge that no additional approximations have been introduced. There are numerous things that can be done immediately with this result, and this chapter focuses on spatio-temporal GPs with Gaussian observation models, in which the optimal approximate posterior is itself Gaussian.

Chapter Four investigates the potential for the use of GPs in the problem of decadal prediction. Existing models used by climate scientists have some technical restrictions, notably the need to utilise data on a grid. This restriction is especially notable in problems involving ocean data, which comprises mostly point measurements from roaming sensors. In this chapter I investigate the potential to extend an existing class of model used for decadal prediction by replacing a couple of components with GPs. The resulting model is able to handle un-gridded data, but the posterior distribution is highly non-Gaussian, and provides a challenging approximate inference problem. Several approaches to approximate inference are investigated on synthetic data sets, and an extension to the pseudo-observation paradigm proves useful. Experiments are also conducted on a subset HadIOD, a high-quality ocean data set comprising many hundred of millions of ocean surface temperature measurements, demonstrating the efficacy of the extension in practice.

Finally, **Chapter Five** provides a summary of the presented work, and discusses important open questions and future work it raises.

Chapter 2

The Gaussian Process Probabilistic Programme

2.1 Introduction

This chapter develops an abstraction which can be employed in the creation of software frameworks for GPs, and shows the benefits this affords one such framework in practice. This abstraction is centred on *affine transformations* of GPs, by which I mean processes induced by sampling from a given GP and applying an affine transformation to the sample. The result is another GP, so I refer to this property as *closure* – GPs are *closed* under affine transformation, and also linear transformation. This class of transformation is important because almost all GPs in the literature are specified recursively through some composition of affine transformations of other GPs. A few examples include additive GPs, GPs whose inputs are transformed, multi-output GPs, integrals of GPs used in probabilistic numerics, derivatives of GPs used in Bayesian optimisation, and convolutions of GPs.

I refer to the software available prior to the development of this work as *kernel-centric*. It exposes only a subset of the components of a model to the user and, while this abstraction has proven highly successful, it makes it hard to undertake some useful and seemingly-simple inference tasks in practice. For example, consider the model

$$f_1 \sim \mathcal{GP}(0, \kappa_1), \quad f_2 \sim \mathcal{GP}(0, \kappa_2), \quad f_3 := f_1 + f_2. \tag{2.1}$$

Note that f_3 is a linear transformation of f_1 and f_2, and is another GP – the precise details are expanded upon later. Existing frameworks typically make it straightforward to implement

such models, but only make it possible to condition on observations of f_3, and to inspect the posterior distribution over f_3. This is because they express the above model in terms of the kernel and marginal distribution of f_3:

$$f_3 \sim \mathcal{GP}(0, \kappa_3) \text{ where } \kappa_3 := \kappa_1 + \kappa_2. \tag{2.2}$$

However, it is equally meaningful to condition on observations of f_1, f_2, or both, and to interrogate the posterior distribution over either process – kernel-centric frameworks typically permit neither of these possibilities.[1] Phrased differently, kernel-centric frameworks fail to provide access to the joint distribution over the entire model, in favour of the marginal distribution over the final process.

Moreover, fundamentally it is the assumptions expressed in Eq. (2.1) that one cares about when modelling a phenomenon. Consider that, when one is first introduced to GPs and ways to combine kernels, one is typically told something along the lines of *the GP induced by the sum of two kernels is equivalent to the GP induced by summing samples from two independent GPs*. This translation is straightforward enough, and as a consequence does not cause too much trouble. A slightly more involved example is integration. Consider the Gaussian random variable g induced by

$$f \sim \mathcal{GP}(0, \kappa), \quad g := \int f(x) \, \mathrm{d}\pi(x) \tag{2.3}$$

for some measure π. In this example it is quite possible that a user may wish to make observations of f in order to infer something about g (Ghahramani and Rasmussen, 2003; O'Hagan, 1991), or observe g in order to infer properties of f (see e.g. Tanaka et al. (2019)), or some combination of the two. The precise form of the kernel is not obviously of interest to the modeller. That being said, in this example, g has variance

$$\iint \kappa(x, x') \, \mathrm{d}\pi(x) \, \mathrm{d}\pi(x'), \tag{2.4}$$

and the covariance between g and any point x in f is

$$\int \kappa(x, x') \, \mathrm{d}\pi(x'). \tag{2.5}$$

The exposure of these expressions to the users of software seems undesirable.

[1] Salvatier et al. (2016) have implemented the decomposition of the posterior over f_1 and f_2 under this kind of model, as it constitutes an important and well-known special-case. It is, however, limited to models of this form, and does not extend to allowing observations of f_1 or f_2.

What is required in this example is a software abstraction in which a user can directly express Eq. (2.3), and separately express which bits of f and g they are interested in performing inference on, given observations of f, g, or both. More generally what is needed is the ability to express a range of affine transformations of GPs, mix and match them in different ways depending on the situation, to provide new affine transformations, to condition on any component of a model, and to perform inference on any component of a model.

For a slightly more involved problem, consider the classical noisy regression problem, in which we wish to infer some latent function given noise-corrupted observations, where the noise is assumed to be i.i.d. and Gaussian. One reasonable approach to this problem is to place a GP prior f over the unknown function, another GP prior ε over a "noise process", and assume that the process of which we make observations, y, is the sum of these two processes. This problem is firmly within-scope for existing GP software. Now suppose that we also have access to some direct observations of f, as shown in Fig. 2.1. Libraries such as `AbstractGPs.jl` make this straightforward to do because they enable a different noise variance to be specified for each observation – other existing GP frameworks have similar features. So, again, this remains within-scope for existing frameworks, but is likely to be slightly less clean to implement.

Now suppose that we suspect that some known subset of our observations have an unknown systematic bias. We might model this with the following generative procedure:

$$\begin{aligned} f &\sim \mathcal{GP}(0, \kappa) \\ \varepsilon &\sim \mathcal{GP}\left(0, \text{Noise}\left(\sigma^2\right)\right) \\ y &= f + \varepsilon \\ b &\sim \mathcal{N}(0, 1) \\ g &= f + b \\ y_b &= g + \varepsilon \end{aligned} \tag{2.6}$$

As before, observations of y and f are made, but also y_b. The results of inference in this type of model are shown in Fig. 2.2. Inference in this problem might also be possible to handle within existing GP frameworks, perhaps through the use of a multi-output GP, but this approach has its limitations and has interpretability problems – see Sec. 2.7.1. Finally, it is only a small generalisation to suppose that the bias is not constant, but instead varies smoothly over time, as depicted in Fig. 2.3. Again, more work is required on the part of the person implementing the model to encode in the kernel information that could conceivably have been extracted in an automatic fashion from the described generative model.

Moreover, it is not hard to imagine other kinds of measurements which are even more involved – perhaps the biased measurement is actually an integral measurement, representing a noisy average of f over time. It could also be a derivative measurement, if a given sensor is only capable of measuring the change in a quantity through time. These are both linear transformations of the underlying process, so it ought to be possible to make it straightforward for them to be expressed in a given model.

In this chapter I develop the mathematical, algorithmic, and practical underpinnings of a simple abstraction which does exactly this. It builds directly on top of a standard kernel-centric framework, meaning that it adds more functionality, and does not necessitate choosing between the existing GP software designs and what is proposed here – they can be mixed and matched in whichever manner is most convenient in a given situation. Moreover, existing approaches to approximate inference on GPs with non-Gaussian observation models are trivially compatible with what is developed, as are pseudo-point approximations. The listings in Fig. 2.1, Fig. 2.2, and Fig. 2.3, comprise code which can be executed to construct a GP using a concrete implementation of this abstraction, `Stheno.jl`, written in the Julia programming language (Bezanson et al., 2012).[2]

2.1.1 How Should Abstractions Be Judged?

This work does not come equipped with any obvious numerical measures that one can compute in order to assess whether it is valuable. In this sense it departs from the usual situation encountered in machine learning, where such measures can usually be obtained in short order. Therefore, I lay out the basis on which the contribution in this chapter ought to be judged, thus motivating the content of the remainder of the chapter.

Firstly, note that the abstractions developed in this work are complementary to existing kernel-centric approaches – they co-exist within the same set of abstractions discussed in Sec. 2.4.1, and are essentially built on top of them. Consequently it is not a question of whether to replace kernel-centric concepts with what is discussed here, but whether these should be built in addition. So the question is simple: is implementing this software worth the effort?

As to its worth, it is necessary only to demonstrate the existence of useful models and associated inferences, that can be cleanly expressed with this process-centric abstraction but which cause difficulties in a kernel-centric abstraction. To this end, I have explored a variety of affine transformations, the kinds of GPs they produce, and how these relate to those in

[2] Available at https://github.com/JuliaGaussianProcesses/Stheno.jl

```
# Specify model.
f = @gppp let
    f = 1.5 * GP(SEKernel())
    ε = 0.5 * GP(WhiteKernel())
    y = f + ε
end

# Specify input locations.
x_f = GPPPInput(:f, rand(Uniform(-5, 5), 3))
x_y = GPPPInput(:y, rand(Uniform(-5, 5), 8))
x = vcat(x_f, x_y)

# Construct posterior process.
f_post = posterior(f(x), y)
```

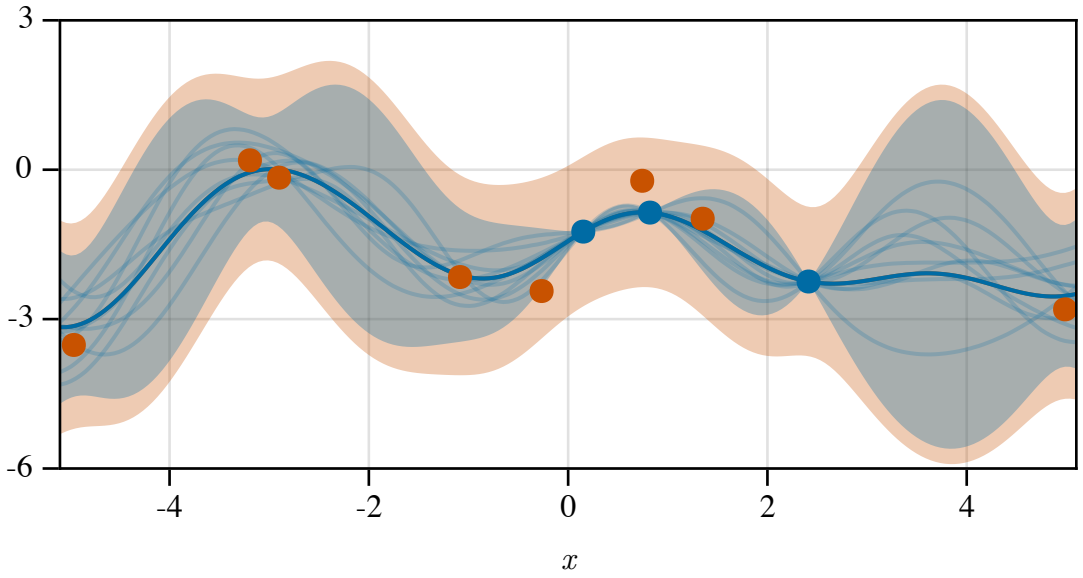


Fig. 2.1 "Partly-noisy" regression. A small number of exact observations of f are made, along with a larger number of observations of the noise-corrupted process y. `SEKernel` is the *exponentiated quadratic* covariance function with unit variance and length-scale. `WhiteKernel` is the white noise kernel given by $\kappa(x, x') := \sigma^2 \mathbb{I}(x = x')$. GPs are assumed to be zero-mean if no mean function is provided. Top: code specification of the generative model. Plotting code is suppressed for brevity. Bottom: posterior distribution over f (blue) and y (orange). Shaded regions are $\mu \pm 3\sigma$ under the posterior. Thin lines are samples from the posterior over f, bold line is posterior mean. Blue dots are observations of f, small red dots are observations of y. The posterior marginals of f have zero variance at exact observations of f, and reduced variance where observations of y are made.

```
f = @gppp let

    # Latent process and noisy observations.
    f = 1.5 * GP(SEKernel())
    ε = 0.5 * GP(WhiteKernel())
    y = f + ε

    # Noisy observations with unknown constant bias.
    b = 2.0 * GP(ConstantKernel())
    g = f + b
    yb = g + ε
end
```

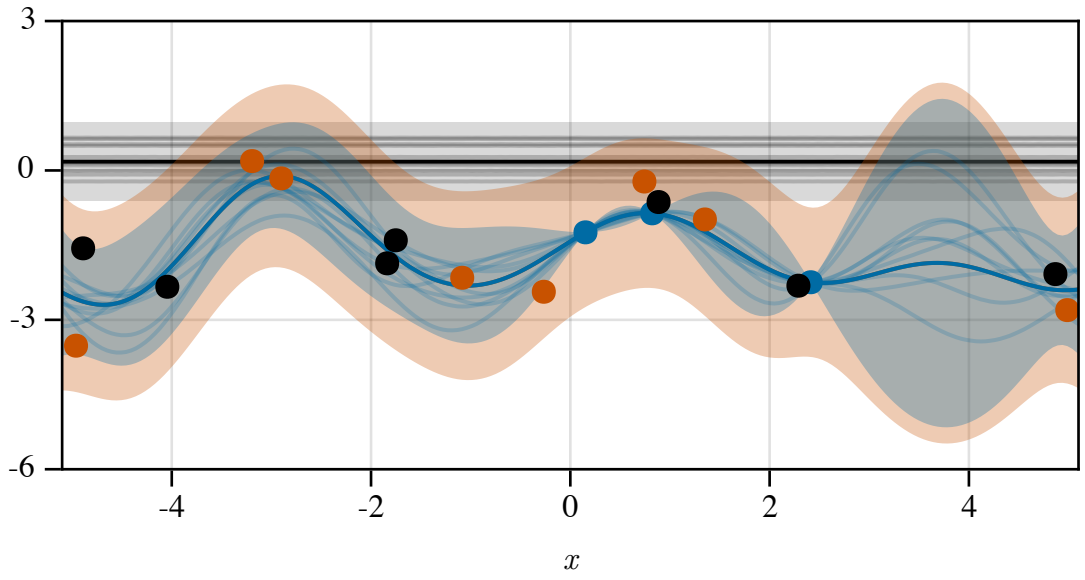


Fig. 2.2 "Partly-biased partly-noisy" regression. A small number of exact observations of f (blue) are made, along with a larger number of observations of the noise-corrupted process y (orange), a number of which are biased by some amount b (black). `SEKernel` is the *exponentiated quadratic* covariance function with unit variance and length-scale. `WhiteKernel` is the white noise kernel, given by $\kappa(x, x') := \sigma^2 \mathbb{I}(x = x')$. GPs are assumed to be zero-mean if no mean function is provided. Top: code specification of the generative model. Bottom: posterior distribution over f, y, and y_b. Shaded regions are $\mu \pm 3\sigma$ under the posterior. Thin lines are samples from the posterior over f, bold line is posterior mean. Blue dots are observations of f, orange dots are observations of y, black dots are observations of y_b. The thick black line is the posterior mean of the bias, and the shaded region around it is the $\pm 3\sigma$ central credible interval.

```
f = @gppp let

    # Latent process and noisy observations.
    f = 1.5 * GP(SEKernel())
    ε = 0.5 * GP(WhiteKernel())
    y = f + ε

    # Noisy observations with unknown varying bias.
    b = stretch(GP(SEKernel()), 0.3)
    g = f + b
    yb = g + ε
end
```

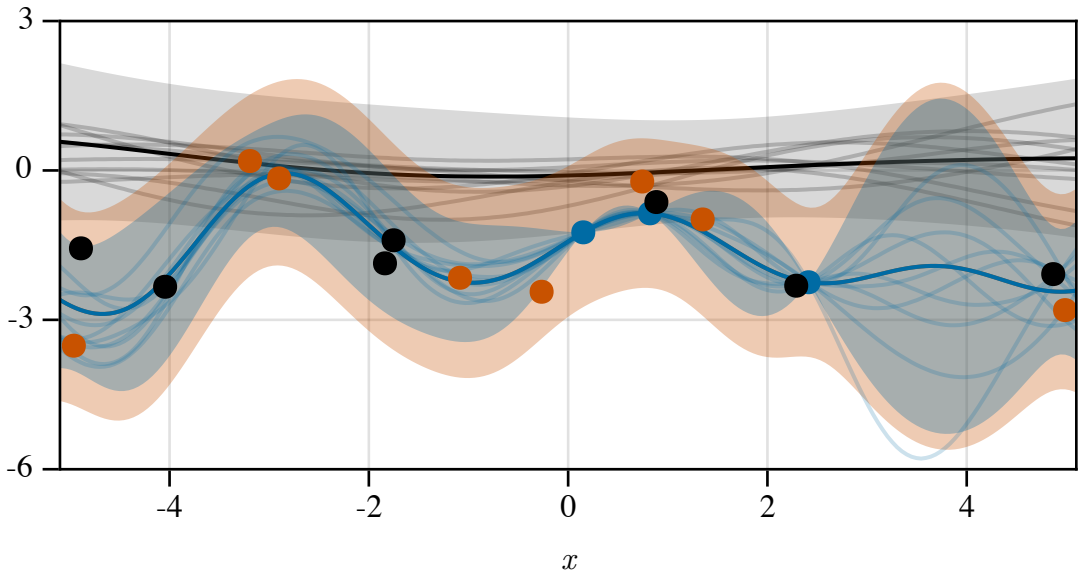


Fig. 2.3 "Partly-biased partly-noisy" regression. A small number of exact observations of f are made, along with a larger number of observations of the noise-corrupted process y, a number of which are biased by some amount b. `SEKenrel` is the *exponentiated quadratic* covariance function with unit variance and length-scale. `WhiteKernel` is the white noise kernel given by $\kappa(x, x') := \sigma^2 \mathbb{I}(x = x')$. GPs are assumed to be zero-mean if no mean function is provided. Top: code specification of the generative model. Bottom: posterior distribution over f (blue), y (orange), and b (black). Shaded regions are $\mu \pm 3\sigma$ under the posterior. Thin lines are samples from the posterior over f, bold line is posterior mean. Blue dots are observations of f, orange dots are observations of y, and black dots y_b. The thick black line is the posterior mean of b, and the shaded region around it is the $\pm 3\sigma$ central credible interval.

the literature. I provide a collection of simple case-studies involving generic models which are representative of general classes of model where the new software could be usefully deployed. Most real-world problems are much messier than the ones presented here, and those typically found in machine learning benchmark problems – covariates, multiple outputs with dependent non-Gaussian measurement error, mixtures of gridded and ungridded data, non-stationarity, and ubiquitous missing data abound in real problems. These case studies are therefore somewhat contrived, as this makes it easy to control which problems are present, and therefore to demonstrate how they are handled by the present work.

The extent of the effort involved in implementing software which utilises this abstraction is determined by a few considerations. One is the simplicity of the concepts and algorithms needed, another that of their implementation. Perhaps the definitive consideration though, is the extent to which what is developed here can interoperate with existing techniques for approximate inference, learning, and analysis, both in principle and in practice – it would be quite problematic if it were the case that all existing approximate inference algorithms needed to be re-derived, or completely new techniques developed. Fortunately, this is not the case. I make strong claims to this end, which will be clear in principle once the abstraction is established, and are further backed up by concrete examples.

One method of assessing the utility of an abstraction that I have intentionally chosen *not* to pursue is the counting of lines of code. While it is easy to count the number of lines of code that two implementations of the same functionality use, it is hard to use this number to say anything useful. For one, fair comparisons between two different pieces of code are hard to construct. Different programming languages and style guides will substantially effect the length of a piece of code, making comparing between code one person has written and another difficult. Even when comparing two pieces of code that a single author has written, in which the same coding style is used, the length of a piece of code is at best an indirect measure of important properties of code, such as the ease with which it can be understood, tested, maintained, and its correctness reasoned about. At worst, it is inversely related to these properties: it is quite easy to write brief code which is utterly impenetrable.[3] Instead, as alluded to above, I show how an implementation of the new abstraction produces readable code in numerous situations, that succinctly expresses the assumptions present in the model.

Intended Audience It is vital to consider the intended audience of the abstractions discussed in this chapter when assessing their utility. To do so, consider three individuals who might potentially be interested in GPs:

[3]Even to its author! Anyone who has written code is familiar with this experience.

- A data scientist who is interested in performing some quick analysis, and really wants access to the *fit-predict* style of interface that you might find in Scikit-learn (Pedregosa et al., 2011).
- A scientist who is willing to spend a bit of time exploiting the types of assumptions that GPs let them make about a given problem, knows what the log marginal likelihood is, and has enough knowledge to apply standard off-the-shelf tools for gradient-based optimisation and parameter handling.
- A scientist working on GP methodology, for example developing new types of GPs for particular problems, exploiting structure to enable scalable inference in some important setting, or approximate inference methods for hierarchical models in which a GP forms a single component.

It is the latter two individuals that I anticipate benefiting from what is developed in this chapter. While the first individual might benefit from a fit-predict interface built on top of this work, they are unlikely to benefit from its direct use.[4] For example, the extended noisy regression example above is something that the second individual might be interested in utilising.

2.1.2 Collaborators

Work in this chapter related to Gaussian Process Probabilistic Programmes was conducted in collaboration with Wessel Bruinsma. Although I do not consider it a core contribution of this chapter, I note that the Julia Gaussian Processes ecosystem is joint work with multiple collaborators: Théo Galy-Fajou, ST John, David Widmann, Ross Viljoen, Tom Wright, Sharan Yalburgi, and Hong Ge.

2.2 The GPPP

All of the transformations discussed so far are affine. Thus, abstracting away the details of the specific affine transformations used in the examples provided previously, a Gaussian

[4] At the time of writing, the Julia Gaussian Processes community is actively developing such an interface.

process probabilistic programme (GPPP) has the following form:

$$\begin{aligned} f_1 &\sim \mathcal{GP}(m_1, \kappa_1) \\ f_2 &\sim \mathcal{GP}(m_2, \kappa_2) \\ f_3 &= \mathcal{A}_1(f_1) \\ f_4 &= \mathcal{A}_2(f_2, f_3) \end{aligned} \tag{2.7}$$

It comprises exactly two kinds of expressions:

- $f \sim \mathcal{GP}(m, \kappa)$ defines a new *atomic* GP, and should be read as "f is distributed according to a Gaussian process distributed with mean m and kernel κ". The first two lines above are such expressions.
- $f' = \mathcal{A}(f_1, f_2, ..., f_P)$ defines a new *derived* GP, and should be read as "f' is an affine transformation of $f_1, f_2, ..., f_P$". The last two lines above are such expressions. In general, a derived GP can depend on any GPs declared in the previous lines of the programme, but will often only depend on a subset of them.

A transformation $\mathcal{A}$, mapping from one vector space $\mathcal{V}$ to another $\mathcal{W}$, is affine if it can be written as the composition of a linear transformation $\mathcal{L} : \mathcal{V} \to \mathcal{W}$ and a translation:

$$\mathcal{A}f = \mathcal{L}f + b, \quad f \in \mathcal{V}, \quad b \in \mathcal{W}. \tag{2.8}$$

Recall that $\mathcal{L}$ is called linear if

$$\mathcal{L}(f + f') = \mathcal{L}f + \mathcal{L}f', \text{ and } \alpha(\mathcal{L}f) = \mathcal{L}(\alpha f), \quad \alpha \in \mathbb{R}, \quad f, f' \in \mathcal{V}. \tag{2.9}$$

$\mathcal{V}$ and $\mathcal{W}$ are the vector spaces in which samples from GPs live. For example, $\mathcal{V}$ will always be the vector space of functions mapping some domain $\mathcal{X}$ to $\mathbb{R}$.

Wherever $\mathcal{A}f$ is written, if f is a function *sampled* from a GP, then $\mathcal{A}f$ is the affine transformation of that sample. Conversely, if f *is* a GP (i.e., a distribution over such functions), then $\mathcal{A}f$ denotes the GP induced by transforming the GP using $\mathcal{A}$. Which one is being referred to will be clear from context.

I refer to the collection of all atomic and derived GPs in a given programme as its *components*. Atomic GPs are the basic building-blocks of a programme, while derived GPs enable the composition of atomic and derived GPs to construct more complicated processes. This is analogous to the manner in which simple kernels, such as the exponentiated quadratic, are composed via addition and multiplication operations in a kernel-centric framework

to construct complicated processes. These expressions may appear in any order, with the obvious exception that the first must specify an atomic GP. They are strictly ordered according to the order in which they are specified, and a derived GP may only depend on previously specified GPs.

Multi-Process Perspective I call the manner in which GPPPs have been described thus far the *multi-process perspective* on GPPPs, in which they are treated as a collection of GPs and affine transformations thereof.

2.2.1 The Single-Process Perspective

While the multi-process perspective is perhaps the most intuitive perspective on a GPPP, it can also be treated as a single GP. This single-process perspective is central in the derivation the mathematical tools required to perform exact and approximate inference in GPPPs, and to the design of composable software.

In particular, let f be a GPPP comprising component processes $f_1, ..., f_P$, with domains $\mathcal{X}_1, ..., \mathcal{X}_P$ respectively. Let its domain $\mathcal{X}$ be

$$\mathcal{X} := \cup_{p=1}^{P} \{p\} \times \mathcal{X}_p \tag{2.10}$$

where $\times$ is the Cartesian product. That is, an element in the domain of f is a 2-tuple (p, x). p is an index which specifies which component of f the input corresponds to, while x picks out a particular element of that component.

For example, the domain of the example programme Eq. (2.7) is

$$\mathcal{X} = \{(1, x) : x \in \mathcal{X}_1\} \cup \{(2, x) : x \in \mathcal{X}_2\} \cup \{(3, x) : x \in \mathcal{X}_3\} \cup \{(4, x) : x \in \mathcal{X}_4\}, \tag{2.11}$$

where $\mathcal{X}_1$ and $\mathcal{X}_2$ are determined by m_1, κ_1 and m_2, κ_2 respectively, and $\mathcal{X}_3$ and $\mathcal{X}_4$ by $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. An input $(3, x)$ would return $f_3(x)$. More generally, $f((p, x)) = f_p(x)$. Notation like this is used regularly – the elements of the 2-tuple are given names on the left-hand side of $=$, and these names used inside the definition of the function on the right-hand side of $=$. This is generally more convenient than writing expressions of the following form: *let* $z := (p, x)$*, then* $f(z) = f_{z_1}(z_2)$, where $z_1 = p$ and $z_2 = x$.

If f_p is an atomic process, then $\mathcal{X}_p$ is assumed known. Conversely, if f_p is derived through an affine transformation $\mathcal{A}_p$ of $f_1, ..., f_{p-1}$, then its domain is specified by $\mathcal{A}_p$ – examples of this will be presented in the subsequent sections.

Given this perspective, it just remains to find the mean and kernel of f, $m : \mathcal{X} \to \mathbb{R}$ and $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. These are specified inductively as follows, by considering a finite sequence of programmes of increasing length.

Base Case

A single-component GPPP f must clearly comprise only a single atomic process f_1 with known mean m_1 and kernel κ_1. The mean and kernel of f are therefore just $m((p, x)) = m_p(x)$ and $\kappa((p, x), (p', x')) = \kappa_p(x, x')$ respectively.

Induction Step

Given a $(P-1)$-component GPPP f with known mean m and kernel κ, we construct a new P-component programme f' through the introduction of a new component GP f_P, which may be either atomic or derived, and appending this to the collection component processes $f_1, ..., f_{P-1}$ of f. If f_P is atomic with mean m_P and kernel κ_P, then the mean and kernel of f' are

$$m'((p, x)) = \begin{cases} m((p, x)) & \text{if } p < P, \\ m_P(x) & \text{if } p = P, \end{cases} \tag{2.12}$$

and

$$\kappa'((p, x), (p', x')) = \begin{cases} \kappa((p, x), (p', x')) & \text{if } p, p' < P, \\ \kappa_P(x, x') & \text{if } p = p' = P, \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

This can be related to the multi-process perspective by considering the matrix of kernels and cross-kernels given by $\kappa_{pq}(x, x') = \kappa'((p, x), (q, x'))$:

$$\begin{bmatrix} \kappa_1 & \kappa_{12} & \dots & \kappa_{1(P-1)} & 0 \\ \kappa_{21} & \kappa_2 & \ddots & & \vdots \\ \vdots & \ddots & & \kappa_{(P-2)(P-1)} & \\ & & \kappa_{(P-1)(P-2)} & \kappa_{P-1} & 0 \\ 0 & \dots & & 0 & \kappa_P \end{bmatrix}$$

where 0 indicates the binary function with an appropriately-defined domain that returns zero everywhere. In addition to providing the collection of kernels and cross-kernels for the multi-process perspective, this view makes it quite clear how adding a new atomic component process affects the kernel of f'; it adds a non-zero function on the last element of the diagonal,

and zero functions everywhere else in the final row and column. A similar description exists for m'.

If the new component process f_P is instead derived through an affine transform $\mathcal{A}$, $f_P := \mathcal{A}f$, and $\mathcal{A} : \mathcal{V} \to \mathcal{W}$ is of the form

$$\mathcal{A}f = \mathcal{L}f + b, \tag{2.14}$$

for linear $\mathcal{L} : \mathcal{V} \to \mathcal{W}$ and $b \in \mathcal{W}$, then the mean function m' of f' is

$$m'((p,x)) = \begin{cases} m((p,x)) & \text{if } p < P, \\ \mathbb{E}[(\mathcal{A}f)(x)] & \text{if } p = P. \end{cases} \tag{2.15}$$

Let $\delta := f - m$, then the kernel κ' is

$$\kappa'((p,x),(p',x')) = \begin{cases} \kappa((p,x),(p',x')) & \text{if } p,p' < P, \\ \mathbb{E}[(\mathcal{L}\delta)(x)\delta((p',x'))] & \text{if } p = P, p' < P, \\ \mathbb{E}[\delta((p,x))\,(\mathcal{L}\delta)(x')] & \text{if } p < P, p' = P, \\ \mathbb{E}[(\mathcal{L}\delta)(x)(\mathcal{L}\delta)(x')] & \text{if } p = p' = P. \end{cases} \tag{2.16}$$

Each of the expectations in m' and κ' are functions of $\mathcal{A}$, m, and κ, quantities that are known by assumption. Some concrete examples are provided shortly. As before, by defining $\kappa_{pq}(x,x') := \kappa'((p,x),(q,x'))$, we can construct a matrix of kernels and cross-kernels:

$$\begin{bmatrix} \kappa_1 & \kappa_{12} & \cdots & \kappa_{1(P-1)} & \kappa_{1P} \\ \kappa_{21} & \kappa_2 & \ddots & & \vdots \\ \vdots & \ddots & & \kappa_{(P-2)(P-1)} & \\ & & \kappa_{(P-1)(P-2)} & \kappa_{(P-1)} & \kappa_{(P-1)P} \\ \kappa_{(P)1} & \cdots & & \kappa_{P(P-1)} & \kappa_P \end{bmatrix}$$

This now constitutes a complete methodology for inductively determining the mean and kernel of a GPPP. It provides a precise specification for the functionality which must be implemented to make any particular affine transformation available within this framework, the *transformation-specific quantities* in Eq. (2.15) and Eq. (2.16):

$$m'((P,x)) := \mathbb{E}[(\mathcal{A}f)(x)] \tag{2.17}$$

$$\kappa'((p,x),(P,x')) := \mathbb{E}[\delta((p,x))\,(\mathcal{L}\delta)(x')] \tag{2.18}$$

$$\kappa'((P,x),(P,x')) := \mathbb{E}[(\mathcal{L}\delta)(x)(\mathcal{L}\delta)(x')]\,. \tag{2.19}$$

Note that only three of the four expectations must be specified due to symmetry:

$$\mathbb{E}[\delta((p,x))\,(\mathcal{L}\delta)(x')] = \mathbb{E}[(\mathcal{L}\delta)(x')\delta((p,x))]\,.$$

With this in hand, I present concrete examples of a few common affine transformations.

2.3 An Extensible Library of Affine Transformations

There are a number of affine transformations which are utilised in just about every GP used in practice in some form or another. Implementing these within a GPPP framework enables the implementation of all the examples shown in this chapter. This is not an exhaustive collection though – in just the same way that a user can add a new kernel to many existing GP libraries, one can add a new affine transformation to the library by following the steps used to derive each of the transformations presented here.

Each derivation follows the same structure. It starts with a GPPP f, comprising $P-1$ component processes, $f_1, ..., f_{P-1}$, with mean function m and kernel κ. It proceeds to determine the mean function m' and kernel κ' of the GP f' induced by applying $\mathcal{A}$ to f to produce f_P, and appending it to $f_1, ..., f_{P-1}$ to form f'. This reduces to finding expressions for Eq. (2.17), Eq. (2.18), and Eq. (2.19). δ and $\mathcal{X}$ are defined as above, and $p < P$.

This same procedure applies to any affine transformation that one wishes to make available for use in a GPPP.

Multiplication of GPs by known constants and functions

The multiplication of a GP by a constant is used in almost every model to change the variance of the process. This is a special case of a more general operation: scaling by a known function. This is a linear transformation of the form

$$f_P(x) := \alpha(x) f_q(x) \tag{2.20}$$

for some $q \in \{1, ..., P-1\}$, and $\alpha : \mathcal{X}_q \to \mathbb{R}$. From this single-process perspective, this is

$$(\mathcal{L}f)(x) := \alpha(x) f((q,x)). \tag{2.21}$$

Simple manipulations yield the transformation-specific quantities:

$$\begin{aligned} m'((P,x)) &:= \alpha(x)\, m((q,x))\,, \\ \kappa'((p,x),(P,x')) &:= \kappa((p,x),(q,x'))\,\alpha(x'), \\ \kappa'((P,x),(P,x')) &:= \alpha(x)\,\kappa((q,x),(q,x'))\,\alpha(x'). \end{aligned}$$

This transformation can be used to construct heteroscedastic GPs, and to perform Bayesian linear regression with input-dependent weights (O'Hagan, 1978).

Translation

The addition of a known function to the q^{th} process in a GPPP constitutes an affine transformation of the GPPP:

$$(\mathcal{A}f)(x) := f((q,x)) + b\,(x), \quad b : \mathcal{X}_q \to \mathbb{R}.$$

The transformation-specific quantities are

$$\begin{aligned} m'((P,x)) &:= m((q,x)) + b\,(x) \\ \kappa'((p,x),(P,x')) &:= \kappa((p,x),(q,x')) \\ \kappa'((P,x),(P,x')) &:= \kappa((q,x),(q,x'))\,. \end{aligned}$$

Domain Transformation

It is well known that one may apply a transformation g to the input of a kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and produce another valid kernel (MacKay, 1998). In other words, $\kappa'(x,x') := \kappa(g(x)\,, g(x'))$ is a kernel provided that κ is a kernel. g may be arbitrary beyond the obvious technical requirement that its range be a subset of $\mathcal{X}$. The use of this transformation is ubiquitous throughout the GP literature, simple examples include linear transformations whereby a kernel which depends upon its inputs through $r^2(x,x') = (x-x')^T A(x-x')$ can be re-written in terms of a linear input transform $g(x) = Px$ where $A = P^T P$ and a kernel which depends only upon the Euclidean distance between its inputs (e.g. see (Rasmussen and Williams, 2006) or Snelson and Ghahramani (2012)), while more complicated examples include highly non-linear transformations such as those proposed by Calandra et al. (2016) and Wilson et al. (2016) in which g is a neural network.

This can in fact be seen as a linear transformation. Taking the single-process perspective, this transformation is

$$(\mathcal{L}f)(x) = f((q, g(x)))\,,$$

for $q \in \{1, ..., P-1\}$. A brief derivation reveals this to be linear in f as required:

$$\begin{aligned}\mathcal{L}(f + \dot{f})(x) &= (f + \dot{f})((q, g(x))) \\ &= f((q, g(x))) + \dot{f}((q, g(x))) \\ &= (\mathcal{L}f)(x) + (\mathcal{L}\dot{f})(x).\end{aligned}$$

The transformation-specific quantities are

$$\begin{aligned}m'((P, x)) &:= m((p, g(x)))\,, \\ \kappa'((p, x), (P, x')) &:= \kappa((p, x), (q, g(x')))\,, \\ \kappa'((P, x), (P, x')) &:= \kappa((q, g(x)), (q, g(x')))\,.\end{aligned}$$

An extremely simple example of this type of transformation is a GP with zero mean and kernel

$$\kappa(x, x') := \sigma^2 \exp\Big(-(x - x')^2 / 2\alpha^2\Big)\,,$$

i.e., the exponentiated quadratic with variance σ^2 and length-scale α. We can specify this process as shown in the listing in Fig. 2.4, and inspect both its posterior, and that of the "standardised" unit-variance and unit-length scale process from which it is derived. The intent of this example is to highlight the existence of this interpretation of this standard model, and to show how straightforward it is to analyse this type of model using a GPPP.

Addition of Pairs of GPs

The sum of a pair of Gaussian processes is itself another Gaussian process. From the multi-processes perspective, the (binary) addition operation yields the sum of the q^{th} and r^{th} processes

$$f_P := f_q + f_r.$$

From the single-process perspective, it is the following linear transformation of f:

$$(\mathcal{L}f)(x) = f((q, x)) + f((r, x)).$$

```
f = @gppp let
    f = GP(SEKernel())
    Af = (f ∘ x->x / α)
end
```

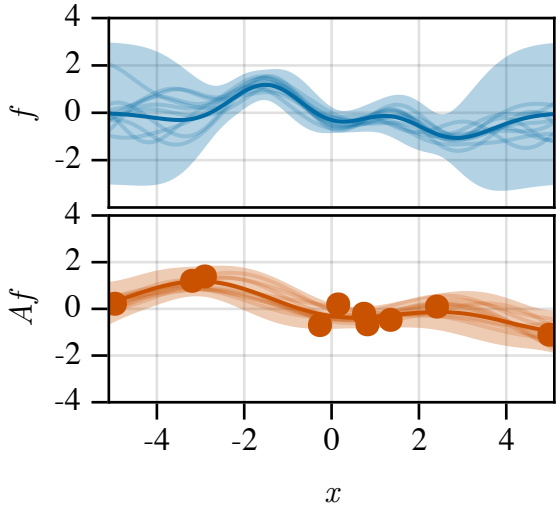


Fig. 2.4 A very simple example of the affine transform interpretation of an input transformation. Left: listing specifying generative model. We specify a latent process f, and make observations of a noisy-version of Af, to which a simple input transformation is applied. `f ∘ (x->x / α)` is the composition of f and the anonymous function which computes its input divided by α. Parentheses are unnecessary, and are included only for clarity. Right: Posterior of f (top) and Af (bottom) given observations of y (red dots).

It is defined provided that $\mathcal{X}_q = \mathcal{X}_r$. The expectation term in the mean function of f' is

$$\begin{aligned} m((P,x)) &:= \mathbb{E}[(\mathcal{A}f)(x)] \\ &= \mathbb{E}[f((q,x)) + f((r,x))] \\ &= \mathbb{E}[f((q,x))] + \mathbb{E}[f((r,x))] \\ &= m((q,x)) + m((r,x))\,. \end{aligned}$$

Similar derivations yield the results for the other transformation-specific quantities:

$$\begin{aligned} \kappa'((p,x),(P,x')) &:= \mathbb{E}[\delta((p,x))\,(\mathcal{L}\delta)(x')] \\ &= \kappa((p,x),(q,x')) + \kappa((p,x),(r,x'))\,, \\ \kappa'((P,x),(P,x')) &:= \mathbb{E}[(\mathcal{L}\delta)(x)(\mathcal{L}\delta)(x')] \\ &= \kappa((r,x),(r,x')) + \kappa((q,x),(q,x')) + \\ &\qquad \kappa((r,x),(q,x')) + \kappa((q,x),(q,x'))\,. \end{aligned}$$

Summation of Many GPs The above straightforwardly generalises to sums of a collection of R component processes, with indices $q_1, q_2, ..., q_R < P$. The corresponding transformation-specific quantities are

$$\begin{aligned} m'((P,x)) &= \sum_{r=1}^{R} m((q_r,x))\,, \\ \kappa'((p,x),(P,x')) &= \sum_{r=1}^{R} \kappa((p,x),(q_r,x'))\,, \\ \kappa'((P,x),(P,x')) &= \sum_{r=1}^{R}\sum_{s=1}^{R} \kappa((q_r,x),(q_s,x'))\,. \end{aligned}$$

A common special case of these transformations is that in which each component GP in the summation is independent *and* atomic. In this case the $\kappa((p,x),(p',x')) = 0$ whenever $p \neq p'$ or $p \notin \{q_1, ..., q_R\}$, meaning that the terms in κ' become

$$\begin{aligned} \kappa'((p,x),(P,x')) &= \mathbf{1}[p \in \{q_1, ..., q_R\}]\, \kappa((p,x),(p,x)) \\ \kappa'((P,x),(P,x')) &= \sum_{r=1}^{R} \kappa((q_r,x),(q_r,x')) \end{aligned}$$

respectively. Examples of GPs involving this transformation abound: the usual nonlinear regression under noise model in the listing of Fig. 2.1 is the addition of two independent GPs, and each of the other models in Fig. 2.2 and Fig. 2.3 also use this transformation.

Additive GPs Additive GPs (Duvenaud et al., 2011) can be seen as a composition of this transformation and a domain transformation. Specifically

$$\mathcal{L}f := \sum_{d=1}^{D} f((d, \phi_d(x))), \quad \phi_d(x) := x_d.$$

ϕ_d returns the d^{th} element of the D-dimensional input x. The additive GP model is quite restrictive. Gilboa et al. (2013) generalise it to arbitrary linear projections of the inputs, in which $\phi_m(x) := x^\top w_m$, $w_m \in \mathbb{R}$, $m \in \{1, ..., M\}$, lifting some of the restrictions.

Derivatives and Antiderivatives

Differentiation is linear. Here I discuss its use in a GPPP, in particular the use of *directional derivatives*.

Directional Derivatives Denote by $\mathcal{D}_v$ the directional-derivative operator in the direction $v \in \mathbb{R}^D$. When applied to a function $g : \mathbb{R}^D \to \mathbb{R}$ it produces another function $\mathcal{D}_v\, g : \mathbb{R}^D \to \mathbb{R}$, given by

$$(\mathcal{D}_v\, g)(x) := \lim_{h \to 0} \frac{g(x + hv) - g(x)}{h}.$$

Furthermore, letting $\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ be a binary function, $\mathcal{D}_v\, \kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ denotes the directional derivative operator applied only to the first argument, while $\kappa\, \mathcal{D}_v^\top : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ denotes the operator applied to the second argument:

$$(\mathcal{D}_v\, \kappa)(x, x') := \lim_{h \to 0} \frac{\kappa(x + hv, x') - \kappa(x, x')}{h},$$
$$(\kappa\, \mathcal{D}_v^\top)(x, x') := \lim_{h \to 0} \frac{\kappa(x, x' + hv) - \kappa(x, x')}{h}.$$

Assume that $\mathcal{D}_v\, (\kappa\, \mathcal{D}_v^\top) = (\mathcal{D}_v\, \kappa)\, \mathcal{D}_v^\top$, meaning $\mathcal{D}_v\, \kappa\, \mathcal{D}_v^\top$ can be written without ambiguity.[5]

Directional Derivative of a GPPP It is particularly helpful here to use the notation associated with the multi-process perspective, specifically

$$f_p(x) := f((p, x)),$$
$$m_p(x) := m((p, x)),$$
$$\kappa_{pq}(x, x') := \kappa((p, x), (q, x')).$$

In this notation, the directional derivative $\mathcal{D}_v$ of the q^{th} process of f is denoted $\mathcal{D}_v\, f_q$. The transformation-specific quantities are

$$m'((P, x)) = (\mathcal{D}_v\, m_q)(x)$$
$$\kappa'((p, x), (P, x')) = (\kappa_{pq}\, \mathcal{D}_v^\top)(x, x')$$
$$\kappa'((P, x), (P, x')) = (\mathcal{D}_v\, \kappa_{pq}\, \mathcal{D}_v^\top)(x, x').$$

Therefore, given programmes which compute m_p, κ_p, and $\kappa_{pp'}$, Automatic Differentiation (Baydin et al. (2017) provide a recent introduction and review) can be employed to compute m', κ'. An important use case for GP derivatives is in Bayesian Optimisation (Osborne et al., 2009; Wu et al., 2017a,b), where observations of both the objective function and its gradient are often available. In `Stheno.jl` we have implemented the operations required to construct the derivative of a GP using the *ForwardDiff.jl* library of Revels et al. (2016).

[5] Schwarz's theorem states that this assumption holds at any point where the second partial derivatives of κ are continuous.

Antiderivatives The derivative GP can also be used to model the integral of a function $g : \mathbb{R} \to \mathbb{R}$. Consider a function g and its integral G given by

$$G(x') := \int_a^{x'} g(x)\,\mathrm{d}x$$

where $x, x', a \in \mathbb{R}$. By the fundamental theorem of calculus

$$\frac{d}{dx}G(x) = g(x)\,.$$

By placing a prior directly over G one can model the integral of g over whatever range one requires by conditioning on observations of g and $G(a) = 0$. This is a useful technique since if one places a GP prior over g, to be able to work with its integral it is necessary to compute the integral of the mean and kernel of g, which is only analytically tractable for a handful of kernels. Moreover it yields a much simpler implementation than hand-coding the required integrals, all required computations can be handled using Automatic Differentiation. That this approach can be taken was noted by Wessel Bruinsma, and utilised by Bunker and Turner (2019).

Fig. 2.5 shows how the models described above can be constructed using `Stheno.jl`. A smooth prior is placed over f, and its derivative process constructed using the unary ∇ operator, which returns the derivative GP of its argument. This model is first used to infer a function and its derivative using observations of both placed at randomly chosen locations. The model on the right hand side of Fig. 2.5 is quite similar to that on the left, but uses the technique described above to infer the integral of f using observations of both f and its derivative process. The point that this pair of examples highlights is just how cleanly the differentiation of processes can be implemented withing a GPPP from the modeller's perspective.

Integral Transformations

As alluded to in the introduction of this chapter, in principle integral transformations are also possible to utilise inside a GPPP. For example, convolutions of the form

$$(\mathcal{L}_\phi f)(x) := \int \phi(y)\, f_q(x - y)\,\mathrm{d}y$$

```
f = @gppp let
    f = GP(SEKernel())
    df = derivative(f)
end
```

```
f = @gppp let
    F = GP(SEKernel())
    f = derivative(F)
end
```

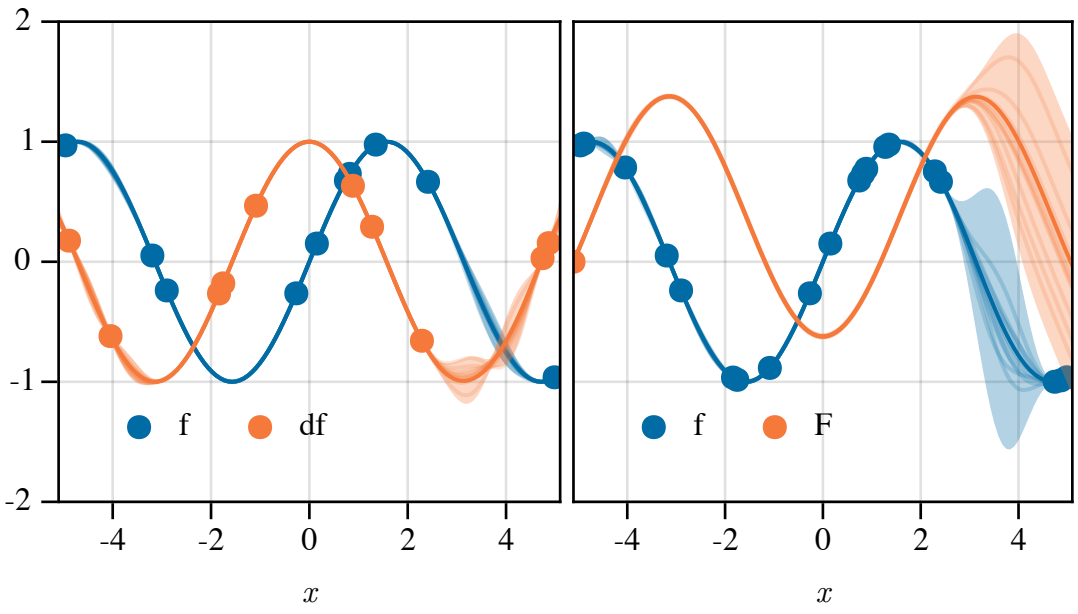


Fig. 2.5 GPPP models for the derivative and antiderivative of a function. The model specified in the top left places a smooth prior over a function f and its derivative process df. The right hand side is the same model, but is interpreted differently. Thick lines are posterior means, filled regions are posterior means $\pm 3\sigma$ under the posterior, thin lines are posterior samples, and dots are observations.

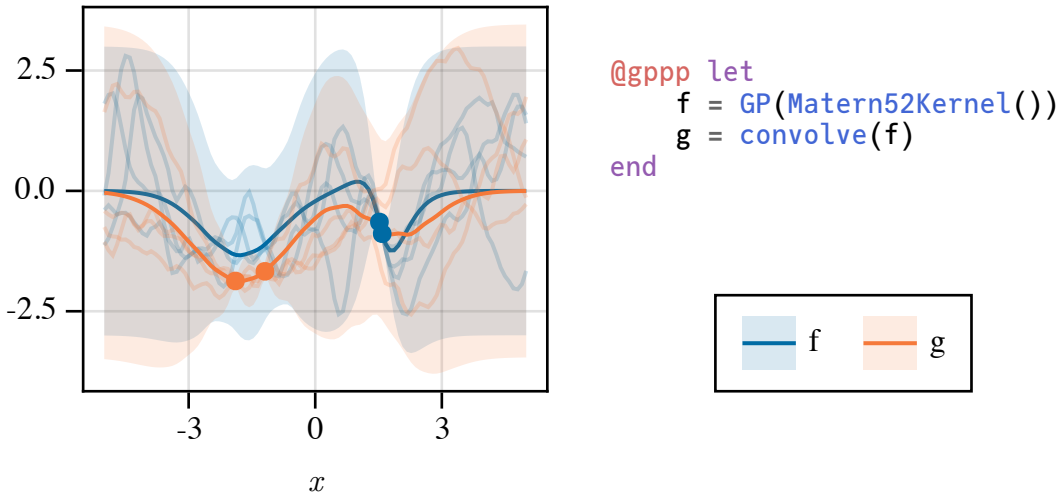


Fig. 2.6 Convolution of `f` with $\phi(x) := \exp(x^{-2})$ to produce `g`. Left: posterior over f and g given observations of both. Dots are observations of the correpsonding component of the GPPP. Top left: code to specify this GPPP.

can be incorporated, and produce the following transformation-specific quantities:

$$
\begin{aligned}
m'((P, x)) &= \int \phi(y)\, m_q(x - y)\, \mathrm{d}y, \\
\kappa'((p, x), (P, x')) &= \int \kappa_{pq}(x, x' - y')\, \phi(y')\, \mathrm{d}y', \\
\kappa'((P, x), (P, x')) &= \int \phi(y)\, \kappa_{pq}(x - y, x' - y')\, \phi(y')\ \mathrm{d}y\, \mathrm{d}y'.
\end{aligned}
$$

The only difficulty with integral transformations in the context of GPPPs is precisely the same difficulty found in prior work: computing the above integrals requires approximation in all but a handful of cases. This does not pose too much of a problem in one or two dimensions, where quadrature or cubature works reasonably well, but of course becomes troublesome in higher dimensions. For the sake of the example shown in Fig. 2.6, a simple operation `convolve` was implemented, which utilises Gauss-Hermite quadrature to approximate convolving a process with $\phi(x) := \exp(x^{-2})$. Implementing greater coverage of the possible integral transformations, including special cases where integrals are tractable, would be a valuable endeavour.

2.3.1 Some Curiosities

Thus far I have discussed affine transformations with clear practical utility. This section catalogues some affine transformations that are interesting, but for which I have failed to find great practical utility. I believe it is worthwhile to note their existence because it is straightforward to conceive of use cases for each of them.

Indexing

Denote by $\delta_{\mathbf{x}}$ the indexing transformation, parametrised by a vector of inputs $\mathbf{x} \in \mathcal{X}^N$ – when applied to a GP f it returns the (finite) N-dimensional GP whose domain is $\{1, ..., N\}$, and whose n^{th} dimension is simply $f(x_n)$. This transformation is linear, since

$$\begin{aligned}\delta_{\mathbf{x}}(\alpha_1 f_1 + \alpha_2 f_2)(n) &= (\alpha_1 f_1 + \alpha_2 f_2)(x_n) \\ &= \alpha_1 f_1(x_n) + \alpha_2 f_2(x_n) \\ &= \alpha_1(\delta_{\mathbf{x}} f_1)(n) + \alpha_2(\delta_{\mathbf{x}} f_2)(n).\end{aligned}$$

Let $\delta_{\mathbf{x}}$ be applied to the q^{th} process in a GPPP, then the transformation-specific quantities are

$$\begin{aligned}m'((P,n)) &= m((q,x_n))\,, \\ \kappa'((p,x),(P,n')) &= \kappa((p,x),(q,x_{x'}))\,, \\ \kappa'((P,n),(P,n')) &= \kappa((q,x_n),(q,x_{n'}))\,.\end{aligned}$$

This transformation would be very useful in the absence of the `AbstractGPs.jl` interface – this transformation allows one to pull out a finite-dimensional object from an infinite-dimensional one, and is as simple as it is due to the marginalisation property of GPs. However, because the `AbstractGPs.jl` interface itself includes an indexing operation, this one is largely redundant in the particular implementation presented here. If one were to implement a GPPP in a different context, where it is required to implement an interface which does not include finite-dimensional projection, it might be very useful.

Inference

Inference can also be written as a simple affine transformation: consider conditioning the component process f_q on observations $\mathbf{y} \in \mathbb{R}^N$ of another component process f_r at $\mathbf{x} \in \mathcal{X}^N$. The affine transformation

$$\mathcal{A}(f_q, f_r)(x) := f_q(x) + \kappa_{qr}(x, \mathbf{x})\,\mathbf{C}_{\mathbf{f}}^{-1}(\mathbf{y} - \mathbf{f}), \quad \mathbf{f}_n := f_r(\mathbf{x}_n)\,,$$

where $\kappa_{qr}(x, \mathbf{x}) \in \mathbb{R}^{1\times N}$ is vector of covariances between $f_q(x)$ and $\mathbf{f}$, produces a new component process which is the posterior over f_q. This can be verified by considering the transformation-specific quantities. Let $\mathbf{m}_n := m((r, x_n))$, then

$$\begin{aligned} m'((P, x)) &= m_q(x) + \kappa_{qr}(x, \mathbf{x})\, \mathbf{C}_{\mathbf{f}}^{-1}(\mathbf{y} - \mathbf{m}), \\ \kappa'((p, x), (P, x')) &= \kappa_{pq}(x, x') - \kappa_{pr}(x, \mathbf{x})\, \mathbf{C}_{\mathbf{f}}^{-1} \kappa_{rq}(\mathbf{x}, x')\,, \\ \kappa'((P, x), (P, x')) &= \kappa_q(x) - \kappa_{qr}(x, \mathbf{x})\, \mathbf{C}_{\mathbf{f}}^{-1} \kappa_{rq}(\mathbf{x}, x')\,. \end{aligned}$$

Observe that the mean function and kernel $\kappa'((P, x), (P, x'))$ are the standard posterior mean function and kernel.

This transformation is known as *Matheron's Rule*. While Hoffman and Ribak (1991) discuss this transformation in the context purely of producing samples from a Gaussian process, and Doucet (2010) elucidates the use of their procedure to sample from a variety of Gaussian systems, none make the important conceptual step of identifying the operation as an affine transformation, indistinct from any other discussed in this work other than in its fundamentally important probabilistic interpretation as computing a posterior process. Wilson et al. (2020, 2021) have recently utilised this trick in order to generate approximate samples from a posterior GP, by forming a finite-dimensional approximation to the prior.

In early implementations of `Stheno.jl` this transformation was utilised to perform inference. While superficially appealing, this approached turned out to have some problems in practice. The first is redundancy. The `AbstractGPs.jl` interface contains a `posterior` function, which works perfectly well for GPPPs via the single process perspective, and it is hard to imagine any kernel-centric framework that would not need to have similar functionality. While this inference transformation does provide a superset of the functionality offered by the `posterior` function, it is unclear what utility this provides. For example, this transformation would allow one to construct a programme in which observations are made of both a process and its posterior. Probably the most important issue pertains to composability – if a GPPP has its own bespoke way to perform exact inference which differs from the framework that it is built on top of, it will be hard to recycle existing functionality not designed with GPPPs specifically in mind.

A related transformation can be utilised to perform approximate inference with pseudo-points by replacing the fixed vector of observations $\mathbf{y}$ with a random variable $\hat{\mathbf{u}} \sim q(\mathbf{u})$, and $\mathbf{x}$ with $\mathbf{z}$.

Multiplying Kernels

It is well understood that the product of two kernels $\kappa_p(x, x') = \kappa_q(x, x')\,\kappa_r(x, x')$ is itself a kernel. It is not, however, the case that the product of two GPs f_q and f_r is another GP f_p. This begs the question as to whether or not there is an interpretation of the product of two kernels in terms of an affine transformation of a GP. It turns out that such an interpretation exists, although its practical utility is unclear. Assume that κ_r admits the eigendecomposition

$$\kappa_r(x, x') = \sum_{n=0}^{\infty} \lambda_n \phi_n(x)\,\phi_n(x')\,,$$

for $0 < \lambda_1 \leq \lambda_2 \leq ...$ and eigenfunctions $\{\phi_1, \phi_2, ...\}$. Given the sequence of i.i.d. GPs $\left\{f_q^{(n)} \sim \mathcal{GP}(0, \kappa_q)\right\}_{n=0}^{\infty}$, it is the case that

$$f(x) = \sum_{n=0}^{\infty} \sqrt{\lambda_n}\phi_n(x)\, f_q^{(n)}(x) \sim \mathcal{GP}(0, \kappa_p)\,.$$

That f has zero mean is almost immediate:

$$m(x) = \mathbb{E}[f(x)] = \sum_{n=1}^{\infty} \sqrt{\lambda_n}\phi_n(x) \underbrace{\mathbb{E}\left[f_q^{(n)}(x)\right]}_{=0} = 0.$$

That the kernel of f is the product of κ_r and κ_q follows quickly:

$$\begin{aligned}
\kappa(x, x') &= \mathbb{E}[f(x)\, f(x')] \\
&= \sum_{n=1}^{\infty}\sum_{n'=1}^{\infty} \sqrt{\lambda_n\lambda_{n'}}\phi_n(x)\,\phi_{n'}(x')\,\mathbb{E}\left[f_q^{(n)}(x)\, f_q^{(n')}(x')\right] \\
&= \sum_{n=1}^{\infty} \lambda_n\phi_n(x)\,\phi_n(x')\underbrace{\mathbb{E}\left[f_q^{(n)}(x)\, f_q^{(n)}(x')\right]}_{=\kappa_q(x,x')} \\
&\quad + \sum_{n=1}^{\infty}\sum_{n'\neq n} \sqrt{\lambda_n\lambda_{n'}}\phi_n(x)\,\phi_{n'}(x')\underbrace{\mathbb{E}\left[f_q^{(n)}(x)\, f_q^{(n')}(x')\right]}_{=0} \\
&= \kappa_q(x, x') \sum_{n=1}^{\infty} \lambda_n\phi_n(x)\,\phi_n(x') \\
&= \kappa_q(x, x')\,\kappa_r(x, x')\,.
\end{aligned}$$

Similar calculations show that the covariance between $f(x)$ and $f_q^{(n)}$ is

$$\mathbb{E}\left[f(x)f_q^{(n)}(x')\right] = \lambda_n \phi_n(x)\kappa_q(x, x').$$

To summarise, we have shown that the product of two kernels can be understood in terms of the limit of the weighted sum of a large number of i.i.d. GPs with one of the kernels, where the weights are determined by the eigenvalues and eigenfunctions of the other kernel. While interesting, the utility of this interpretation is unclear, since it involves infinitely many GPs and requires that the eigendecomposition of κ_r be available, which is only the case for a limited number of kernels.

2.4 Practical Considerations

This section addresses various important practical aspects of an implementation of the GPPP abstraction.

2.4.1 The Primary AbstractGPs.jl Interface

The Julia Gaussian Processes organisation has developed a set of interfaces which specify what functionality it is expected that different kinds of GPs should provide. Functionality for exact inference, sparse approximations, and approximate inference under non-Gaussian likelihoods is implemented using the functionality this interface requires. Consequently, any GP which implements one of these interfaces can immediately make use of this functionality – the GPPP implementation utilised in this chapter is one such example. This section introduces these interfaces, and explains their design.

This interface is utilised in several places in the ecosystem of software packages that have been developed, including that on which this chapter is based, so it is necessary to introduce it here. The interfaces described in this section rely heavily on *multiple dispatch*. I anticipate that many readers will not have encountered it before, so an introduction to it is provided in Chapter A.

This interface is specified at a slightly higher level of abstraction than GP frameworks often sit. As alluded to, a typical GP framework specifies an interface for kernels, and implements a GP object in terms of a kernel. The primary `AbstractGPs.jl` interface does not say anything in particular about kernels or mean functions, rather it requires that any object

```
fx = f(x, S) # marginal distribution over f at x, plus noise
y = rand(fx) # sample from fx
marginals(fx) # compute vector of marginal distributions of fx
logpdf(fx, y) # compute log marginal probability of y under fx
f_post = posterior(fx, y) # compute posterior over f
```

Fig. 2.7 An example listing

calling itself a GP must be able to perform certain operations when certain additional data are provided:

1. compute the log marginal probability of a vector of observations,
2. generate samples at a finite collection of inputs,
3. condition on observations and produce a new GP which represents the posterior distribution,
4. compute the marginal distribution at a collection of inputs.

It is simplest to demonstrate this through an example; consider Fig. 2.7. In this example f is something which supports the `AbstractGPs.jl` interface, $\mathbf{x}$ is a vector of inputs, $\mathbf{y}$ a vector of real-valued observations, and $\mathbf{S}$ a covariance matrix. The first line of the listing constructs an object which represents the random variable given by the sum of the multivariate Gaussian given by f at inputs $\mathbf{x}$ and a $\mathcal{N}(\mathbf{0}, \mathbf{S})$-distributed noise vector. This object provides a useful way to specify a finite-dimensional subset of the random variables in the GP, on which actual computation can be performed.

The subsequent lines provide examples of these operations: sample generation, construction of the marginals, log marginal probability density computation, and exact posterior inference. f_{post} is another object which implements the `AbstractGPs.jl` interface, meaning that all of the operations which can be applied to f can be applied to it. This enables the generation of posterior samples, construction of posterior marginals, computation of posterior predictive log marginal probability densities, and conditioning on further observations.

The purpose of providing this interface at this level of abstraction is to state *what* a GP must be able to do, not *how* it must do it. For example, while f will often be a data type which explicitly contains a mean function and kernel, and utilises these to implement the required operations, this need not be the case. Indeed, f_{post} is a type containing f itself,

and some pieces of data computed from f, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{S}$, as opposed to a type containing a posterior-mean function and posterior-kernel, for example.

There are immediate benefits to this design choice. For example, it is easy to achieve greater computational efficiency by avoiding repeated computations in some important situations. For example, both the mean vector and covariance matrix of f_{post} at $\mathbf{x}_*$ are required in order to implement `rand`. These two quantities both require the computation of the covariance matrix between f at $\mathbf{x}$ and $\mathbf{x}_*$. If all that one has access to is a mean function and kernel, and it is not known that they correspond to the mean function and kernel of the same posterior GP, it is not possible to exploit this shared computation. Conversely, if we know that a particular GP represents a posterior, we can specialise `rand` accordingly.

More generally, it frees up the implementer from having to worry about providing a kernel for a given GP when this is not the most convenient option, instead focusing attention on what operations it is necessary to implement in order to make it possible to do inference.

However, note that I am *not* claiming that it is impossible to produce an efficient implementation of any of the GPs discussed by explicitly implementing a kernel. Indeed, I am entirely confident that it will always be possible to provide a performant implementation given a sufficiently powerful type system, as it will always be possible to specialise on your particular kind of kernel. What I do claim, however, is that it is not always most convenient to provide an explicit kernel for a given GP, but to implement the important operations directly. This is the character of the class of GP introduced in this chapter.

Furthermore, is it certainly true that *specialisation* of the implementations of the functions in Fig. 2.7 is essential to achieve performance in some situations. For example, there are scenarios in which computing the mean vector and covariance matrix, and using these to generate samples of compute log marginal probability densities, is simply the wrong way to go about things. In those situations, it is essential to exploit the structure in the GPs to implement these functions efficiently, and it is convenient to do this specialisation at the level of the GP.

A near-trivial example of this is Bayesian linear regression. The model utilised in Bayesian linear regression is simply a finite-dimensional GP, so it can implement the `AbstractGPs.jl` interface, and be treated like any other GP. Linear-time algorithms are available for all of the operations listed in Fig. 2.7, so specialisation is required in order to avoid naïve cubic-time algorithms that are required in general. Chapter 3 pertains to another class of GPs in which this need for specialisation is paramount.

This is, of course, not the only location in which one could specialise implementations to exploit model structure. Another potential option is through specialised matrix types. For example, the covariance matrix associated with a Bayesian linear regression at a collection of inputs will be at most rank D, where D is the input dimension. This information could be used to provide a lazy matrix type with specialised implementations of the operations neessary to implement the `AbstractGPs.jl` interface in a generic way.

I chose not to take this route because specifying the interface at the GP level ensures that the implementer is free to implement the operations needed to do useful things with their GP in whichever way is most straightforward. Conversely, it may not always be apparent how the structure in a particular GP can be expressed through the covariance matrix. The GPs discussed in Chapter 3 are such an example, in which the algorithms for implementing the `AbstractGPs.jl` interface are well understood, but it is not immediately obvious how they map on to structure in covariance matrices.

2.4.2 The Other Interfaces

A notable exclusion from the above interface is covariance matrix computation. As suggested above, this is because not all GPs are able to construct one efficiently, but they have algorithms for implementing the important operations which do not require the use of a covariance matrix. However, in many cases it is perfectly reasonable to compute the covariance matrix.

This brings us to the *Secondary AbstractGPs.jl Interface*. This is simply the Primary Interface, with the addition of a function to compute the covariance matrix, shown in Fig. 2.8.

The point of having these separate interfaces is to guide those who use objects which implement the interface as to which operations they can rely on being implemented for *any* GP – anything in the Primary Interface. Phrased differently, the different interfaces instruct users to avoid asking for the covariance matrix where possible, and to instead utilise higher-level functionality. If they are unable to do this, it will simply restrict the kinds of GPs they can use to those which implement the Secondary Interface.

The final interface is an internal one, which makes it easy for new GP implementers to provide implementations of the Primary and Secondary interfaces if there is no special structure to exploit in their particular GP, and the best option available is simply to compute the covariance matrix directly. The methods in this interface are shown in the bottom half of Fig. 2.8, where f is again a GP, and $\mathbf{x}$ a collection of inputs.

It is this interface that is utilised throughout this chapter.

```
# Secondary API
cov(fx) # compute the covariance matrix

# Internal Dense Interface
mean(f, x) # mean vector at x
cov(f, x, x′) # cross-covariance matrix between x and x′
var(f, x) # diagonal of cov(f, x, x)
```

Fig. 2.8 Another example listing

2.4.3 Other Important Implementation Details

Lazy Calculation The previous sections lay out the relationship between all of the processes in a GPPP. While a practical implementation of a GPPP could implement them directly, for example building up a matrix of mean functions / kernels and querying them when needed, the approach adopted in practice differs in a couple of regards. These differences are purely practical in nature, and remain faithful to the preceding sections.

In particular, the definition of a GPPPP suggests that a collection of P^2 kernels must be explicitly constructed for each programme, which suggests that the programmes would need to be kept short in practice. However, in practice, many fewer than that may be needed in order to construct any given kernel matrix. Instead, we can set up a recursive kernel calculation algorithm which only implements the kernels that are needed.

The mean function $m : \mathcal{X} \to \mathbb{R}$ of f is derived in a similar manner. If the p^{th} component of f is atomic, then its mean function is assumed to be known, whereas if it is derived it is given by transforming the mean function of the processes from which it is derived:

$$m((p,x)) := m_p(x), \text{ where } m_p(x) := \begin{cases} \text{is given} & : f_p \text{ is atomic} \\ \mathcal{A}_p(m_1, ..., m_{p-1}) & : \text{otherwise.} \end{cases} \tag{2.22}$$

The kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is slightly more involved:

$$\kappa((p,x),(q,x')) := \kappa_{pq}(x,x') \text{ where}$$
$$\kappa_{pq}(x,x') := \begin{cases} 0 & : p \neq q \text{ , and both } f_p \text{ and } f_q \text{ are atomic} \\ \kappa_p(x,x') & : p = q \text{ and } f_p \text{ is atomic} \\ 0 & : p > q \text{ and } f_p \text{ is atomic} \\ \mathcal{A}_p(\kappa_{1,q}(\cdot,x'),...,\kappa_{p-1,q}(\cdot,x'))(x) & : p > q \text{ and } f_p \text{ is derived} \\ \kappa_{qp}(x',x) & : p < q \\ \mathbb{E}[f_p(x)f_p(x')] - m_p(x)m_p(x') & : p = q \text{ and } f_p \text{ is derived.} \end{cases}, \tag{2.23}$$

where κ_p is the known covariance function associated with the atomic GP f_p.

Batched Calculation Broadly speaking, the code in `Stheno.jl` just implements the above functions. The only notable deviation is that functions to compute the mean vector at a collection of inputs and the covariance matrix between all pairs of inputs in two collections of inputs are provided, rather than functions to compute the mean at a single point, or covariance at a single point. This is to conform with the `AbstractGPs.jl` internal interface – it requires this for practical performance-related reasons.

Implicit Mean Function and Kernel There are a couple of possible ways in which the above could be implemented in practice. I focus on covariance computation, but similar arguments hold for the mean. One approach is to produce a collection of kernel types which can represent any kernel that a GPPP may require. Another, is to side-step this step, and directly implement a function which computes the covariance between any two collections of points in a GP. The approach taken is of no direct consequence to the user, but substantially reduces the amount of code which must be written in order to achieve the needed functionality.

To see this, consider the bottom half of Fig. 2.9, which considers what is required in each approach to compute the quantities demanded for the process resulting from the summation of two other processes, `f` and `g`. The first example shows how the covariance matrix is computed if the direct approach is adopted. It is necessary only to compute each of the relevant covariance matrices, and sum them. Contrast that with the indirect approach, for which it is necessary to provide a (cross-)kernel data structure capable of representing each of the four kernels needed, and another to represent their summation. More generally, the

```
# Direct approach.
cov(f, x)

# Indirect approach.
k = kernel(f)
kernel_matrix(k, x)

# Addition using direct approach.
cov((::typeof(+), f, g), x) =
    cov(f, x) + cov(g, x) + cov(f, g, x) + cov(g, f, x)

# Addition using indirect approach.
kernel(::typeof(+), f, g) =
    kernel(f) + kernel(g) + kernel(f, g) + kernel(g, f)
```

Fig. 2.9 `Stheno.jl` provides functionality to compute the covariance matrix between two collections of inputs (direct), rather than a standalone data structure that corresponds to the kernel of the GPPP (indirect). An example involving the summation of two processes demonstrates the two approaches. The direct approach yields a simpler implementation than the indirect approach.

indirect approach mandates that a kernel library which is flexible enough to represent the kernel of any GPPP be maintained alongside the GPPP library. Typically this requires one or more (cross-)kernels per affine transformation.

Thus the direct approach achieves a reduction in the amount of code which must be written that is roughly proportional to the size of the affine transformation library. However the main benefit is the removal of an entire layer of unnecessary abstraction, which reduces the complexity of the overall implementation for the implementer of the library.

Initial versions of `Stheno.jl` adopted the indirect approach, but it was dropped in favour of the direct when their relative merits became apparent.

Numerics Clearly, it is straightforward to produce rank-deficient covariance matrices using the constructions discussed, which will lead to problems in the computations required for sampling, inference, and log marginal likelihood computation. However, note also that these same problems arise in standard GP software frameworks, and there exist a variety of practical strategies for addressing them. For example, careful documentation which points out having observations located too close together in a very smooth GP will likely cause numerical problems, and suggesting that a pseudo-point approximation be applied if this

```
@gppp let

    # Shared trend process.
    f_trend = stretch(GP(SEKernel()), θ.λ_trend)

    # Specify model for CO2.
    f_co2_latent = θ.CO2.σ_latent * f_trend
    f_co2_wiggle = θ.CO2.σ_wiggle *
        stretch(GP(SEKernel()), θ.CO2.λ_wiggle)
    f_co2_period = θ.CO2.σ_period *
        GP(SEKernel() ∘ PeriodicTransform(θ.CO2.freq))
    f_co2 = f_co2_latent + f_co2_wiggle + f_co2_period +
        θ.CO2.σm * GP(ConstantKernel())

    # Specify model for temperature.
    f_T_trend = θ.T.σ_trend * f_trend
    f_T_wiggle = θ.T.σ_wiggle * stretch(GP(SEKernel()), θ.T.λ_wiggle)
    f_T = f_T_trend + f_T_wiggle + θ.T.σm * GP(ConstantKernel())
end
```

Fig. 2.10 GPPP to jointly model CO2 concentration and global average temperature.

seems to be a problem in practice. Providing an easy way to perform the standard trick of adding a small constant to the diagonal of any covariance matrix which needs to be inverted or whose Cholesky factorisation needs to be computed is a necessity in practice.

2.5 A Climatological Example

In this example we consider a joint model for atmospheric CO2 concentration and temperature[6].

Fig. 2.10 shows the GPPP used to model these two quantities. A shared slowly-varying `f_trend` is intended to capture the climate-change related signal present in both signals. The other components of the model are specific to each of the signals, and are intended to account for other factors which affect the measured data in different ways.

[6]CO2 measurements are taken over Mauna Loa, Hawaii, and are due to Dr. Pieter Tans, NOAA/ESRL `www.esrl.noaa.gov/gmd/ccgg/trends/` and Dr. Ralph Keeling, Scripps Institution of Oceanography `scrippsco2.ucsd.edu/`. Temperature data is the Met Office Hadley Centre's monthly global sea-surface temperature data set, due to Kennedy et al. (2011), and was retrieved from `https://www.metoffice.gov.uk/hadobs/hadsst3/data/download.html`. Both of these data sets are subject to occasional revision by their maintainers.

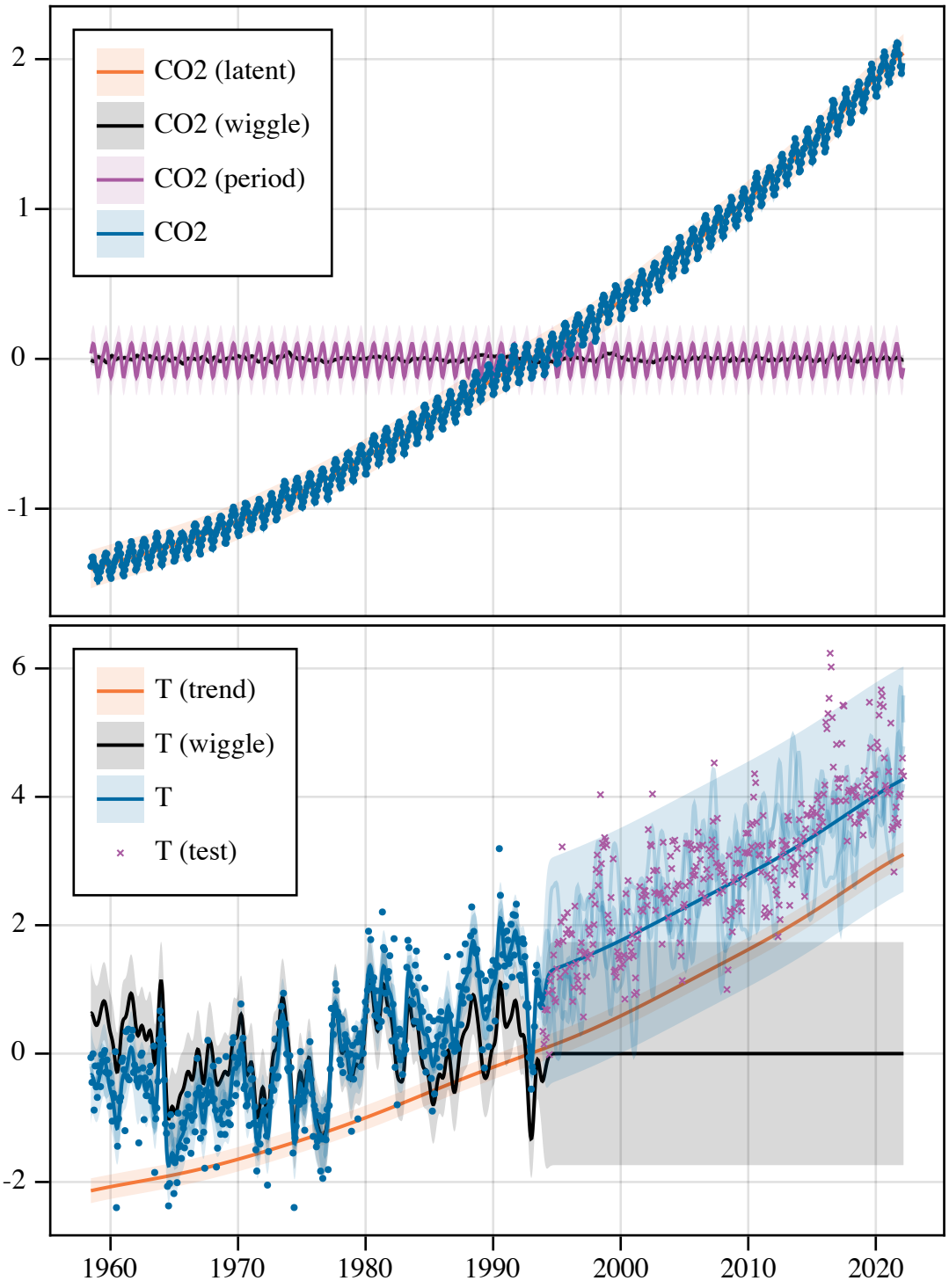


Fig. 2.11 CO2 (top) and temperature (bottom) over time, in addition to the decomposition of the posterior distribution over .

Type-II maximum likelihood was performed in the hyperparameters. The model does not see temperature data after 1995 during training. Instead, predictions are made for it using CO2 data from the entire period, and temperature data up to 1995. This general class of prediction task occurs in practice – the IPCC regularly publish different CO2 scenarios for the coming years. These are not intended to form predictions for how much CO2 will be emitted, but rather to characterise how much CO2 will be emitted under certain kinds of policy decisions.

The model seems to do a surprisingly good job of forecasting temperature given actual CO2 emissions. The marginal predictive variance grows quickly to become quite large, but this is to be expected, because the year-to-year fluctuations in temperature constitute quite a large component of the variability of the signal. The important shared information, however, persists and enables meaningful extrapolation. Visually comparing the prediction to the temperatures which were actually observed between 1995 and 2022 suggests that the extrapolation is reasonable.

2.6 The Interoperability Offered by Abstraction

The single-process perspective enables a GPPP to implement the same interface as any other AbstractGP, so it is possible to utilise all of the functionality designed with that interface in mind, without any modification whatsoever.

2.6.1 Scalability with Pseudo-Point Approximations

As discussed in Chapter 1, pseudo-point approximations form a core part of the modern use of GPs in practice, owing to the improved scalability that they offer. It is natural, therefore, to ask whether pseudo-point approximations can be incorporated into the GPPP abstraction introduced in this chapter. The answer is yes, and it is entirely trivial to do so.

First, some notation. Let f be a GPPP which distributes over functions mapping from $\mathcal{X}$ to $\mathbb{R}$. As before let $\mathcal{X} := \cup_{p \in \{1,\ldots,P\}} \{(p, x) : x \in \mathcal{X}_p\}$. Let $\mathbf{x} \in \mathcal{X}^N$ be a collection of inputs, $\mathbf{y} \in \mathbb{R}^N$ the corresponding outputs, and $\mathbf{z} \in \mathcal{X}^M$ a collection of pseudo-inputs. Crucially, note that the elements of $\mathbf{x}$ can reside in any of the process in f, as can those of $\mathbf{z}$. Denote by $\mathbf{f}$ the vector-valued random variable comprising $(f(\mathbf{x}_1), ..., f(\mathbf{x}_N))$, and by $\mathbf{u}$ the vector-valued random variable comprising $(f(\mathbf{z}_1), ..., f(\mathbf{z}_M))$.

Recall that all which is required to utilise pseudo-point approximations is the computation of a few covariance matrices ($\mathbf{C_f}$, $\mathbf{C_{fu}}$, and $\mathbf{C_u}$) and mean vectors ($\mathbf{m_f}$ and $\mathbf{m_u}$). This chapter

has made it clear how these quantities can be computed, so a generic implementation of a pseudo-point approximation can immediately be applied to a GPPP.

Below I present a few worked examples designed to illustrate the flexibility that the above offers in some more detail.

The Trivial Case To recover a standard pseudo-point approximation, suppose that we have a trivial GPPP comprising only a single process ($P = 1$), then $\mathbf{x}$ be of the form $\mathbf{x} = ((1, x_1), (1, x_2), ..., (1, x_N))$ where $x_n \in \mathcal{X}_1$, and $\mathbf{z} = ((1, z_1), (1, z_2), ..., (1, z_M))$ where $z_m \in \mathcal{X}_1$. In this case, the elements of $\mathbf{x}$ and $\mathbf{z}$ live in the same component process of f, which is what is typically done in practice.

A Sum of Processes However, we can easily go further. If $P > 1$, there is no particular reason to place all of the pseudo-inputs in only a single component of the GPPP f. Fig. 2.12 depicts a three-process GPPP, in which the third process is the sum of the first two. In this example, observations have been made of the first and third component processes of f. This means that, letting N_1 and N_3 be the total number of observations made of the first and third processes respectively, $\mathbf{x}$ is of the form

$$\mathbf{x} = ((1, x_1), ..., (1, x_{N_1}), (3, x_{N_1+1}), ..., (3, x_{N_1+N_3})),$$

where $x_n, \in [-5, 5]$ for all $n \in \{1, ..., N\}$.

We have complete freedom over the choice of location of pseudo-observations – we can place them in any of the available processes. In this example I chose to put them in the first and second processes. This means that, letting M_1 and M_2 be the number of pseudo-inputs located in the first and second processes respectively, $\mathbf{z}$ is of the form

$$\mathbf{z} = ((1, z_1), ..., (1, z_{M_1}), (2, z_{M_1+1}), ..., (2, z_{M_1+M_2}).$$

where $z_m \in [-5, 5]$ for all $m \in \{1, ..., M\}$.

Inspection of Fig. 2.12 reveals that this choice of pseudo-inputs provides a reasonable approximation to the exact posterior distribution. Note that had the pseudo-inputs only been placed in one of the component processes, the approximation quality would have been quite poor. For example, if all of the pseudo-inputs resided in the third component process of f, the approximation would necessarily have large uncertainty over the first and second component processes. This would be okay if the observations had only been made of the third process,

but because there are also observations available for the first process, the exact posterior has quite low uncertainty as to the value of the first and second processes.

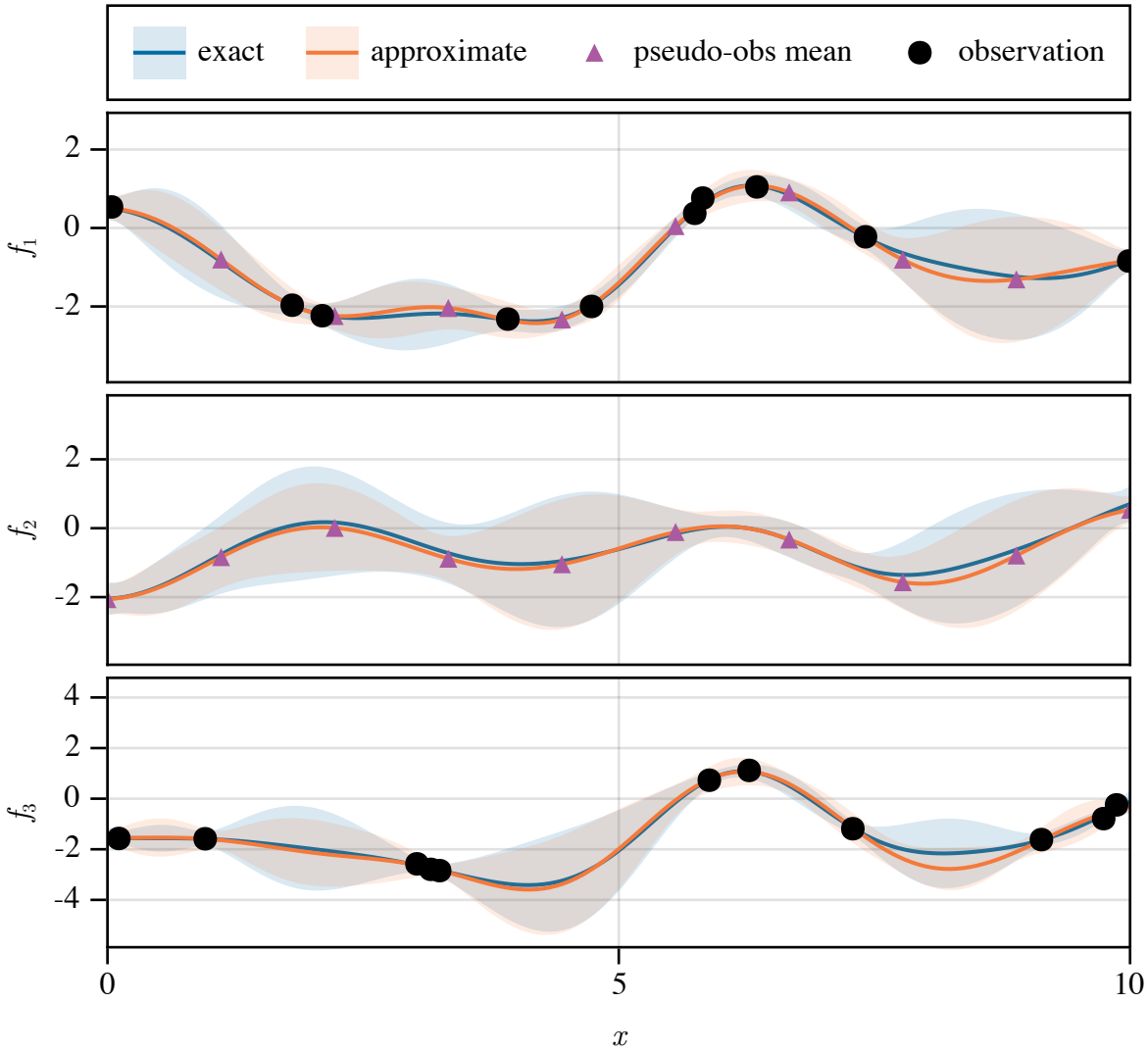


Fig. 2.12 Exact inference vs pseudo-point approximation to the posterior over a GPPP comprising the sum of two GPs, $f_3 := f_1 + f + 2$. A handful of observations are made (black dots) of f_1 and f_3. Pseudo-points (purple triangles) are placed in f_1 and f_2. Exact posterior marginals (mean ± 3 standard deviations) are shown in blue, approximate posterior marginals in orange.

Inter-Domain Pseudo-Point Approximations In the last example, pseudo-inputs were only placed in component processes which were already part of the model. The inter-domain approximations introduced by Lazaro-Gredilla and Figueiras-Vidal (2009), however, place

```
f = @gppp let
    f₁ = GP(randn(), SEKernel())
    f₂ = GP(SEKernel())
    f₃ = f₁ + f₂
end

# Specify inputs in f₁ and f₃.
x₁ = GPPPInput(:f₁, rand(10) * 10)
x₃ = GPPPInput(:f₃, rand(11) * 10)
x = vcat(x₁, x₃)

# Specify pseudo-inputs in f₁ and f₂.
z_locations = range(0, 10; length=10)
z₁ = GPPPInput(:f₁, z_locations)
z₂ = GPPPInput(:f₂, z_locations)
z = vcat(z₁, z₂)

# Construct the approximate posterior.
f_post_approx = posterior(VFE(f(z)), f(x, 1e-2), y)
```

Fig. 2.13 Listing for Fig. 2.12.

pseudo-inputs in component processes which are constructed through integral transforms of existing processes.

Matthews et al. (2016) formalise this idea in the context of variational inference by letting $\mathbf{u} := \mathcal{L}f$, where $\mathcal{L} : (\mathcal{X} \to \mathbb{R}) \to \mathbb{R}^M$ is a bounded integral transform. They show that, as before, $q(\mathbf{u})$ can be chosen arbitrarily provided that $q(f \mid \mathbf{u}) = p(f \mid \mathbf{u})$, and that the KL divergence between approximate posterior and exact posterior over f is minimised by optimising the bound introduced in Chapter 1. This can be achieved with standard pseudo-point approximations in a GPPP by adding $(\mathcal{L}f)$ be the $(P)^{th}$ process in an existing GPPP f comprising $P-1$ processes, and letting $\mathbf{z} := [(P,1), ..., (P,M)]$, thus provided that the integral transformations needed for inter-domain transformations are available in the GPPP framework, inter-domain approximations can be readily employed.

The GPPP abstraction separates the two components of an inter-domain pseudo-point approximation: the specification of a process which is an integral transformation of existing processes, and the placement of pseudo-points within that process. For example, once the `convolve` transformation discussed in Sec. 2.3 has been implemented, it can be used either as a model component of which actual observations are made, or as simply an additional component of the programme in which pseudo-points can be placed. The point is that once an integral transformation has been specified and implemented within the GPPP abstraction,

it can immediately be used as either an important part of the model or as somewhere to locate pseudo-points.

Other Examples in the Literature There is a variety of other examples in the literature: van der Wilk et al. (2017), Hensman et al. (2017), Adam (2017), and Wilson et al. (2016) all place pseudo-points in processes other than the one in which they are interested in doing inference. This works shows that the GPPP provides a unifying abstraction, and a practical framework, from which they can all be reached.

Pseudo-Points for Additive GPs

The rest of this section considers the class of additive processes introduced by Duvenaud et al. (2011) as they are especially easy to analyse from our perspective and, as we shall see, the placement of pseudo-points turns out to be quite important for the design of efficient approximate inference schemes. Fig. 2.14 presents a very similar situation to that discussed by Adam (2017). We have placed pseudo-points in f_1 and f_2 to induce an approximation to the posterior over f_3, and see that we quite accurately recover the posterior marginals over each latent process f_1 and f_2, which is in agreement with their results. That such a good approximation can be achieved with comparatively few pseudo-data is possible due to the additive structure of the model: the function f_3 on $\mathbb{R}^2$ is degenerate in the sense that it can be fully described in terms of two functions on $\mathbb{R}$. However, note that the placement of the pseudo-points in f_1 and f_2 rather than f_3 is not of crucial importance for the quality of the approximation. For example, we could have placed $M_l = 25$ pseudo-points regularly between $(0, -10)$ and $(0, 10)$ and another $M_l = 25$ regularly between $(-10, 0)$ and $(10, 0)$, for a total of $M = 50$ pseudo-points, and arrived at the same results up to rounding errors. Moreover we need not tile the axes, but could simply space the points such that they are parallel to the axes and find little difference in the quality of the results obtained; the point is that there exist many possible places that one could locate the pseudo-inputs and achieve the same results, thus the *statistical* efficiency which enables such high quality approximate inference is not inherently tied to the process in which the pseudo-points live.

In light of these observations one might reasonably question whether or not there is any benefit at all to placing pseudo-points in a particular component process of this GPPP as opposed to some other. The answer is yes, but the benefit is found in computational efficiencies rather than statistical ones. Consider the largest matrices that must be computed to perform approximate inference related tasks: $\mathbf{C_u}$, $\mathbf{C_{fu}}$, and the diagonal of $\mathbf{C_{ff}}$. $\mathbf{C_{ff}}$ is invariant to changes in the pseudo-inputs, but the same is not true for $\mathbf{C_u}$ and $\mathbf{C_{fu}}$.

To understand this, consider the computation of $\mathbf{C_{fu}}$ in more depth. In particular consider the general case of a GPPP comprising D one-dimensional processes $f_1, ..., f_D$, where f_d has kernel κ_d and an associated vector of M_l pseudo-points $\mathbf{u}^{(d)}$ with corresponding pseudo-inputs $\mathbf{z}^{(d)} \in \mathbb{R}^{M_l}$, for a total of $M := M_l D$ pseudo-points. The marginal process f, whose sample paths map $\mathbb{R}^D \to \mathbb{R}$, is

$$f(x) := \sum_{d=1}^{D} f_d(x_d)\,. \tag{2.24}$$

The cross-covariance matrix is

$$\mathbf{C_{fu}} = \begin{bmatrix} \mathbf{C_{fu^{(1)}}} & \dots & \mathbf{C_{fu^{(D)}}} \end{bmatrix}$$

where

$$[\mathbf{C_{fu^{(d)}}}]_{nm} = \kappa_d\big(x_n, z_m^{(d)}\big)\,, \quad m \in \{1, ..., M_l\}.$$

The total number of operations required to compute $\mathbf{C_{fu}}$ in this situation is $\mathcal{O}(NM_lD)$.

Contrast this with what happens if all M pseudo-inputs are located in f, and each pseudo-input is therefore an element of $\mathbb{R}^D$. In this case, each element of $\mathbf{C_{fu}}$ requires a sum over all D dimensions:

$$[\mathbf{C_{fu}}]_{nm} = \sum_{d=1}^{D} \kappa_d(x_n, z_{md})\,, \quad m \in \{1, ..., M\}.$$

Consequently, the total number of operations required is $\mathcal{O}(NMD) = \mathcal{O}(NM_lD^2)$, a factor of D greater than the previous situation.

While the construction of these matrices is dominated asymptotically by the $\mathcal{O}(M^2N) = \mathcal{O}(M_l^2D^2N)$ matrix multiplication $\mathbf{C_{uf}C_{fu}}$ which is required when computing the evidence lower bound, in the finite-data regime it is by no means a given that the time and memory required to compute these matrices is negligible. Fig. 2.15 shows that, at least when considering small problems, the growth in the time taken to compute the matrices is as expected and that they comprise a non-negligible amount of the computation undertaken. It is particularly interesting to note that, at least at this small scale, the multiplication $\mathbf{C_{uf}C_{fu}}$ comprises a relatively small amount of the work required to compute the evidence lower bound. This is not attributable to improved parallelism in this multiplication routine as all computations were limited to use a single thread. Instead, it is a reflection of the high-quality implementation of the level-3 BLAS operation invoked to compute the product. The analysis presented in Fig. 2.15 does not constitute an exhaustive study of the performance of our implementation of a GPPP, but it does approximately show the expected properties, and

demonstrate the computational benefits of having the ability to easily place pseudo-inputs in any component of a given GP model.

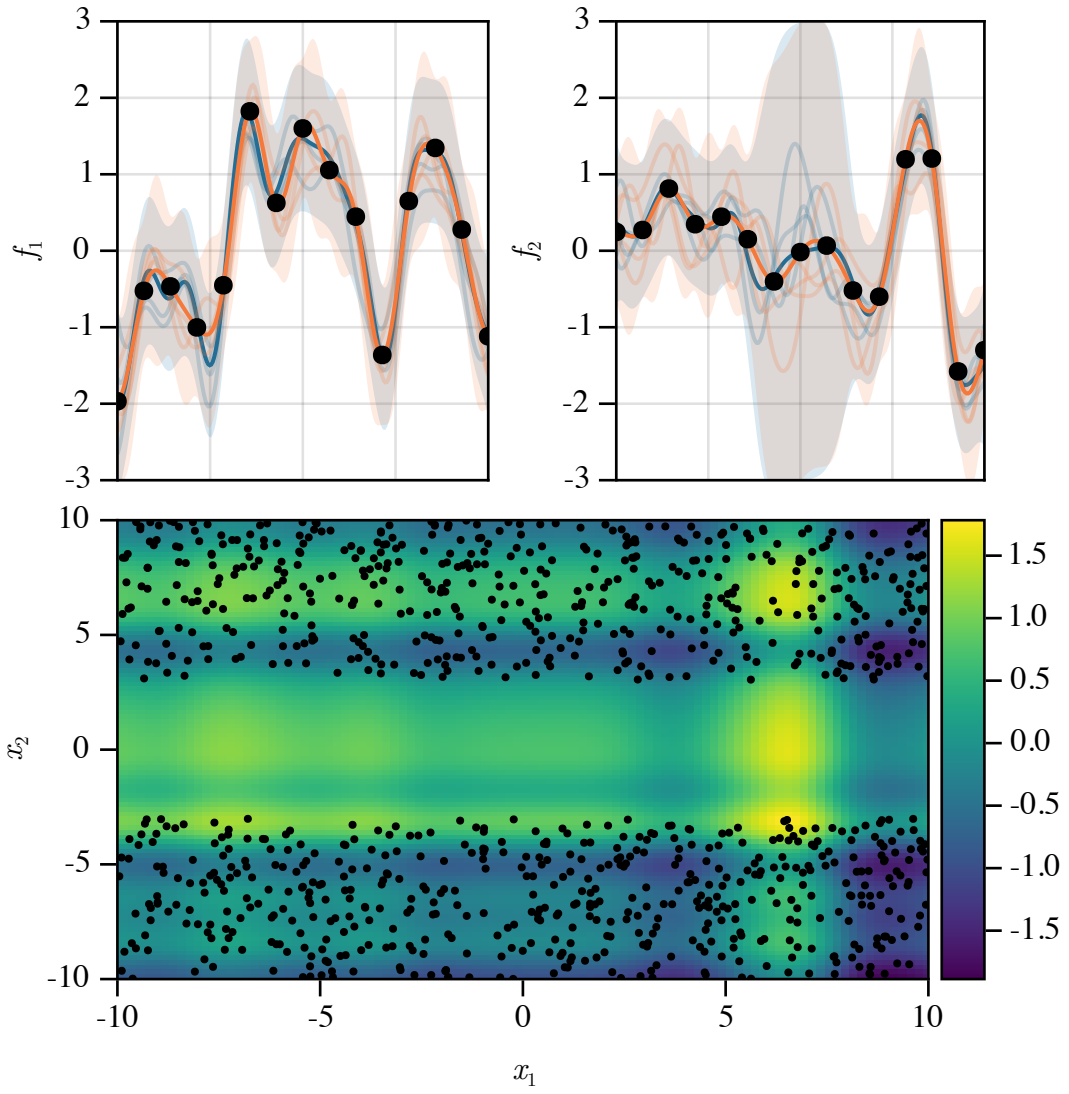


Fig. 2.14 Approximate inference in the GP $f_3(x_1, x_2) := \frac{1}{2}(f_1(x_1) + f_2(x_2))$ which is the direct sum of f_1 and f_2. $N = 1000$ observations are made of f_3 (small black dots), and $M = 50$ pseudo-points are used (large black dots). $M_l = 25$ of these are spaced regularly over the domain of f_1, the other $M_l = 25$ over the domain of f_2. Exact and approximate posterior quantities are indicated in blue and orange respectively. Exact posterior mean is indicated by background colour in heatmap.

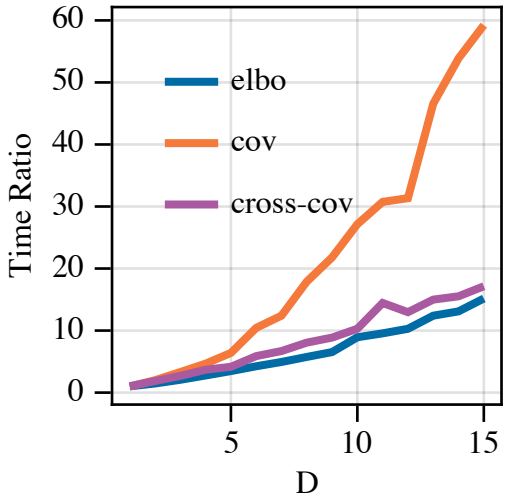


Fig. 2.15 Ratio of time taken to compute $\mathbf{C_{uu}}$, $\mathbf{C_{uf}}$, and the ELBO when pseudo-inputs are located in f and $f_1, ..., f_D$, as D is varied between 1 and 15, with $N = 1000$ and $M_l = 10$ fixed. Standardised EQ kernel is used for all processes. For example, a ratio of 10 indicates that a computation takes 10 times longer when the pseudo-inputs are located in f than in $f_{1:D}$.

2.6.2 Non-Gaussian Observation Models

Just as pseudo-point approximations designed without GPPPs in mind can be utilised to perform approximate inference in GPPPs, so can approximations designed to perform approximate inference in GPs with non-Gaussian observation models. For example, let $\text{Exp}(\lambda)$ denote the exponential distribution with rate λ, and consider the simple two-level hierarchical model

$$\begin{aligned} f &\sim \mathcal{GP}(m, \kappa)\,, \\ y_n \mid f(x_n) &\sim \text{Exp}(\lambda_n), \quad \lambda_n := e^{f(x_n)}, \quad n \in \{1, ..., N\}. \end{aligned}$$

`ApproximateGPs.jl` and `ConjugateComputationVI.jl` contain code to perform approximate inference and learning in the above model using the Laplace approximation (see e.g. Rasmussen and Williams (2006)) and Conjugate Computation VI (Khan and Lin, 2017) respectively.[7] These packages operate at a sufficiently high level of abstraction that

[7] Code available at `https://github.com/JuliaGaussianProcesses/ApproximateGPs.jl/` and `https://github.com/willtebbutt/ConjugateComputationVI.jl`.

they are useful for both GP research and use in practice, but are lower-level than a fit-predict interface.

There is nothing preventing us from replacing f in the above with a GPPP, since they satisfy the same interface. In this case, `Stheno.jl` handles all of the jointly-Gaussian components of the model, while entirely separate code handles the non-Gaussian components. For example, Fig. 2.16 shows the results of replacing it with the GPPP specified in Fig. 2.13, and performing inference and learning in the above model using these approximations when observations are made of both f_2 and f_3. In this case, both approximations appear to have produced very similar approximate posteriors.

There are any number of variations on the above obtained by replacing the Exponential distribution observation model with something else. For example a Bernoulli to perform binary classification, a Negative Binomial to model count data, or an observation model which is a function of multiple locations in f. All of these can elegantly utilise GPPPs by using the single-process perspective.

The point of this example is not that the approximations demonstrated are known to be especially well-suited to an Exponential observation model, nor is it that this particular model is necessarily very interesting in and of itself – they were chosen arbitrarily from those presently available within the JuliaGPs ecosystem, and the model is chosen to be representative of a broad class of important models, so as to illustrate the following: neither of the implementations of the approximations needs to know about the existence of GPPPs – they are designed to work with *any* GP which implements the `AbstractGPs.jl` interface, and the single-process perspective makes it possible to treat a GPPP as a single GP, so interoperability is seamless. Indeed, one can quite easily verify this fact by checking that neither `ConjugateComputationVI.jl` nor `ApproximateGPs.jl` depend on `Stheno.jl`, either directly or indirectly.

2.7 Related Work

As alluded to earlier in this chapter, there are numerous excellent and highly-successful kernel-centric software packages for working with GPs. Probably the most well-known of these is GPML (Rasmussen and Nickisch, 2010). Similar libraries include GPy (GPy, 2012), GPflow (Matthews et al., 2017), GaussianProcesses.jl (Fairbrother et al., 2021), and GPyTorch (Gardner et al., 2018b). Being kernel-centric, they all have the limitations discussed at the start of the this chapter.

```
f = @gppp let
    f1 = f1_σ * stretch(GP(SEKernel()), f1_λ)
    f2 = f2_σ * stretch(GP(SEKernel()), f2_λ)
    f3 = f1 + f2
end
```

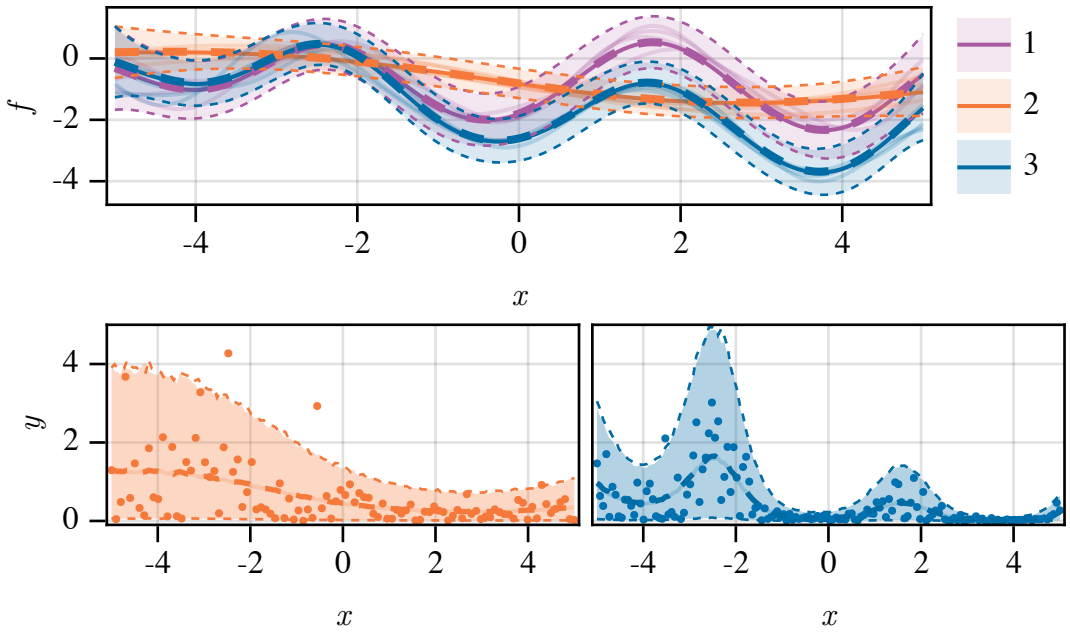


Fig. 2.16 Approximate posterior obtained using the Laplace and CVI approximations. Top: GPPP in which approximate inference and learning are performed. Middle: approximate posterior over each of the latent processes. Bottom: observations of f_2 (left) and f_3 (right), and approximate posterior over location-dependent rate. Dashed lines show the approximate posterior obtained using CVI, while the solid lines and filled regions the approximate posterior obtained using the Laplace approximation.

However, in addition to its relationship with existing GP software, the present work fits squarely within the field of *Probabilistic Programming* – indeed it is from this field that the GPPP derives its name. Over the last decade or so there has been a surge of interest in Probabilistic Programming, with 2018 seeing the first conference dedicated to the subject, although work in the area goes back to at least the early 1990s. Its goals concern all aspects of the creation of software frameworks and programming languages that enable one to write programmes that specify a probabilistic model, and approximate inference therein.

Broadly speaking, they attempt to exploit one of the core appeals of probabilistic modelling: the clean divide which exists between modelling and inference. Once a model has been fully specified, *what* needs to be computed in order to perform inference is fully specified.

The advantages of achieving this separation are substantial, and impact several aspects of the modelling process. Firstly, the individual already well-versed in probabilistic modelling and approximate inference would experience a significant increase in their productivity when designing a new model, as they could spend their time developing and refining a model without concern for the technical details of approximate inference. Conversely it has the potential to both increase the impact of advances in approximate inference techniques, while simultaneously reducing the time taken for widespread adoption; once an approximate inference technique has been implemented within a probabilistic programming framework, it is available to be used by all of those who use the framework. Similarly significant would be the effect for those whose primary expertise is in some domain other than probabilistic modelling and inference, for whom the implementation of a probabilistic model and accompanying approximate inference would be impossible without either a very significant amount of effort on their part to understand the relevant material, or collaboration with those who already have the appropriate expertise. They would instead just have to acquire an understanding of how to the specify an appropriate model for their problem, and then allow the framework to handle inference for them.

Of course, the problem is that *how* it is best computed (approximately) is situation-specific, each different approximation technique possesses different failure modes and demands different quantities be derived from the model the user specifies in order for it to operate. Moreover, we do not currently have good ways to automatically choose between different approximate inference techniques for a given model. Different probabilistic programming frameworks take different approaches to handling this problem.

In practice, the extent to which inference can be automated depends strongly on the types of problems that one considers. If a framework can specify *any* arbitrary programme that generates data, it will admit arbitrarily hard inference problems that even an expert would

not be able to hand-craft a solution for. Languages such as Church (Goodman et al., 2012), Anglican (Tolpin et al., 2015), Turing.jl (Ge et al., 2018), Gen.jl (Cusumano-Towner et al., 2019), and Soss.jl (Scherrer and Zhao, 2020) fall into this category. These frameworks must therefore offer the user a variety of options for performing approximate inference, and ought to come with some kind of health warning for users. For example, Turing.jl offers the user a range of high-level choices as to which approximate inference algorithm should be deployed and tools to diagnose whether they have produced accurate results. However, it shields them from the technical issues associated with their implementation for a particular model – from any given model it algorithmically derives a procedure to perform ancestral sampling in it, and attempts to derive a function to compute the log joint probability density and its gradient.

As more constraints are placed on the types of programmes that a framework can express inference becomes more straightforward. For example Stan (Carpenter et al., 2017) requires that programmes be specified such that their log joint probability density is differentiable w.r.t. the random variables on which inference is to be performed. This means that highly efficient Hamiltonian Monte Carlo (see Neal et al. (2011) and Betancourt (2017) for reviews), along with its various extensions / refinements (Girolami and Calderhead, 2011; Hoffman and Gelman, 2014), and gradient-based black-box variational inference techniques (Kucukelbir et al., 2015) built on the so-called *reparameterisation trick* (Kingma and Welling, 2013; Rezende et al., 2014; Titsias and Lázaro-Gredilla, 2014) can always be employed. Earlier work on probabilistic programming includes the Bayesian inference Using Gibbs Sampling (BUGS) (Lunn et al., 2009; Thomas et al., 1992) and Just Another Gibbs Sampler (JAGS) (Plummer et al., 2003) projects, which constrain programmes to those for which Gibbs samplers can be constructed. Of course, it is still possible to produce programmes within these frameworks in which approximate inference is highly inaccurate, or takes a very long time to run.

The GPPP introduced here is an example of a family of highly-constrained probabilistic programmes. This permits highly accurate inference, which can be more automatic than that offered by other GPPPs. Most probabilistic programmes assume that they specify the entire model under consideration, however, it is exceedingly rare for a model to contain only Gaussian random variables in practice. At the very least, hyperparameters will need to be inferred. So the role of the GPPP is somewhat different from a typical probabilistic programme: it handles the tractable Gaussian components of a model and the relationships between them, and marginalises away infinite-dimensional quantities. In this sense, the GPPP and more general probabilistic programmes are complementary to one another – one could embed a GPPP inside a larger probabilistic programme.

2.7.1 Multi-Output GPs

One could express a lot of the models discussed above using multi-output GP frameworks. In particular, consider the climatological example in Sec. 2.5. This example could also be implemented as a linear mixing model with two outputs and 6 latent processes. Specifically, a collection of independent latent processes:

$$\begin{bmatrix} f_{\text{trend}} & f_{\text{CO2-wiggle}} & f_{\text{CO2-period}} & GP(0, \kappa_{\text{const}}) & f_{\text{T-wiggle}} & GP(0, \kappa_{\text{const}}) \end{bmatrix}, \tag{2.25}$$

and a mixing matrix

$$\begin{bmatrix} \sigma_{\text{CO2-latent}} & 1 & 1 & 1 & 0 & 0 \\ \sigma_{\text{T-latent}} & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.26}$$

A key problem here is interpretability. One has to compare the mixing matrix and the latent processes in order to understand how the observables are derived, and how to interpret them. Contrast this with Fig. 2.10, in which each process is named, and it is quite clear how each of the processes are to be combined and interpreted.

This kind of additive model is just about all that can be expressed with this particular multi-output GP though, and while multi-output GPs have been devised which mix processes using more flexible integral linear transformations (for example see Alvarez and Lawrence (2008)), the above issue surrounding interpretability persists. Moreover, while some authors note that each output of a multi-output GP need not have the same domain (Alvarez et al., 2011), in practice I am not aware of any multi-output GP tool which actually achieves this or is designed with it in mind.

I speculate, however, that it may be the case that a sufficiently flexible multi-output GP implementation would provide a good *compilation target* for a GPPP. That is, if one could automatically translate a GPPP into a flat multi-output GP structure, it might be possible to exploit a multi-output GP implementation to provide a highly-optimised approach to inference. I have not pursued work in this direction, but believe it would be interesting to do so.

2.7.2 Revisiting Kernels

One could straightforwardly utilise a GPPP to derive a kernel, and plug that into a standard kernel-centric framework for working with GPs. Indeed, this is how I originally approached the first iteration of this work.

However, doing so produces two inconveniences. The first is that deriving a kernel using this framework does not also derive the mean function. Both would need to be plugged into a GP object, and failure to do so could yield inconsistent behaviour. The second is simply that it produces extra code and layers of complexity that are unnecessary – rather than having a single GPPP, you would have a GP containing a kernel which is defined by a GPPP. Such additional complexity only has utility if an existing GP framework forces the user to provide a kernel, thus making the production of a kernel a necessity. If a framework instead operates at the level of GPs, abstracting away their implementation details, a kernel-based approach provides only the disadvantage of added complexity – this is the situation in the Julia Gaussian Processes ecosystem, owing to the `AbstractGPs.jl` interfaces.

2.8 Conclusion

In this chapter, the need for the GPPP has been provided, its details explained, and its utility demonstrated. It has been shown to provide an approach to working with GPs which is more closely tied to the model users actually wish to express, interoperate seamlessly with existing approaches to approximate inference, and to enable simple models to be expressed by users in a simple manner. A general approach to its practical implementation has been provided, which should be transferable to many different programming languages, in addition to a focus on one particular implementation in the Julia programming language, `Stheno.jl`.

On-Going Work While the abstractions introduced in this chapter appear to work well, there remains both algorithmic and practical work to be done. On a practical note, there are plenty of affine transformation which have yet to be implemented and included within the framework. For example, only rudimentary integral and differential transformations are currently available. In the case of differential transformations, this is simply because the tooling for algorithmic differentiation in Julia has yet to reach a stage of maturity where it is simple to implement much more, and will be resolved when a greater level of maturity is reached – for example, a framework implemented using Jax (Bradbury et al., 2018) would not have this problem as it is able to handle nested differentiation easily. Integral transformations are somewhat trickier – closed-form solutions are only known in a limited number of cases, and approximations are infeasible in more than a few dimensions. As with differential transformations, this is entirely orthogonal to the abstractions introduced in this chapter: integrals which were hard to compute in a kernel-centric framework remain hard in a GPPP.

Furthermore, there are numerous GPs for which efficient inference algorithms exist that circumvent the computation of the covariance matrix entirely, such as GPs with Markovian

structure (which are discussed at length in the next chapter) and finite-rank GPs such as Bayesian linear regression models. No attention has been paid in this chapter to such GPs, but it would be valuable to determine how best to include them within the framework. Similarly, the pathwise sampling techniques introduced by Wilson et al. (2021) ought to be simple to extend to GPPPs.

The Krylov subspace methods discussed by Gibbs and MacKay (1997) and Gardner et al. (2018b) just rely on having access to the covariance matrix. There is not currently code in the Julia GPs ecosystem for working with these approaches to approximate inference, but it would be trivial to utilise them with a GPPP provided that they were implemented in terms of one of the `AbstractGPs.jl` interfaces.

Possible algorithmic improvements include the caching of intermediate computations when computing covariance matrices for multiple components of a GPPP at the same time. This is likely to be very important in multi-output GPs, where the same covariance matrices are recycled extensively.

Chapter 3

Combining Pseudo-Point and State Space Approximations for Sums of Separable Gaussian Processes

3.1 Introduction

Large spatio-temporal data containing millions or billions of observations arise in various domains, in particular in climate science. While GP models can be useful models in these settings, the computational expense of exact inference is typically prohibitive, necessitating approximation. Pseudo-point approximations alone are typically insufficient due to their limitations in the context of time-series problems, a discussed in Sec. 1.3.5.

To address this limitation, this chapter combines pseudo-point approximations with state-space (Särkkä and Solin, 2019; Särkkä et al., 2013) approximations, whose strengths are almost entirely complementary. Fig. 3.1 shows a single time-slice of a spatio-temporal model for daily maximum temperature, which extrapolates from fixed weather stations, constructed using this technique.

This work hinges on a conditional independence property possessed by separable GPs. This property was identified by O'Hagan (1998), and appears to have gone largely unnoticed within the GP community. In conjunction with the imposition of some structure on the pseudo-point locations, this property yields a collection of methods for approximate inference algorithm which scale linearly in time, the same as standard pseudo-point methods in space, and which can be implemented straightforwardly by utilising standard Kalman filtering-like algorithms.

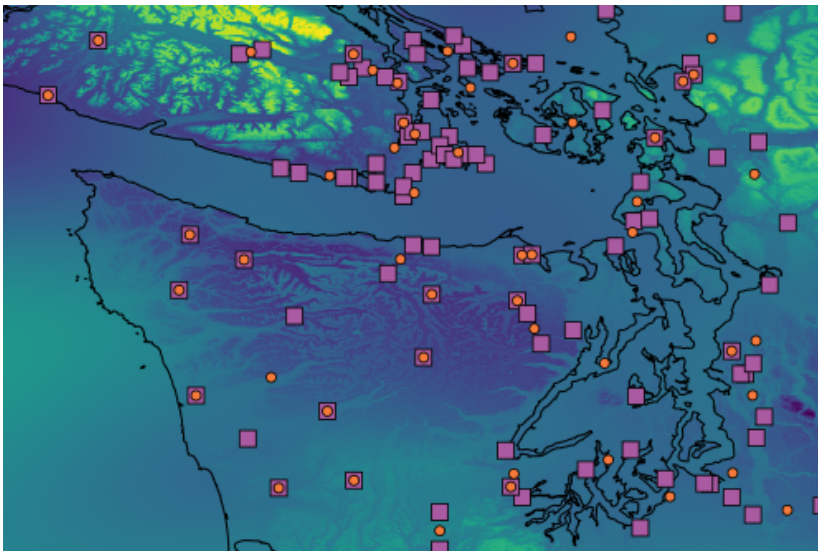


Fig. 3.1 Spatial slice of a large-scale spatio-temporal modelling problem: The posterior mean belief over max temperature (standardised scale, -3 3) on a day in early 2020 around Seattle and Vancouver. Pink squares are weather stations, orange dots are pseudo-points.

In particular, it is shown

- how O'Hagan's conditional independence property can be utilised to significantly accelerate the variational inference scheme of Titsias (2009) for GPs with separable and sum-separable kernels *without* the need for any further approximation,
- how this can be straightforwardly combined with the Markov property utilised by state space approximations (Särkkä and Solin, 2019) to obtain an accurate approximate inference algorithm for sum-separable spatio-temporal GPs that scales linearly in time, and
- how the earlier work of Hartikainen et al. (2011) on this topic, who consider a very similar setting, is more closely related to the pseudo-point work of Csató and Opper (2002) and Snelson and Ghahramani (2005) than previously realised.

3.2 Sum-Separable Spatio-Temporal GPs

A GP is *separable across space and time* if its kernel is of the form

$$\kappa((\mathbf{r},\tau),(\mathbf{r}',\tau')) = \kappa^{\mathbf{r}}(\mathbf{r},\mathbf{r}')\,\kappa^{\tau}(\tau,\tau') \tag{3.1}$$

where $\mathbf{r},\mathbf{r}' \in \mathcal{X}$ are spatial inputs and $\tau,\tau' \in \mathbb{R}$ are temporal inputs. Kernels such as κ are also referred to as being separable. There is no particular restriction on what $\mathcal{X}$ is defined to be – it could be 3-dimensional Euclidean space in the literal sense, or it could be something else, such as a graph or the surface of a sphere. Moreover, no restrictions are placed on the form of $\kappa^{\mathbf{r}}$, in particular it need not be separable. Similarly, while the temporal inputs must be in $\mathbb{R}$, it is irrelevant whether this dimension actually corresponds to time or to something else entirely.

This work considers a generalisation of separable GPs that I call *sum-separable across space and time*, or simply *sum-separable*. A GP is *sum-separable* if it can be sampled by summing samples from a collection of independent separable GPs. Specifically, let $f_p \sim \mathcal{GP}(0,\kappa_p)$, $p = \{1,...,P\}$, be a collection of P independent separable GPs with kernels κ_p, and $f := \sum_{p=1}^{P} f_p$, then f is sum-separable. f has kernel

$$\kappa((\mathbf{r},\tau),(\mathbf{r}',\tau')) = \sum\nolimits_{p=1}^{P} \kappa_p((\mathbf{r},\tau),(\mathbf{r}',\tau'))\,, \tag{3.2}$$

which is *not* separable, meaning that sum-separable GPs such as f are not generally separable. In fact they are a much more expressive family of models, as they can represent processes which vary on multiple length scales in space and time. Note that these are also distinct from additive GPs (Duvenaud et al., 2011) since each function depends on both space and time.

3.3 State Space Approximations to Sum-Separable Spatio-Temporal GPs

Many time-series GPs can be augmented with additional latent dimensions in such a way that the marginal distribution over the original process is unchanged, but with the highly beneficial property that conditioning on all D dimensions at any point in time renders past and future time points independent (Särkkä and Solin, 2019). This augmentation is exact for many GPs, in particular the popular half-integer Matérn family, and a good approximation for others, such as those with exponentiated-quadratic kernels. Consequently, for any collection of T points in time, $\tau_1 < \tau_2 < ... < \tau_T$, the augmented GP forms a D-dimensional Gauss-Markov

chain, whose transition dynamics are a function of the kernel of the GP. This means that standard algorithms (similar to Kalman filtering) can be utilised to perform inference under Gaussian likelihoods, thus achieving linear scaling in T.

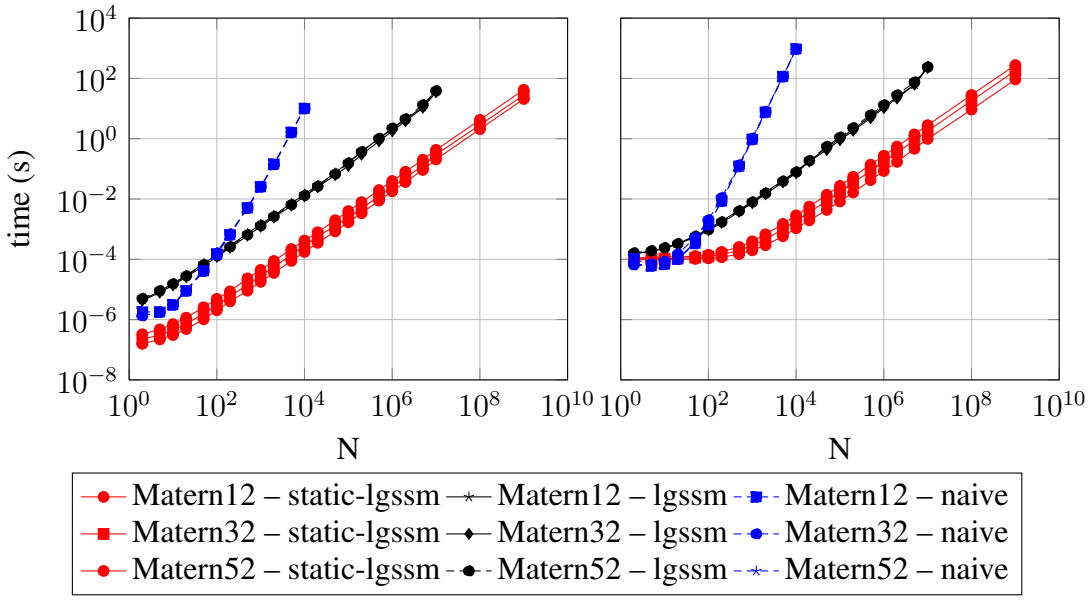


Fig. 3.2 The total time to compute the log marginal likelihood (left) and its gradient (right) of a GP with N observations for various inference methods, all of which are exact.

Performance in Practice It is helpful to have a rough sense of how well this can work in practice. Fig. 3.2 depicts the time performance of state-space methods for three half-integer Matérn kernels, in comparison to the usual procedure of constructing a dense covariance matrix and computing its Cholesky factorisation and so forth. There are two implementations of the state-space approximation labelled *lggsm* and *lggsm-static*. All computations performed using the former utilise standard heap allocated arrays, while the latter uses stack-allocated arrays that are optimised for operations on small arrays.[1] Both scale linearly in N, but the latter is faster in absolute terms in this case. Overall, the precise numbers are less important than a few qualitative observations. Firstly, the naïve approach to computing the log marginal likelihood scales as expected, and is outperformed by both implementations of the state-space method for $N > 100$. Secondly, the heap-allocated implementation is easily able to handle 10^7 observations, and the stack-allocated 10^9. The upshot is that this approach to inference works really very well in some cases, allowing one to scale to large data sets.

These techniques can be extended to separable and sum-separable spatio-temporal GPs for rectilinear grids of inputs, the details of which are as follows.

[1] `https://github.com/JuliaArrays/StaticArrays.jl`

Separable GPs Let $\bar{f}$ be an augmentation of f such that the distribution over $\bar{f}(\tau, \mathbf{r}, 1)$ is approximately equal to that of $f(\tau, \mathbf{r})$, and conditioning on all latent dimensions renders $\bar{f}$ Markov in τ. $\bar{f}$ is specified implicitly through a linear stochastic differential equation, meaning that inference under Gaussian observations can be performed efficiently via filtering / smoothing in a Linear-Gaussian State Space Model (LGSSM). Let $\bar{\mathbf{f}}_t$ be the collection of random variables in $\bar{f}$ at inputs given by the Cartesian product between the singleton $\{t\}$, N_T arbitrary locations in space $\mathbf{r}_{1:N_T}$, and all of the latent dimensions $\{1, \ldots, D\}$. Let the kernel of f be separable: $\kappa((\mathbf{r}, \tau), (\mathbf{r}', \tau')) = \kappa^{\mathbf{r}}(\mathbf{r}, \mathbf{r}')\, \kappa^{\tau}(\tau, \tau')$. Any collection of finite dimensional marginals $\bar{\mathbf{f}} := \bar{\mathbf{f}}_{1:T}$, each using the same $\mathbf{r}_{1:N_T}$, form an LGSSM with $N_T D$-dimensional state, and dynamics

$$\bar{\mathbf{f}}_t \mid \bar{\mathbf{f}}_{t-1} \sim \mathcal{N}\big([\mathbf{I}_{N_T} \otimes \mathbf{A}_t]\, \bar{\mathbf{f}}_{t-1}, \mathbf{C}^{\mathbf{r}}_{\mathbf{f}} \otimes \mathbf{Q}_t\big) \tag{3.3}$$

$$\mathbf{H}_{ab} := \mathbf{I}_a \otimes \begin{bmatrix} 1 & \mathbf{0}_{1\times b-1} \end{bmatrix} \tag{3.4}$$

$$\mathbf{f}_t = \mathbf{H}_{N_T D}\, \bar{\mathbf{f}}_t, \tag{3.5}$$

$$\mathbf{y}_t \mid \mathbf{f}_t \sim \mathcal{N}(\mathbf{f}_t, \mathbf{S}_t) \tag{3.6}$$

where $\otimes$ denotes the Kronecker product, $\mathbf{A}_t \in \mathbb{R}^{D\times D}$ and $\mathbf{Q}_t \in \mathbb{R}^{D\times D}$ are functions of κ^{τ}, $\mathbf{Q}_t$ is positive definite, $\mathbf{C}^{\mathbf{r}}_{\mathbf{f}}$ is the covariance matrix associated with $\kappa^{\mathbf{r}}$ and $\mathbf{r}_{1:N_T}$, $\mathbf{0}_{p\times q}$ is a $p \times q$ matrix of zeros, $\mathbf{y}_t$ is the block of $\mathbf{y}$ containing the observations at the t^{th} time, and the diagonal matrix $\mathbf{S}_t$ is the on-diagonal block of $\mathbf{S}$ corresponding to $\mathbf{y}_t$. See Solin (2016) for further details regarding $\mathbf{A}_t$ and $\mathbf{Q}_t$.

Sum-Separable GPs Let f be the sum-separable GP given by summing over $f_p \sim \mathcal{GP}(0, \kappa_p)$. A state space approximation to f is obtained by constructing a D_p-dimensional state space approximation for each f_p, the finite dimensional marginals of which form an LGSSM

$$\bar{\mathbf{f}}^p_t \mid \bar{\mathbf{f}}^p_{t-1} \sim \mathcal{N}\big([\mathbf{I}_{N_T} \otimes \mathbf{A}^p_t]\, \bar{\mathbf{f}}^p_{t-1}, [\mathbf{C}^{\mathbf{r},p}_{\mathbf{f}} \otimes \mathbf{Q}^p_t]\big) \tag{3.7}$$

$$\mathbf{f}_t = \sum_{p=1}^{P} \mathbf{H}_{N_T D_p} \bar{\mathbf{f}}^p_t \tag{3.8}$$

where $\mathbf{A}^p_t$, $\mathbf{Q}^p_t$, and $\mathbf{C}^{\mathbf{r},p}_{\mathbf{f}}$ are defined in the same way as above for each f_p, and $\mathbf{y}_t \mid \mathbf{f}_t$ is again given by Eq. (3.6). This LGSSM has $N_T \sum_{p=1}^{P} D_p$ latent dimensions, increasing the time and memory needed to perform inference when compared to a separable model, and is the price of a more flexible model.

Benefits and Limitations While this formulation truly scales linearly in T it has two clear limitations, *(i)* all locations of observations must lie on a rectilinear time-space grid if any computational gains are to be achieved; and *(ii)* inference scales cubically in N_T, meaning that inference is rendered infeasible by time or memory constraints if a large number of spatial locations are observed.

3.4 Conditional Independence Results

The key idea in this chapter will be to locate pseudo-points on a rectilinear grid, freeing up observations to lie anywhere in space. This section establishes the key conditional independence properties associated with this arrangement of pseudo-points and observations.

First, the key pre-existing conditional independence for separable GPs (Sec. 3.4.1) is introduced and explained. It is then built upon and utilised in a couple of ways, which enable the introduction of the approximation in the next section. The main conditional independence result is extended in Sec. 3.4.2 from specifying the conditional independence structure between individual points, to the structure between certain sets of points. Next, Sec. 3.4.3 shows that a state-space approximation to a separable GP will always retain separability structure which exists in the original un-approximated prior, meaning that such approximations can be constructed without risking losing this property. Sec. 3.4.4 pin-points the conditional independence structure found between pseudo-points and observations of the above form in separable GPs, and Sec. 3.4.5 extends it to sum-separable GPs.

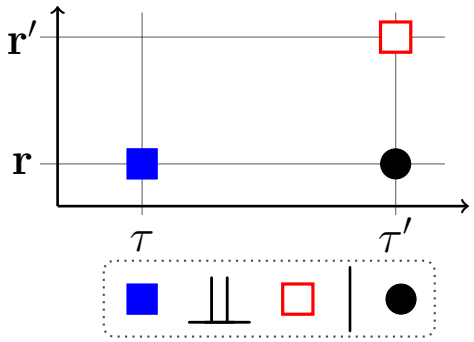


Fig. 3.3 Depiction of the conditional independence property in Eq. (3.9). The blue square is $f(\mathbf{r}, \tau)$, the red square is $f(\mathbf{r}', \tau')$, and the black circle is $f(\mathbf{r}, \tau')$.

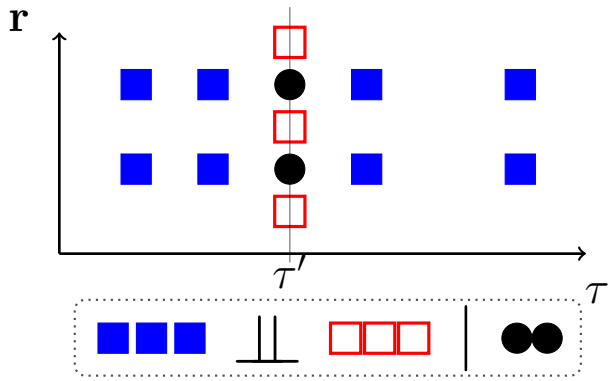


Fig. 3.4 Depiction of the conditional independence property in Eq. (3.11). The blue squares are $f(\mathcal{R}, \mathcal{T})$, the red squares are $f(\mathcal{R}', \mathcal{T}')$, and the black circles are $f(\mathcal{R}, \mathcal{T}')$.

3.4.1 The Conditional Independence Structure of Separable GPs

O'Hagan (1998) showed that a separable GP $f(\mathbf{r}, \tau)$ has the following conditional independence properties:

$$f(\mathbf{r}, \tau) \perp\!\!\!\perp f(\mathbf{r}', \tau') \mid f(\mathbf{r}, \tau')\,, \tag{3.9}$$

$$f(\mathbf{r}, \tau) \perp\!\!\!\perp f(\mathbf{r}', \tau') \mid f(\mathbf{r}', \tau)\,. \tag{3.10}$$

These are explained graphically in Fig. 3.3. It is straightforward to show that this property extends to collections of random variables in f:

$$\begin{aligned} &f(\mathcal{R}, \mathcal{T}) \perp\!\!\!\perp f(\mathcal{R}', \mathcal{T}') \mid f(\mathcal{R}, \mathcal{T}') \quad \text{where} \\ &\quad f(\mathcal{R}, \mathcal{T}) := \{f(\mathbf{r}, \tau) \mid \mathbf{r} \in \mathcal{R}, \tau \in \mathcal{T}\} \\ &\quad f(\mathcal{R}', \mathcal{T}') := \{f(\mathbf{r}, \tau') \mid \mathbf{r} \in \mathcal{R}'\} \\ &\quad f(\mathcal{R}, \mathcal{T}') := \{f(\mathbf{r}, \tau') \mid \mathbf{r} \in \mathcal{R}\} \end{aligned} \tag{3.11}$$

where $\mathcal{R}$ and $\mathcal{R}'$ are sets of points in space, $\mathcal{T}$ is a set of points through time, and $\tau' \in \mathcal{T}$. This conditional independence property is depicted in Fig. 3.4, and it is this second property that sits at the core of the approximation introduced in the next section. In this section this extension is derived and explained, and the manner in which it can be applied to the state-space approximation $\bar{f}$ demonstrated.

3.4.2 Extending The Conditional Independence Result

The following lemma establishes an analogue of the conditional independence result introduced by O'Hagan (1998), which applies to individual points, to collections of points in a separable Gaussian process.

Lemma 3.4.1. *Let $\mathcal{X}$ and $\mathcal{Y}$ be sets, $f \sim \mathcal{GP}(0, \kappa)$ where $\kappa((x,y),(x',y')) := \kappa_x(x,x')\,\kappa_y(y,y')$, $x, x' \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$, and κ_x and κ_y are non-degenerate, meaning covariance matrices constructed using them are invertible. Then for finite sets $\mathcal{X}_1, \mathcal{X}_2 \subset \mathcal{X}$, $\mathcal{Y}_1, \mathcal{Y}_2 \subset \mathcal{Y}$, and sets of random variables*

$$
\begin{aligned}
f(\mathcal{X}_1, \mathcal{Y}_1) &= \{f(x,y) \mid x \in \mathcal{X}_1, y \in \mathcal{Y}_1\},\\
f(\mathcal{X}_2, \mathcal{Y}_2) &= \{f(x,y) \mid x \in \mathcal{X}_2, y \in \mathcal{Y}_2\},\\
f(\mathcal{X}_2, \mathcal{Y}_1) &= \{f(x,y) \mid x \in \mathcal{X}_2, y \in \mathcal{Y}_1\},
\end{aligned}
$$

it is the case that

$$
f(\mathcal{X}_1, \mathcal{Y}_1) \perp\!\!\!\perp f(\mathcal{X}_2, \mathcal{Y}_2) \mid f(\mathcal{X}_2, \mathcal{Y}_1)\,. \tag{3.12}
$$

Proof. Since $f(\mathcal{X}_1, \mathcal{Y}_1)$, $f(\mathcal{X}_2, \mathcal{Y}_2)$, and $f(\mathcal{X}_2, \mathcal{Y}_1)$ are jointly Gaussian, it is sufficient to show that the conditional covariance $\operatorname{cov}(f(\mathcal{X}_1, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_2) \mid f(\mathcal{X}_2, \mathcal{Y}_1))$ is always $\mathbf{0}$. Assign an arbitrary ordering to the elements in each of $\mathcal{X}_1, \mathcal{X}_2, \mathcal{Y}_1$, and $\mathcal{Y}_2$, and let $\mathbf{C}_{\mathcal{X}_i\mathcal{X}_j}$ be the covariance matrix obtained by evaluating κ_{x} at each pair of points $\mathcal{X}_i$ and $\mathcal{X}_j$, such that the $(p,q)^{th}$ element of $\mathbf{C}_{\mathcal{X}_i\mathcal{X}_j}$ is κ_{x} evaluated at the p^{th} and q^{th} elements of $\mathcal{X}_i$ and $\mathcal{X}_j$ respectively. Let $\mathbf{C}_{\mathcal{Y}_i,\mathcal{Y}_j}$ be analogously defined for κ_{y} and $\mathcal{Y}_i$, $\mathcal{Y}_j$. Denote the Kronecker product by $\otimes$, and order the elements of $f(\mathcal{X}_1, \mathcal{Y}_1)$, $f(\mathcal{X}_2, \mathcal{Y}_2)$, and $f(\mathcal{X}_2, \mathcal{Y}_1)$ such that

$$
\begin{aligned}
\operatorname{cov}(f(\mathcal{X}_1, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_2)) &= \mathbf{C}_{\mathcal{X}_1\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2},\\
\operatorname{cov}(f(\mathcal{X}_1, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_1)) &= \mathbf{C}_{\mathcal{X}_1\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_1},\\
\operatorname{cov}(f(\mathcal{X}_2, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_2)) &= \mathbf{C}_{\mathcal{X}_2\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2},\\
\operatorname{cov}(f(\mathcal{X}_2, \mathcal{Y}_1)) &= \mathbf{C}_{\mathcal{X}_2\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_1}.
\end{aligned}
$$

κ_{x} and κ_{y} are non-degenerate, so $\mathbf{C}_{\mathcal{X}_2\mathcal{X}_2}$, $\mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_1}$, and $\mathrm{cov}(f(\mathcal{X}_2, \mathcal{Y}_1))$ are invertible, and the conditional covariance is

$$
\begin{aligned}
&\mathrm{cov}(f(\mathcal{X}_1, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_2) \mid f(\mathcal{X}_2, \mathcal{Y}_1)) \\
&\quad = \mathrm{cov}(f(\mathcal{X}_1, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_2)) - \\
&\qquad \mathrm{cov}(f(\mathcal{X}_1, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_1))\,\mathrm{cov}(f(\mathcal{X}_2, \mathcal{Y}_1))^{-1}\,\mathrm{cov}(f(\mathcal{X}_2, \mathcal{Y}_1)\,, f(\mathcal{X}_2, \mathcal{Y}_2)) \\
&\quad = \mathbf{C}_{\mathcal{X}_1\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2} - (\mathbf{C}_{\mathcal{X}_1\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_1})\,(\mathbf{C}_{\mathcal{X}_2\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_1})^{-1}\,(\mathbf{C}_{\mathcal{X}_2\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2}) \\
&\quad = \mathbf{C}_{\mathcal{X}_1\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2} - \left(\mathbf{C}_{\mathcal{X}_1\mathcal{X}_2}\mathbf{C}_{\mathcal{X}_2\mathcal{X}_2}^{-1}\mathbf{C}_{\mathcal{X}_2\mathcal{X}_2}\right)\left(\mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_1}\mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_1}^{-1}\mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2}\right) \\
&\quad = \mathbf{C}_{\mathcal{X}_1\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2} - \mathbf{C}_{\mathcal{X}_1\mathcal{X}_2} \otimes \mathbf{C}_{\mathcal{Y}_1\mathcal{Y}_2} \\
&\quad = \mathbf{0}.
\end{aligned}
$$

□

This result is depicted in Fig. 3.5 – specifically letting $f(\mathcal{X}_1, \mathcal{Y}_1)$ be the red squares, $f(\mathcal{X}_2, \mathcal{Y}_2)$ the blue squares, and $f(\mathcal{X}_2, \mathcal{Y}_1)$ the black dots. $\mathcal{X}_1$ are the x-coordinates of the red squares, $\mathcal{X}_2$ the x-coordinates of the blue squares / black circles, $\mathcal{Y}_1$ the y-coordinates of the red squares / black circles, and $\mathcal{Y}_2$ the y-coordinates of the blue squares.

Lemma 3.4.1 establishes that a GP being separable implies the presented conditional independence between collections of points. O'Hagan (1998) goes further in the single-point case, also showing that the conditional independence property implies separability, hence showing that the separability and conditional independence statements are equivalent. While it seems plausible that such an equivalence could be established for collections of points, such a result is not needed in this work, and is therefore not pursued.

3.4.3 Separability of the State-Space Approximation

Solin (2016) shows (chapter 5) that a separable spatio-temporal GP f, whose time-kernel has Markov structure, can be expressed as another GP $\bar{f}$ with some auxiliary dimensions. The kernel of $\bar{f}$ is not given explicitly – instead it is expressed in terms of an infinite-dimensional Kalman filter. Consequently, it is unclear without further investigation whether or not the kernel over $\bar{f}$ is separable. I show that, grouping together time and the latent dimension, it in fact separates over space and the grouped dimensions.

Let $\tau \in \mathbb{R}$ denote a point in time, and samples from $\bar{f}(\mathbf{r}, \tau, d)$ be the random variable in f associated with the spatial location $\mathbf{r}$, time point τ, and latent dimension d. Denote $\bar{f}_\tau = \bar{f}(\cdot, \tau, \cdot)$, then Solin (2016) shows that

$$
\bar{f}_\tau \sim \mathcal{GP}(0, \kappa_{\mathbf{r}}(\mathbf{r}, \mathbf{r}')\,\alpha_\tau(d, d')) \tag{3.13}
$$

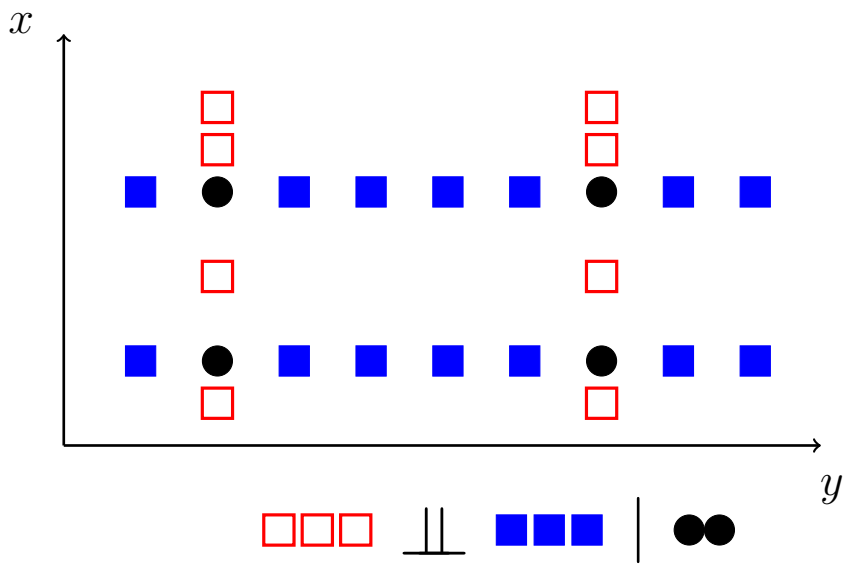


Fig. 3.5 Under a separable GP prior, the random variables at the red squares are conditionally independent of those at the blue squares given those at the black circles.

and, for any τ', kernel $\alpha_\tau : \{1, ..., D\} \times \{1, ..., D\} \to \mathbb{R}$ (isomorphic to a $D \times D$ matrix), kernel over space $\kappa_{\mathbf{r}}$. Let $\mathcal{A}_{\tau\to\tau'} : \mathcal{X} \times \{1, ..., D\} \to \mathcal{X} \times \{1, ..., D\}$ be the linear transition operator,

$$\bar{f}_{\tau'} = \mathcal{A}_{\tau\to\tau'} \bar{f}_\tau + q_{\tau\to\tau'} \tag{3.14}$$

$$(\mathcal{A}_{\tau\to\tau'} \bar{f}_\tau)(\mathbf{r}, d) = \sum_{j=1}^{D} [\mathbf{A}_{\tau\to\tau'}]_{dj} \bar{f}_\tau(\mathbf{r}, j)\,, \tag{3.15}$$

where each $q_{\tau\to\tau'}$ is an independent GP, samples from which are functions $\mathcal{X} \times \{1, ..., D\} \to \mathbb{R}$, and $\mathbf{A}_{\tau\to\tau'}$ is a $D \times D$ matrix of real numbers.

Lemma 3.4.2. *Let $\bar{f} \sim \mathcal{GP}(0, \bar{\kappa})$, the distribution over any time-marginal $\bar{f}_\tau$ be defined according to Eq.* (3.13)*, and the conditional distribution over $\bar{f}_{\tau'}$ given $\bar{f}_\tau$ Eq.* (3.14)*. It follows that $\bar{\kappa}$ is separable, and of the form*

$$\bar{\kappa}((\mathbf{r}, \tau, d), (\mathbf{r}', \tau', d')) = \kappa_{\mathbf{r}}(\mathbf{r}, \mathbf{r}')\, \kappa_{\tau d}((\tau, d), (\tau', d')) \tag{3.16}$$

for some kernel $\kappa_{\tau d}$.

Proof.

$$\begin{aligned}
\bar{\kappa}((\mathbf{r},\tau,d),(\mathbf{r}',\tau',d')) &:= \operatorname{cov}\big(\bar{f}(\mathbf{r},\tau,d)\,,\,\bar{f}(\mathbf{r}',\tau',d')\big) \\
&= \operatorname{cov}\big(\bar{f}_\tau(\mathbf{r},d)\,,\,\bar{f}_{\tau'}(\mathbf{r}',d')\big) \\
&= \mathbb{E}\big[\bar{f}_\tau(\mathbf{r},d)\,\bar{f}_{\tau'}(\mathbf{r}',d')\big] \\
&= \mathbb{E}\big[\bar{f}_\tau(\mathbf{r},d)\,\{(\mathcal{A}_{\tau\to\tau'}\bar{f}_\tau)(\mathbf{r}',d') + q_{\tau\to\tau'}(\mathbf{r}',d')\}\big] \\
&= \mathbb{E}\big[\bar{f}_\tau(\mathbf{r},d)\,\{(\mathcal{A}_{\tau\to\tau'}\bar{f}_\tau)(\mathbf{r}',d')\big]\,.
\end{aligned}$$

where the penultimate equality follows from independence. Applying Eq. (3.15) yields

$$\begin{aligned}
\bar{\kappa}((\mathbf{r},\tau,d),(\mathbf{r}',\tau',d')) &= \sum_{j=1}^{D}[\mathbf{A}_{\tau\to\tau'}]_{d'j}\mathbb{E}\big[\bar{f}_\tau(\mathbf{r},d)\,\bar{f}_\tau(\mathbf{r}',j)\big] \\
&= \sum_{j=1}^{D}[\mathbf{A}_{\tau\to\tau'}]_{d'j}\mathbb{E}\big[\bar{f}(\mathbf{r},\tau,d)\,\bar{f}(\mathbf{r}',\tau,j)\big]
\end{aligned}$$

Applying Eq. (3.13) yields

$$\begin{aligned}
\bar{\kappa}((\mathbf{r},\tau,d),(\mathbf{r}',\tau',d')) &= \sum_{j=1}^{D}[\mathbf{A}_{\tau\to\tau'}]_{d'j}\kappa_{\mathbf{r}}(\mathbf{r},\mathbf{r}')\,\alpha_\tau((d,j) \\
&= \kappa_{\mathbf{r}}(\mathbf{r},\mathbf{r}')\sum_{j=1}^{D}[\mathbf{A}_{\tau\to\tau'}]_{d'j}\alpha_\tau(d,j) \\
&= \kappa_{\mathbf{r}}(\mathbf{r},\mathbf{r}')\,\kappa_{\tau d}((\tau,d),(\tau',d'))
\end{aligned}$$

where

$$\kappa_{\tau d}((\tau,d),(\tau',d')) := \sum_{j=1}^{D}[\mathbf{A}_{\tau\to\tau'}]_{d'j}\alpha_\tau(d,j)\,.$$

□

This separability result is important, because it says that any of the infinite-dimensional state-space approximations to separable spatio-temporal GPs used in this chapter themselves correspond to separable spatio-temporal GPs. It would be a contradiction for this not to be the case when state-space augmentations are exact (such as for GPs with Matern half-integer kernels), but it is not immediately obvious when they are approximations.

3.4.4 Conditional Independence Structure of Observations and Pseudo-Points Under a Separable Prior

Assume that the set of pseudo-inputs $\bar{\mathbf{z}}$ form the rectilinear grid

$$\bar{\mathbf{z}} := \mathcal{Z}_{\mathbf{r}} \times \mathcal{T} \times \{1, ..., D\} \tag{3.17}$$

where $\times$ denotes the Cartesian product, and $\mathcal{Z}_{\mathbf{r}} \subset \mathcal{X}$ and $\mathcal{T} \subset \mathbb{R}$ are the finite sets of spatial and temporal locations at which pseudo-inputs are present, with sizes M_τ and T respectively. Let the pseudo-points be

$$\bar{\mathbf{u}} := \{\bar{f}(\mathbf{r}, \tau, d) \mid (\mathbf{r}, \tau, d) \in \bar{\mathbf{z}}\}. \tag{3.18}$$

Furthermore, let

$$\mathbf{u}_\tau := \{\bar{f}(\mathbf{r}, \tau, 1) \mid \mathbf{r} \in \mathcal{Z}_{\mathbf{r}}\}, \tag{3.19}$$

then $\bar{\mathbf{u}} \setminus \mathbf{u}_\tau$ is the collection of all pseudo-points not in $\mathbf{u}_\tau$.

Let $\mathcal{X}_1, ..., \mathcal{X}_T \subset \mathcal{X}$ be finite sets of of points in space, one for each point in $\mathcal{T}$. The set of points

$$\mathbf{x}_\tau := \{(\mathbf{r}, \tau, 1) \mid \mathbf{r} \in \mathcal{X}_\tau\} \tag{3.20}$$

are the elements of $\bar{f}$ which are observed (noisily) at time τ, so let

$$\mathbf{f}_\tau := \{\bar{f}(\mathbf{r}, \tau, d) \mid (\mathbf{r}, \tau, d) \in \mathbf{x}_\tau\} \tag{3.21}$$

It is now possible to present the key result:

Theorem 3.4.3.

$$\mathbf{f}_\tau \perp\!\!\!\perp \bar{\mathbf{u}} \setminus \mathbf{u}_\tau \mid \mathbf{u}_\tau. \tag{3.22}$$

That is: $\mathbf{f}_\tau$ is conditionally independent of all pseudo-points not in the first latent dimension of $\bar{f}$ at time τ, given all of the pseudo-points in the first latent dimension of $\bar{f}$ at time τ – Fig. 3.6 visualises this property.

Proof. Let $\mathcal{X} := \mathcal{X}$ and $\mathcal{Y} := \mathbb{R} \times \{1, ..., D\}$ – i.e., group together time and latent dimension – and (abusing notation) let

$$\bar{f}(\mathbf{r}, (\tau, d)) := \bar{f}(\mathbf{r}, \tau, d), \quad \mathbf{r} \in \mathcal{X}, \quad (\tau, d) \in \mathcal{Y}. \tag{3.23}$$

By Lemma 3.4.2 the kernel over $\bar{f}$ is separable across $\mathcal{X}$ and $\mathcal{Y}$. Applying Lemma 3.4.1 to $\bar{f}$ and

$$\mathcal{X}_1 := \mathcal{X}_\tau, \quad \mathcal{X}_2 := \mathcal{Z}_{\mathbf{r}}, \quad \mathcal{Y}_1 := \{(\tau, 1)\}, \quad \text{and } \mathcal{Y}_2 := [\mathcal{T} \times \{1, ..., D\}] \setminus \mathcal{Y}_1. \tag{3.24}$$

yields the desired result, as $\bar{f}_{\mathcal{X}_1,\mathcal{Y}_1} = \mathbf{f}_\tau$, $\bar{f}_{\mathcal{X}_2,\mathcal{Y}_2} = \bar{\mathbf{u}} \setminus \mathbf{u}_\tau$, and $\bar{f}_{\mathcal{X}_2,\mathcal{Y}_1} = \mathbf{u}_\tau$. □

This result is depicted in Fig. 3.6. Of the entire 3 dimensional grid of pseudo-points, only those in the first latent dimension at the same time as $\mathbf{f}_\tau$ are needed to achieve conditional independence from all others.

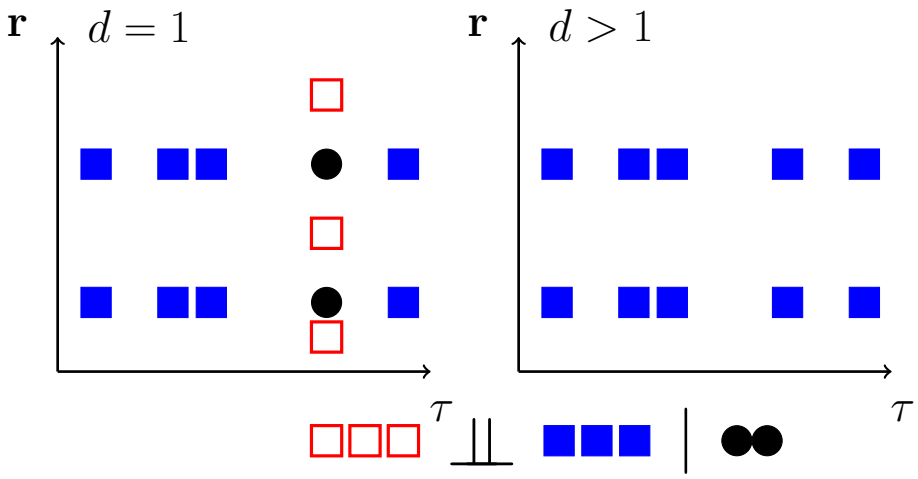


Fig. 3.6 Slices of the 3-dimensional rectilinear grid of pseudo-points / inputs, as well as inputs of observations, depicting the conditional independence structure presented in Theorem 3.4.3. Unfilled red squares correspond to $\mathbf{f}_\tau$, black circles to $\mathbf{u}_\tau$, and filled blue squares to $\bar{\mathbf{u}} \setminus \mathbf{u}_\tau$. The left-hand side corresponds to $d = 1$, while the right-hand side corresponds to $d > 1$. Notice that $\mathbf{u}_\tau$ and $\mathbf{f}_\tau$ only appear in the $d = 1$ slice.

3.4.5 Conditional Independence Structure under a Sum-Separable Prior

Here I extend the above result to sum-separable GPs. Recall that a sum-separable GP $\bar{f}^s$ is defined to be a GP of the form

$$\bar{f}^s := \sum_{p=1}^{P} \bar{f}^p, \quad \bar{f}^p \sim \mathcal{GP}(0, \bar{\kappa}^p), \tag{3.25}$$

where each $\bar{f}^p$ is an independent separable GP. Locate rectilinear grids of pseudo-inputs in each of the separable processes:

$$\bar{\mathbf{z}}^p := \mathcal{Z}_{\mathbf{r}}^p \times \mathcal{T} \times \{1, ..., D\} \tag{3.26}$$

where $\mathcal{Z}_{\mathbf{r}}^p$ are a collection of points in space which are specific to each p. Each of these P grids of points is of the same form as those utilised for separable processes previously. Define sets of pseudo-points for each process:

$$\bar{\mathbf{u}}^p := \{\bar{f}^p(\mathbf{r}, \tau, d) \mid (\mathbf{r}, \tau, d) \in \bar{\mathbf{z}}^p\}, \tag{3.27}$$

$$\mathbf{u}_\tau^p := \{\bar{f}^p(\mathbf{r}, \tau, 1) \mid \mathbf{r} \in \mathcal{Z}_{\mathbf{r}}^p\}, \tag{3.28}$$

$$p \in \{1, ..., P\}, \tag{3.29}$$

and sets containing all of the of pseudo-points through the union of the above process-specific pseudo-points:

$$\bar{\mathbf{u}} := \cup_{p=1}^P \bar{\mathbf{u}}^p, \tag{3.30}$$

$$\mathbf{u}_\tau := \cup_{p=1}^P \mathbf{u}_\tau^p. \tag{3.31}$$

Furthermore, let

$$\mathbf{f}_\tau^p := \{\bar{f}^p(\mathbf{r}, \tau, 1) \mid \mathbf{r} \in \mathcal{X}_\tau\} \tag{3.32}$$

$$\mathbf{f}_\tau^s := \{\bar{f}^s(\mathbf{r}, \tau, 1) \mid \mathbf{r} \in \mathcal{X}_\tau\} \tag{3.33}$$

Theorem 3.4.4 (Conditional Independence in Sum-Separable GPs)**.**

$$\mathbf{f}_\tau^s \perp\!\!\!\perp \bar{\mathbf{u}} \setminus \mathbf{u}_\tau \mid \mathbf{u}_\tau$$

Proof. As in Theorem 3.4.3, it suffices to show that $\text{cov}(\mathbf{f}_\tau^s, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau \mid \mathbf{u}_\tau) = \mathbf{0}$. It is the case that

$$\text{cov}\big(\bar{f}_s(\mathbf{r}, \tau, d), \bar{f}_p(\mathbf{r}', \tau', d')\big) = \text{cov}\big(\bar{f}_p(\mathbf{r}, \tau, d), \bar{f}_p(\mathbf{r}', \tau', d')\big)\,, \tag{3.34}$$

and the covariance between any points in $\bar{f}_p$ and $\bar{f}_{p'}$ is 0 if $p \neq p'$, so

$$\begin{aligned}
\operatorname{cov}(\mathbf{f}_\tau^s, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau) &= \begin{bmatrix} \operatorname{cov}(\mathbf{f}_\tau^1, \bar{\mathbf{u}}^1 \setminus \mathbf{u}_\tau^1) & \dots & \operatorname{cov}\left(\mathbf{f}_\tau^P, \bar{\mathbf{u}}^P \setminus \mathbf{u}_\tau^P\right) \end{bmatrix} \\
\operatorname{cov}(\mathbf{f}_\tau^s, \mathbf{u}_\tau) &= \begin{bmatrix} \operatorname{cov}(\mathbf{f}_\tau^1, \mathbf{u}_\tau^1) & \dots & \operatorname{cov}\left(\mathbf{f}_\tau^P, \mathbf{u}_\tau^P\right) \end{bmatrix} \\
\operatorname{cov}(\mathbf{u}_\tau) &= \begin{bmatrix} \operatorname{cov}(\mathbf{u}_\tau^1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \operatorname{cov}\left(\mathbf{u}_\tau^P\right) \end{bmatrix} \\
\operatorname{cov}(\mathbf{u}_\tau, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau) &= \begin{bmatrix} \operatorname{cov}(\mathbf{u}_\tau^1, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau^1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \operatorname{cov}\left(\mathbf{u}_\tau^P, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau^P\right) \end{bmatrix}.
\end{aligned}$$

Therefore

$$\begin{aligned}
\operatorname{cov}(\mathbf{f}_\tau^s, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau \mid \mathbf{u}_\tau) &= \operatorname{cov}(\mathbf{f}_\tau^s, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau) - \operatorname{cov}(\mathbf{f}_\tau^s, \mathbf{u}_\tau) \left[\operatorname{cov}(\mathbf{u}_\tau)\right]^{-1} \operatorname{cov}(\mathbf{u}_\tau, \bar{\mathbf{u}} \setminus \mathbf{u}_\tau) \\
&= \begin{bmatrix} \operatorname{cov}(\mathbf{f}_\tau^1, \bar{\mathbf{u}}^1 \setminus \mathbf{u}_\tau^1 \mid \mathbf{u}_\tau^1) & \dots & \operatorname{cov}\left(\mathbf{f}_\tau^P, \bar{\mathbf{u}}^P \setminus \mathbf{u}_\tau^P \mid \mathbf{u}_\tau^P\right) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \qquad (3.35)
\end{aligned}$$

where the final equality follows from Theorem 3.4.3. □

3.5 Utilising Separability to Obtain the Best of Both Worlds

We now turn to the main contribution of this chapter: combining the pseudo-point and state space approximations using the various properties established in previous sections of this chapter. The result is an approximation which is applicable to any sum-separable GP whose time kernels can be approximated by a linear SDE. We do this simply by constructing a variational pseudo-point approximation to the state space approximation to the original process. In cases where the state space approximation is exact, this is similar in spirit to constructing an inter-domain pseudo-point approximation (Lazaro-Gredilla and Figueiras-Vidal, 2009) to the original process, where some of the pseudo-points are placed in auxiliary dimensions.

In this section we show that by constraining the pseudo-inputs, approximate inference becomes linear in time.

3.5.1 Combining the Approximations

We now combine the pseudo-point and state space approximations, and show how the temporal conditional independence property means that the optimal approximate posterior is Markov. This in turn leads to a closed-form expression for the optimum under Gaussian observation models and the existence of a simplified LGSSM in which exact inference yields optimal approximate inference in the original model.

Pseudo-Point Approximation of State Space Augmentation We perform approximate inference in a separable GP f with the kernel in Eq. (3.1) by applying the standard variational pseudo-point approximation (Sec. 1.3) to its state space augmentation (Sec. 3.3) $\bar{f}$:

$$q(\bar{f}) := q(\bar{\mathbf{u}})\, p(\bar{f}_{\neq \bar{\mathbf{u}}} \mid \bar{\mathbf{u}}), \quad q(\bar{\mathbf{u}}) = \mathcal{N}(\bar{\mathbf{u}}; \mathbf{m}_{\bar{\mathbf{u}}}^{\mathrm{q}}, \mathbf{C}_{\bar{\mathbf{u}}}^{\mathrm{q}}),$$

where the pseudo-points $\bar{\mathbf{u}} = \bar{\mathbf{u}}_{1:T}$ form a rectilinear grid of points in time, space, and *all* of the latent dimensions with the same structure as $\bar{\mathbf{f}}$ in Sec. 3.3, but replacing $\mathbf{r}_{1:N_T}$ with a collection of M_τ spatial pseudo-inputs, $\mathbf{z}_{1:M_\tau}$, for a total of $TM_\tau D$ pseudo-points. $p(\bar{\mathbf{u}})$ is therefore Markov-through-time with conditional distributions

$$\bar{\mathbf{u}}_t \mid \bar{\mathbf{u}}_{t-1} \sim \mathcal{N}([\mathbf{I}_{M_\tau} \otimes \mathbf{A}_t]\bar{\mathbf{u}}_{t-1}, \mathbf{C}_{\mathbf{u}}^{\mathrm{r}} \otimes \mathbf{Q}_t), \tag{3.36}$$

$$\mathbf{u}_t := \mathbf{H}_{M_\tau D}\bar{\mathbf{u}}_t. \tag{3.37}$$

where $\mathbf{C}_{\mathbf{u}}^{\mathrm{r}}$ is the covariance matrix associated with κ^{r} and $\mathbf{z}_{1:M_\tau}$. Note the resemblance to Eq. (3.3). No constraint is placed on the location of the pseudo-points in space, only that they must remain at the same place for all time points.

Crucially, we now relax the assumption that the inputs associated with $\mathbf{f}$ must form a rectilinear grid. Instead, it is necessary only to require that each observation is made at one of the T times at which we have placed pseudo-points. We denote the number of observations at time t by N_t, and continue to denote by $\mathbf{f}_t$ the set of observations at time t.

Utilising Conditional Independence Due to the extensions to O'Hagan (1998)'s conditional independence property derived in the previous section, and the locations of the pseudo-inputs, $p(\mathbf{f}_t \mid \bar{\mathbf{u}}) = p(\mathbf{f}_t \mid \mathbf{u}_t)$. That is, the conditional distribution over $\mathbf{f}_t$ given all of the pseudo-point depends only on those at time t – moreover, it only depends on the observed component of the pseudo-points at time t, $\mathbf{u}_t$, as opposed to all dimensions, $\bar{\mathbf{u}}_t$. Consequently, the reconstruction term associated with the observations at time t in the ELBO depends only

on $\mathbf{u}_t$, as opposed to the entirety of $\bar{\mathbf{u}}$:

$$\mathcal{L} = \sum_{t=1}^{T} r_t - \mathcal{KL}[q(\bar{\mathbf{u}}) \,\|\, p(\bar{\mathbf{u}})]\,, \tag{3.38}$$

$$r_t := \mathbb{E}_{q(\mathbf{u}_t)}\left[\mathbb{E}_{p(\mathbf{f}_t \,|\, \mathbf{u}_t)}[\log p(\mathbf{y}_t \,|\, \mathbf{f}_t)]\right] \tag{3.39}$$

This property alone can be utilised to yield substantial computational savings – only the covariance between $\mathbf{u}_t$ and $\mathbf{f}_t$ need be computed, as opposed to all of $\bar{\mathbf{u}}$ and $\mathbf{f}_t$. Moreover, in accordance with Sec. B.1.3, this means that

$$\mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}} = \begin{bmatrix} \mathbf{B}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{B}_T \end{bmatrix},\ \mathbf{B}_t := \mathbf{C}_{\mathbf{f}_t\mathbf{u}_t}\Lambda_{\mathbf{u}_t}\mathbf{H}_{M_\tau D}. \tag{3.40}$$

The Optimal Approximate Posterior is Markov As an immediate consequence of Eq. (3.38), and by the same argument as that made by Seeger (1999), highlighted by Opper and Archambeau (2009), the optimal approximate posterior precision satisfies

$$\Lambda^{\mathrm{q}}_{\bar{\mathbf{u}}} = \Lambda_{\bar{\mathbf{u}}} + \begin{bmatrix} \mathbf{G}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{G}_T \end{bmatrix}, \mathbf{G}_t := -2\nabla_{\mathbf{C}^{\mathrm{q}}_t} r_t, \tag{3.41}$$

where $\Lambda^{\mathrm{q}}_{\bar{\mathbf{u}}} := [\mathbf{C}^{\mathrm{q}}_{\bar{\mathbf{u}}}]^{-1}$, and $\mathbf{C}^{\mathrm{q}}_t$ is the t^{th} block on the diagonal of $\mathbf{C}^{\mathrm{q}}_{\bar{\mathbf{u}}}$, and r_t is defined as in Eq. (3.39). Recall that the precision matrix of a Gauss-Markov model is block tridiagonal (see e.g. Grigorievskiy et al. (2017)), so $\Lambda_{\bar{\mathbf{u}}}$ is block tridiagonal. Further, the exact posterior precision of an LGSSM with a Gaussian observation model is given by the sum of this block tridiagonal precision matrix and a block-diagonal matrix with the same block size. $\Lambda^{\mathrm{q}}_{\bar{\mathbf{u}}}$ has precisely this form, so the optimal approximate posterior over $\bar{\mathbf{u}}$ must be a Gauss-Markov chain.

Approximate Inference via Exact Inference in an Approximate Model Provided that each $\mathbf{G}_t$ is positive definite, the above is equivalent to the optimal approximate posterior having density proportional to

$$q(\bar{\mathbf{u}}) \propto \prod_{t=1}^{T} p(\bar{\mathbf{u}}_t \,|\, \bar{\mathbf{u}}_{t-1})\, \mathcal{N}\left(\mathbf{y}^{\mathrm{q}}_t; \bar{\mathbf{u}}_t, \mathbf{G}_t^{-1}\right), \tag{3.42}$$

where $\mathbf{y}_1^{\mathrm{q}}, ..., \mathbf{y}_T^{\mathrm{q}}$ are a collection of T surrogate observations, detailed in Sec. B.1.1. Thus the optimal $q(\bar{\mathbf{u}})$ is produced by performing exact inference in an LGSSM which is closely-related to the original model. Moreover, Ashman et al. (2020) (App. A) show that $\mathbf{G}_t$ can be written as a sum of N_t rank-1 matrices.

$\mathbf{G}_t$ will always be positive definite when $\log p(\mathbf{y}_t \mid \mathbf{f}_t)$ is convex in $\mathbf{f}_t$. To see this, recall that Opper and Archambeau (2009) show that

$$-2\,\nabla_{\mathbf{C}_t^{\mathrm{q}}} r_t = -2\,\mathbb{E}_{q(\mathbf{u}_t)}\left[\nabla^2_{\mathbf{f}_t} \log p(\mathbf{y}_t \mid \mathbf{f}_t)\right] \tag{3.43}$$

provided that $\log p(\mathbf{y}_t \mid \mathbf{f}_t)$ is twice-differentiable in $\mathbf{f}_t$. This Hessian inside the expectation will be negative definite if $\log p(\mathbf{y}_t \mid \mathbf{f}_t)$ is strictly log concave in $\mathbf{f}_t$. Since convex combinations of negative definite matrices are negative definite, of which the expectation under q is an example, the positive definiteness of the optimal $\mathbf{G}_t$ follows. This means that we can safely deploy standard filtering (Kalman, 1960) and smoothing (Rauch et al., 1965) algorithms to perform approximate inference.

On the other hand, it is not possible to say anything in general about whether the optimal choice of $\mathbf{G}_t$ will be positive definite when the observation model density is not log-concave in $\mathbf{f}_t$. This is a practical problem because standard implementations of inference algorithms for LGSSMs assume that $\mathbf{G}_t$ is positive definite. While it might be possible to re-implement such algorithms in a manner that does not require this constraint, it would require additional effort, and may result in a loss of numerical stability.

Solution for Gaussian Observation Models Under a Gaussian observation model, the optimal approximate posterior is given by the exact posterior under the DTC observation model, as discussed in section Sec. 1.3. Eq. (3.40) means that the DTC observation model can be written as

$$\mathcal{N}(\mathbf{y}; \mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\bar{\mathbf{u}}, \mathbf{S}) = \prod_{t=1}^{T} \mathcal{N}(\mathbf{y}_t; \mathbf{B}_t\bar{\mathbf{u}}_t, \mathbf{S}_t)\,, \tag{3.44}$$

where $\mathbf{B}_t$ is given in Eq. (3.40). In conjunction with $p(\bar{\mathbf{u}})$ in Eq. (3.36), this yields the required LGSSM.

This LGSSM can be utilised both to perform approximate inference and compute the saturated bound in linear time, repurposing existing code – see Sec. B.1.2. This LGSSM also makes it clear, for example, how to employ the parallelised inference procedures proposed by Särkkä and García-Fernández (2020) and Loper et al. (2020) within this approximation.

Sum-Separable Models Extending this approximation to sum-separable processes is similar to the standard state space approximation. A rectilinear grid of pseudo-points is placed in each of the P latent processes, for a total of $TM_\tau \sum_{p=1}^{P} D_p$ pseudo-points. The resulting LGSSM is

$$\begin{aligned} \bar{\mathbf{u}}_t^p \mid \bar{\mathbf{u}}_{t-1}^p &\sim \mathcal{N}\left([\mathbf{I}_{M_\tau} \otimes \mathbf{A}_t^p]\, \bar{\mathbf{u}}_{t-1}^p, [\mathbf{C}_{\mathbf{u}}^{\mathrm{r},p} \otimes \mathbf{Q}_t^p]\right) \\ p(\mathbf{y}_t \mid \bar{\mathbf{u}}_t) &= \mathcal{N}(\mathbf{y}_t; \textstyle\sum_{p=1}^{P} \mathbf{B}_t^p \bar{\mathbf{u}}_t^p, \mathbf{S}_t)\,. \\ \mathbf{B}_t^p &:= \mathbf{C}_{\mathbf{f}_t^p \mathbf{u}_t^p} \Lambda_{\mathbf{u}_t^p} \mathbf{H}_{M_\tau D_p}. \end{aligned} \tag{3.45}$$

Note the resemblance to Eq. (3.7).

Efficient Inference in the Conditionals The structure present in each $\mathbf{B}_t^p$ can be used to accelerate inference. In particular note that $\mathbf{H}_{M_\tau D_p}$ has size $M_\tau \times D_p M_\tau$ while $\mathbf{C}_{\mathbf{f}_t \mathbf{u}_t}^p \Lambda_{\mathbf{u}_t}^p$ is $N_t \times M_\tau$. Certainly $M_\tau \leq D_p M_\tau$ and typically $M_\tau < N$, so this linear transformation forms a bottleneck. Sec. B.3 explores this property, and shows how to utilise it to accelerate inference.

Computational Complexity The total number of flops required to compute the saturated ELBO is $T(DM_\tau)^3 + D^3 M_\tau^2 + M_\tau^2 \sum_{t=1}^{T} N_t$ to leading order. This is a great deal fewer when T is large than the $M^3 + M^2 N = M_\tau^3 T^3 + M_\tau^2 T^2 N$ required if the bound is computed naively. Similar improvements are achieved when making posterior predictions.

Utilising Other Pseudo-Point Approximations The conditional independence property utilised to develop the variational approximation in this section also shines new light on the work of Hartikainen et al. (2011). In the specific case of their equation 5, in which the observation model is (adopting their notation) $p(\mathbf{y}_k \mid \mathbf{x}_k) = \mathcal{N}(\mathbf{y}_k; [\mathbf{I}_N \otimes \mathbf{H}]\mathbf{x}_k, \mathbf{S}_t)$, they perform approximate inference in $p(\bar{\mathbf{u}})$ using the modified observation model

$$\begin{aligned} \tilde{p}(\mathbf{y}_t \mid \bar{\mathbf{u}}_t) &:= \mathcal{N}\left(\mathbf{y}_t; \mathbf{C}_{\mathbf{f}_t \bar{\mathbf{u}}_t} \Lambda_{\bar{\mathbf{u}}_t} \bar{\mathbf{u}}_t, [\tilde{\mathbf{C}}_{\mathbf{y}}]_t\right), \\ [\tilde{\mathbf{C}}_{\mathbf{y}}]_t &:= \operatorname{diag}(\mathbf{C}_{\mathbf{f}_t} - \mathbf{C}_{\mathbf{f}_t \bar{\mathbf{u}}_t} \Lambda_{\bar{\mathbf{u}}_t} \mathbf{C}_{\bar{\mathbf{u}}_t \mathbf{f}_t}) + \mathbf{S}_t \end{aligned}$$

which is inspired by the well-known FITC (Csató and Opper, 2002; Snelson and Ghahramani, 2005) approximation. However, due to O'Hagan (1998)'s conditional independence property,

this is equivalent to

$$\tilde{p}(\mathbf{y} \mid \bar{\mathbf{u}}) := \mathcal{N}\left(\mathbf{y}; \mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\bar{\mathbf{u}}, \tilde{\mathbf{C}}_{\mathbf{y}}\right),$$
$$\tilde{\mathbf{C}}_{\mathbf{y}} := \mathrm{diag}(\mathbf{C}_{\mathbf{f}} - \mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\mathbf{C}_{\bar{\mathbf{u}}\mathbf{f}}) + \mathbf{S}.$$

While Hartikainen et al. (2011) did not actually consider the Gaussian observation model in their work, it is clear from the above that they would have utilised *exactly* the FITC approximation applied to $\bar{f}$ had they done so.

Bui et al. (2017) showed that both FITC and VFE can be viewed as edge cases of the Power EP algorithm introduced by Minka (2004). Consequently the equivalent approximate model generalised both that of FITC and VFE – only $\tilde{\mathbf{C}}_{\mathbf{y}}$ is changed from FITC: let $\alpha \in [0, 1]$, then

$$\tilde{\mathbf{C}}_{\mathbf{y}} := \alpha\,\mathrm{diag}(\mathbf{C}_{\mathbf{f}} - \mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\mathbf{C}_{\bar{\mathbf{u}}\mathbf{f}}) + \mathbf{S}.$$

In short, most standard pseudo-point approximations can be straightforwardly combined with state space approximations for sum-separable spatio-temporal GPs in the manner proposed, due to the conditional independence property.

Relationship with Other Approximation Techniques There are several existing methods that could be used to scale GPs to large spatio-temporal problems beyond those already considered – each method makes different assumptions about the types of problems considered, therefore making different trade-offs relative to ours.

The popular Kronecker-product methods for separable kernels explored by Saatçi (2012) are unable to handle heteroscedastic observation noise or missing data, scale cubically in time, and require observations to lie on a rectilinear grid. Our approach suffers none of these limitations.

Wilson and Nickisch (2015) introduced a pseudo-point approximation they call *Structured Kernel Interpolation* (SKI) which is closely-related to the Kronecker-product methods, but removes many of their constraints. In particular, SKI places pseudo-points on a grid across all input dimensions, and utilises them to construct a sparse approximation to the prior covariance matrix over the data – crucially it is local in the sense that the approximation to the covariance between the pseudo-points and any given point depends only on a handful of pseudo-points. SKI covers the domain in a regular grid of points, which results in exponential growth in the number of pseudo-points as the number of dimensions grows. So, while this approximation scales very well in low-dimensional settings, it does not scale to input domains comprising

more than a few dimensions. Moreover, to utilise this grid structure, separability across all dimensions is required. Gardner et al. (2018a) alleviates this exponential scaling problem, but still require that the kernel be separable across all dimensions if their approximation is to be applied. Our approach does not suffer from this constraint as only the time dimension must be covered by pseudo-points – there are no constraints on their spatial locations. Naturally, that we do not perform similar approximations to SKI across the spatial dimensions means that our method will have the standard set of limitations experienced by all pseudo-point methods as the number of points in space grows. In short, the two classes of method are applicable to different kinds of spatio-temporal problems. They take somewhat orthogonal approaches to approximate inference, so combining them by utilising SKI across the spatial dimensions could offer the benefits of both classes of approximation in situations where SKI is applicable to the spatial component.

Similarly, approximations based on the relationship between GPs and Stochastic Partial Differential Equations (Lindgren et al., 2011; Whittle, 1963) could be combined with this work to improve scaling in space when the spatial kernel is in the Matérn family. In low-dimensional settings other standard inter-domain pseudo-point approximations such as those of Hensman et al. (2017), Burt et al. (2020b), and Dutordoir et al. (2020) could be applied.

3.6 Inference Under Non-Gaussian Observation Models

Thus far, only Gaussian observation models centred on affine transformations of the latent GP have been considered, under which the optimal approximate posterior distribution is Gaussian. Once outside this family of observation models, the optimal $q(\mathbf{u})$ is no longer Gaussian. It is, however, standard to restrict the family of approximations to the posterior to be Gaussian (e.g. Hensman et al. (2015)). As shown in the previous section, the optimal approximate posterior precision is block-tridiagonal regardless of the observation model, from which it follows that the optimal Gaussian approximation must be a Gauss-Markov model when the density of the observation model is log-convex in the latent process.

While in general such a model has a total of $T(DM_\tau + 2(DM_\tau)^2)$ free variational parameters, in the cases considered here the off-diagonal blocks of the precision are the same as in the prior, meaning that there are at most $T(DM_\tau + (DM_\tau)^2)$ free (variational) parameters – this is also clear from Eq. (3.41). While one could directly parametrise the precision, this might be inconvenient from the perspective of numerical stability and implementation (standard filtering / smoothing algorithms do not work directly with the precision). Consequently, it probably makes sense to set up a surrogate model in line with that discussed by Khan and

Lin (2017), Chang et al. (2020), and Ashman et al. (2020) – if the observation model is non-convex in the latent process this may introduce an additional loss of accuracy. Alternatively one could parametrise the filtering distributions directly, from which the posterior marginals could be obtained using standard smoothing algorithms.

3.7 Experiments

We view the proposed approximation to be a useful contribution if it is able to outperform the vanilla state space approximation (Sec. 3.3), which is a strong baseline for the tasks we consider. To that end, we benchmark inference against synthetic data in Sec. 3.7.1, on a large-scale temperature modelling task to which both the vanilla and pseudo-point state space approximations can feasibly be applied (Sec. 3.7.2), and finally to a problem to which it is completely infeasible to apply the vanilla state space approximation (Sec. 3.7.3). We do not compare directly against the vanilla pseudo-point approximations of Titsias (2009) and Hensman et al. (2013). As noted in Sec. 1.3, they are asymptotically no better than exact inference for problems with long time horizons.

3.7.1 Benchmarking

We first conduct two simple proof-of-concept experiments on synthetic data with a separable GP to verify our proposed method. In both experiments we consider quite a large temporal extent, but only moderate spatial, since we expect the proposed method to perform well in such situations – if the spatial extent of a data set is very large relative to the characteristic spatial variation, pseudo-point methods will struggle and, by extension, so will our method. Sec. B.2.1 contains additional details on the setup used, and Sec. B.2.1 contains the same experiments for a sum-separable model.

Arbitrary Spatial Locations Fig. 3.7 (top) shows how inputs were arranged for this experiment; at each time 10 spatial locations were sampled uniformly between 0 and 10, so $N = 10T$. The spatial location of pseudo-inputs are regular between 0 and 10. When using pseudo-points, we are indeed able to achieve substantial performance improvements relative to exact inference by utilising the state space methodology, while retaining a tight bound.

Grid-with-Missings Fig. 3.8 (top) shows how (pseudo) inputs were arranged for this experiment for $M_\tau = 10$; the same 50 spatial locations are considered at each time point, but 5 of the observations are dropped at random, for a total of $N_t = 45$ observations per time

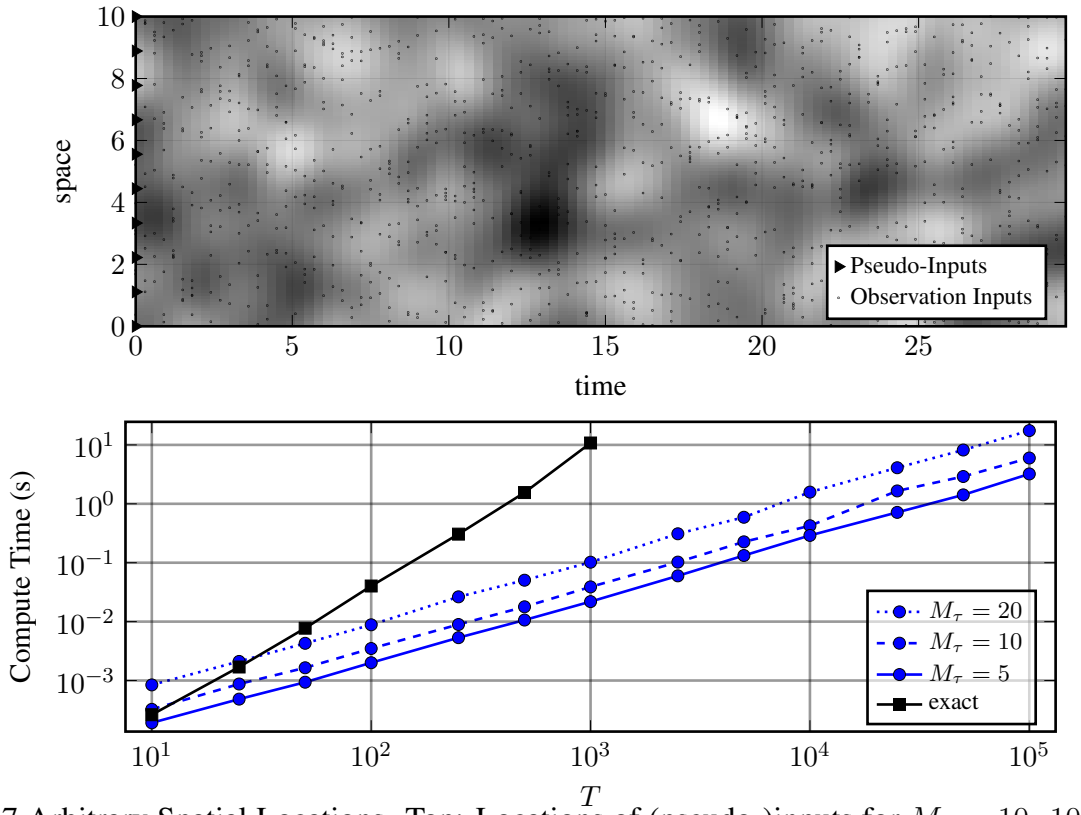


Fig. 3.7 Arbitrary Spatial Locations. Top: Locations of (pseudo-)inputs for $M_\tau = 10$. 10 locations in space chosen randomly at each time point. Bottom: Time to compute ELBO vs performing exact inference. ELBO tight for $M_\tau = 20$; see Fig. B.1.

point – our largest case therefore involves $N = 4.5 \times 10^6$ observations. The timing results show that we are able to compute a good approximation to the LML using roughly a third of the computation required by the standard state space approach to inference.

3.7.2 Climatology Data

The Global Historical Climatology Network (GHCN) (Menne et al., 2012) comprises daily measurements of a variety of meteorological quantities, going back more than 100 years. We combine this data with the NASA Digital Elevation Model (NASA-JPL, 2020) to model the daily maximum temperature in the region $(47°, -127°)$ and $(49°, -122°)$, which contains 99 weather stations. We utilise all data in this region since the year 2000, training on 90% (331522) and testing on 10% (36835) of the data. This experiment was conducted on a workstation with a 3.60 GHz Intel i7-7820X CPU (8 cores), and 46 GB of 3000 MHz DDR3 RAM.

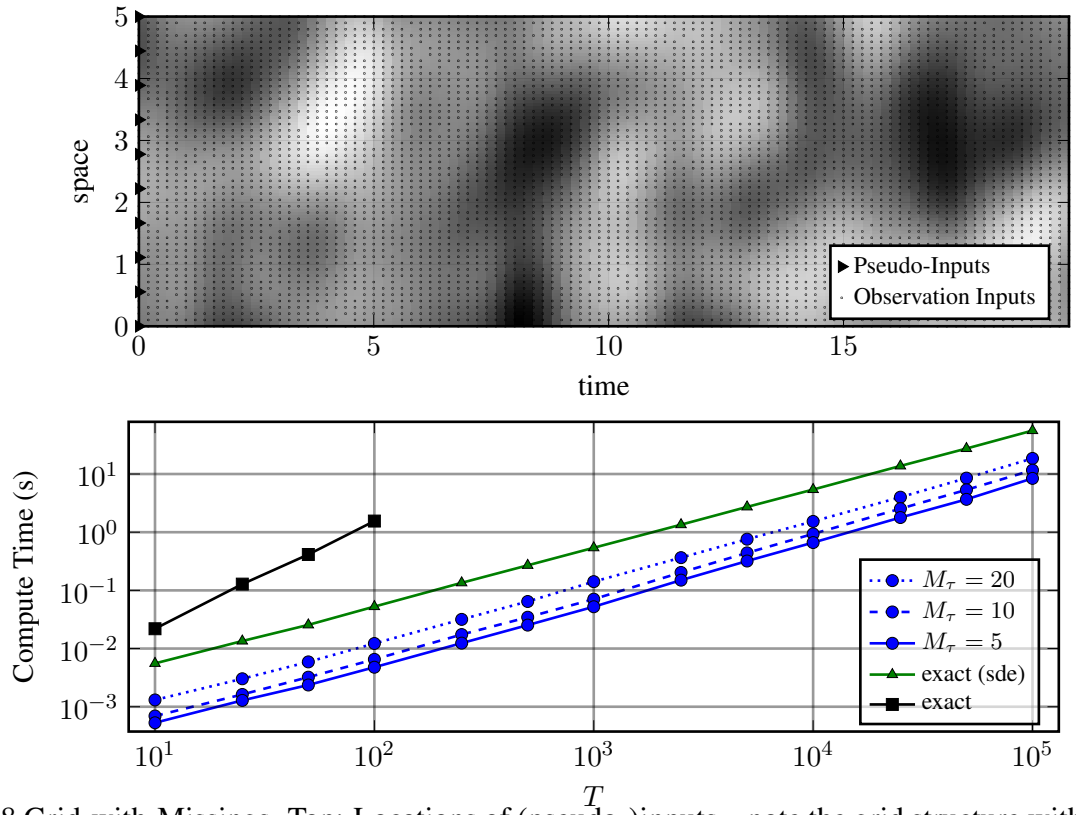


Fig. 3.8 Grid-with-Missings. Top: Locations of (pseudo-)inputs – note the grid structure with 50 observations per time point, of which 5 are missing. Bottom: Time to compute ELBO vs LML naively and via state space methods (*sde*). ELBO tight for $M_\tau = 20$; see Fig. B.1.

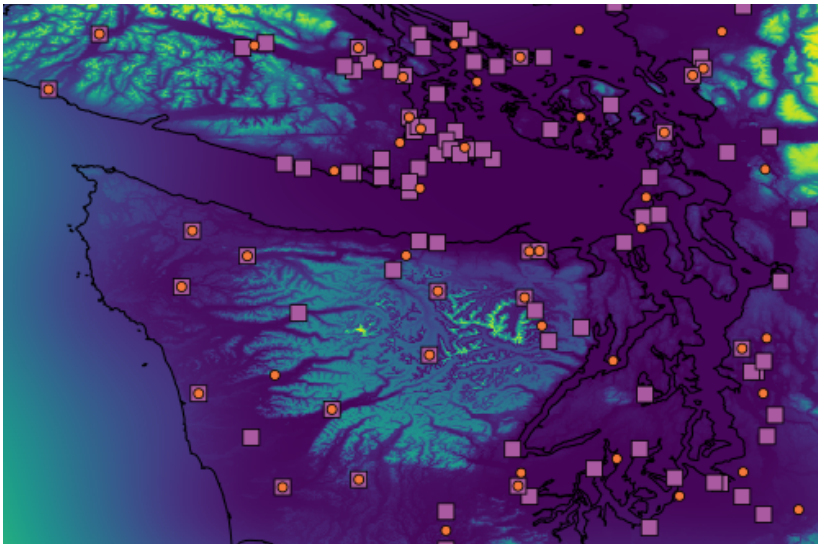


Fig. 3.9 Counterpart to Fig. 3.1 depicting the posterior standard deviation. The colour scale (0 1.75) is relative, pink squares are weather stations, and orange dots pseudo-points.

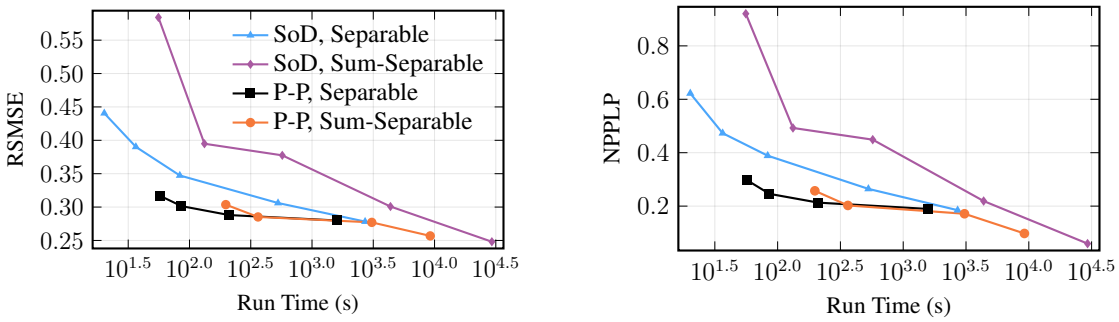
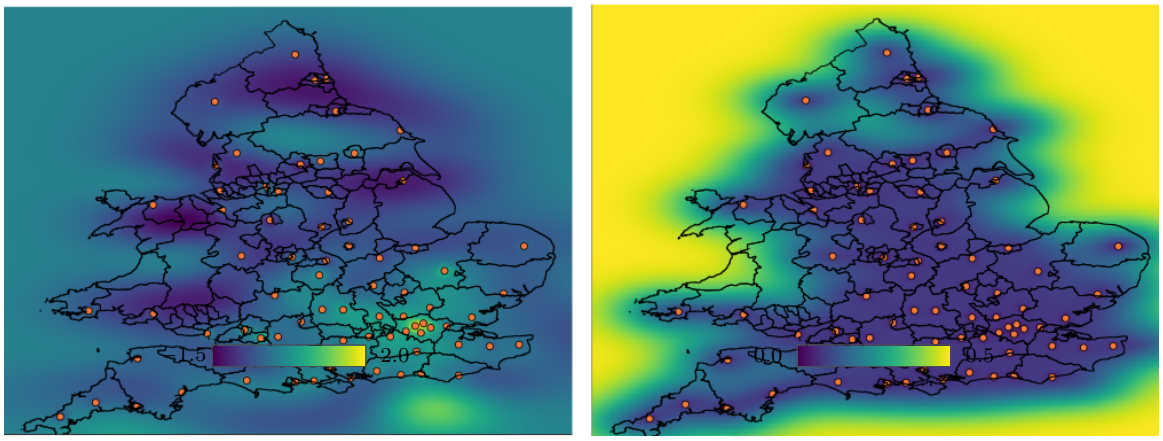


Fig. 3.10 Test Root Standardised Mean-Squared Error (RSMSE) and Negative Posterior Predictive Log Probability (NPPLP). Marked points on Pseudo-Point curves used $M \in \{5, 10, 20, 50\}$ moving from left to right – similarly for SoD markers, with the addition of $M = 99$, corresponding to learning with the exact LML. Larger M improves performance, but time taken to train is increased. Sum-Separable models take longer to train than Separable but can produce better results.



(a) Mean (b) Std. dev.

Fig. 3.11 Apartment price posterior mean and standard deviation on a day near the end of 2020. Pseudo-point locations picked using K-means and marked with orange dots.

Two models were utilised: a simple separable model with a Matérn-$\frac{5}{2}$ kernel over time, and Exponentiated Quadratic over space, and a sum of two such kernels with differing length scales and variances. Additional details in Sec. B.2.2.

Fig. 3.10 compares a simple subset-of-data (SoD) approximation, which is exact when $M = 99$, with the pseudo-point (P-P) approximation developed in this work. The results demonstrate that *(i)* the pseudo-point approximation has a more favourable speed-accuracy trade-off than the SoD, offering near exact inference in less time for a separable kernel, and *(ii)* a sum-separable model offers substantially improved results over a separable in this scenario.

Table 3.1 Performance on apartment price data. $M_\tau = 75$.

	RSMSE	NPPLP
Separable	0.658	2920
Sum-Separable	0.618	192

3.7.3 Apartment Price Data

Property sales data by postcode across England and Wales are provided by HM Land Registry (2014). There are over 10^6 unique postcodes in England and Wales, of which a tiny proportion contain a sale on a given day. Consequently this data set has essentially arbitrary spatial locations at each point in time, which our approximation can handle, but which renders the vanilla state-space method infeasible.

We follow a similar procedure to Hensman et al. (2013), cross-referencing postcodes against a separate database (Camden, 2015) to obtain latitude-longitude coordinates, which we regress against the standardised logarithm of the price. However, we train on 843766 of the 1687536 apartment sales between 2010 and 2020, and test on the remainder. We again consider a separable and sum-separable GP that are similar to those in Sec. 3.7.2, but the temporal kernel is Matérn-$\frac{3}{2}$. More detail in Sec. B.2.3.

Table 3.1 again demonstrates that a sum-separable model is able to capture more useful structure in the data than the separable model; Fig. 3.11 shows the variability and uncertainty in the prices on an arbitrarily chosen day.

3.8 Discussion

This work shows that pseudo-point and state space approximations can be directly combined in the same model to effectively perform approximate inference and learning in sum-separable GPs, and ties up loose ends in the theory related to combining these models. This is important in spatio-temporal applications, where the model admits a form of an arbitrary-dimensional (spatial) random field with dynamics over a long temporal horizon. Experiments on synthetic and real-world data show that this approach enables a favourable trade-off between computational complexity and accuracy.

Standard approximations for non-Gaussian observation models, such as those discussed by Wilkinson et al. (2020), Chang et al. (2020), and Ashman et al. (2020), can be applied straightforwardly within our approximation. The results of this chapter therefore represent the simplest point in a range of possible approximations. As such there are several promising

paths forward to achieve further scalability beyond simply utilising hardware acceleration, including *(i)* applying the estimator developed by Hensman et al. (2013) to our method to utilise mini batches of data, *(ii)* embedding the infinite-horizon approximation introduced by Solin et al. (2018) to trade off some accuracy for a substantial reduction in the computational complexity of our approximation, *(iii)* removing the constraint that observations must appear at the same time as pseudo-points by utilising the method developed by Adam et al. (2020).

Indeed, in the time since the contents of this chapter were published, Wilkinson et al. (2021) and Hamelijnck et al. (2021) have utilised the results presented to justify their approach to combining state-space and pseudo-point approximations. In particular, both works consider non-Gaussian likelihoods, and Hamelijnck et al. (2021) consider mini-batches of data.

Code `github.com/JuliaGaussianProcesses/TemporalGPs.jl` contains an implementation of the approximation developed in this work.

`github.com/willtebbutt/PseudoPointStateSpace-UAI-2021` contains code built on top of TemporalGPs.jl to reproduce the experiments.

Chapter 4

Towards Gaussian Processes for Decadal Climate Prediction

In this chapter, I investigate the potential for the use of GP-based models to address a variety of limitations present in techniques used for *decadal prediction*, the problem of forecasting properties of the climate 5 to 20 years into the future. I elucidate the techniques that are currently being used in this field, and how they are related to GP-based models used within the machine learning community. Some limitations of these techniques, in conjunction with properties of the data available, motivate the development of a new GP model. This new GP model retains the core assumptions present in the existing techniques, but removes some of the limitations. Approximate inference in this new model is not entirely straightforward, and various approaches are explored, which forms the main contribution of this chapter. Proof-of-concept experiments on synthetic data are utilised to demonstrate the efficacy of the approximate inference routines developed. Basic experiments on real sea surface temperature data indicate that the model is successfully capturing important high-level features of the climate, suggesting that addressing remaining scalability issues and testing the model on large-scale decadal prediction experiments will be fruitful.

4.1 The Decadal Prediction Problem

Decadal Climate Prediction (Hawkins and Sutton, 2009b; Meehl et al., 2009) is the study of the behaviour of the climate over periods somewhere between a couple of years and a couple of decades into the future, and sits between seasonal and multi-decadal forecasting. It is one

of the World Climate Research Programme's (WCRP's) seven grand challenges,[1], Meehl et al. (2014) cite water management as an important use-case for decadal predictions, and Fiedler et al. (2021) highlight that it is an important planning time horizon for a variety of financial institutions.

Decadal Climate Prediction differs from weather forecasting, seasonal forecasting, and long term prediction in terms of where it derives its potential predictability. In particular, weather and seasonal forecasting are driven largely by short term natural variability, such as the El Niño–Southern Oscillation (ENSO), while long term forecasting on time scales greater than 20 years or is dominated by anthropogenic climate change. Decadal prediction, on the other hand, comprises a mixture of the two. Hawkins and Sutton (2009b) analyse the relative importance of different source of uncertainty for predictions over the 100 years between 2000 and 2100. They show that the uncertainty about the state of regional climate on time horizons up to a few decades is driven to a non-negligible extent by natural *internal variability* in the climate. Thus a reduction in uncertainty on decadal time horizons might be achieved by exploiting predictability in this internal variability. The most famous and pronounced example of internal variability is ENSO. It has been known to be predictable on a period of up to a year in advance a long time (Kirtman et al., 2002), and improvements in forecasts at longer time horizons remain a topic of active research – recent developments utilising modern deep learning techniques increasing this to a year and half (Ham et al., 2019). ENSO, however, is not hugely relevant on decadal time scales as it is not predictable on them. Instead, Meehl et al. (2009) point towards a variety of lower-frequency oscillations in the oceans as a potential source of predictability on decadal time scales. The include the Pacific Decadal Oscillation (Deser et al., 2004; Mantua and Hare, 2002; Mantua et al., 1997), the Interdecadal Pacific Oscillation (Power et al., 1999), Atlantic meridional overturning circulations (Delworth et al., 1993), and the Atlantic Multidecadal Oscillation (Delworth and Mann, 2000; Kushnir, 1994).

While the potential sources of predictability are to be found in the oceans, the most important impacts are associated with weather over land. There are, however, links between ocean state and weather over land. For example, Foland et al. (1986) and Folland et al. (1991) show that there is a strong relationship between wet and dry periods in the Sahel region of Africa and global SSTs, with a particularly pronounced relationship with Atlantic SSTs and rainfall over the Western Sahel, on up to decadal time scales.

[1] `https://www.wcrp-climate.org/grand-challenges/gc-near-term-climate-prediction`

4.1.1 Approaches to Decadal Prediction

There are broadly two approaches taken to decadal prediction. One approach involves setting the initial conditions of a large physics-driven climate simulator to the current state of the climate, and running it forwards in time for the required duration. For example, this is the approach adopted by those involved in the Decadal Climate Prediction Project (Boer et al., 2016). Alternatively, numerous works (Foster et al., 2020; Hawkins et al., 2011; Hawkins and Sutton, 2009a; Huddart et al., 2017; Newman, 2007, 2013; Zanna, 2012) adopt a more directly data-driven model, utilising a type of model known as a *Linear Inverse Model* (LIM), introduced by (Penland, 1989; Penland and Magorian, 1993; Penland and Sardeshmukh, 1995). This is typically referred to as a *statistical* approach within the climate literature, in contrast with an approach based on a *physical* model of the climate, and it is on this approach that this chapter focusses.

Let $\mathbf{y}_t \in \mathbb{R}^D$ be the high dimensional vector of observations associated with some climatological phenomenon for each time $t \in \{1, ..., T\}$. For example, $\mathbf{y}_t$ might contain a grid of sea surface temperature data at time t. That is, the d^{th} element of $\mathbf{y}_t$ might be the average sea surface temperature over a small spatial region, at time t. Predictions are made with a LIM as follows.

Compute Empirical Orthogonal Functions (EOFs). In Machine Learning parlance, the EOFs are simply the principal components of the collection of data $\mathbf{y}_{1:T}$. They are computed in the usual manner, by computing the first $J < D$ eigenvectors and eigenvalues of the $D \times D$ empirical covariance matrix associated with $\mathbf{y}_{1:T}$. This produces a collection of J principal components, $\mathbf{h}_j \in \mathbb{R}^D$, and eigenvalues $\lambda_j > 0$. Let $\mathbf{H} \in \mathbb{R}^{D \times J}$ be the orthogonal matrix whose j^{th} column is $\mathbf{h}_j$. Let $\Lambda \in \mathbb{R}^{J \times J}$ be the diagonal matrix for which $\Lambda_{jj} := \sqrt{\lambda_j}$.

Project Data For each t, compute $\mathbf{x}_t := \Lambda^{-1} H^\top \mathbf{y}_t$.

Estimate Latent Dynamics Assume that $\mathbf{x}_{1:T}$ are generated by the stationary Gauss-Markov process $\mathbf{x}_t \mid \mathbf{x}_{t-1} := \mathbf{B}\mathbf{x}_{t-1} + \xi_t$, where $\xi_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. Since $\mathbf{x}_t$ is assumed stationary with $\mathbf{0}$-mean, the following holds:

$$\begin{aligned} \mathbf{C} &:= \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right] = \mathbf{B}\mathbf{C}\mathbf{B}^\top + \mathbf{Q}, \\ \mathbf{C}_1 &:= \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_{t-1}^\top\right] = \mathbf{B}\mathbf{C}. \end{aligned}$$

This motivates the following procedure to estimate $\mathbf{B}$ and $\mathbf{Q}$:

1. $\mathbf{C} \leftarrow \sum_{t=1}^{T} \mathbf{x}_t \mathbf{x}_t^\top$ (compute empirical covariance of $\mathbf{x}_t$)
2. $\mathbf{C}_1 \leftarrow \sum_{t=2}^{T} \mathbf{x}_t \mathbf{x}_{t-1}^\top$ (compute empirical cross-covariance between $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$)
3. $\mathbf{B} \leftarrow \mathbf{C_1} \mathbf{C}^{-1}$
4. $\mathbf{Q} \leftarrow \mathbf{C} - \mathbf{B}\mathbf{C}\mathbf{B}^\top$

Prediction Knowing the model parameters, a single-step ahead prediction is made as follows:

1. $\mathbf{x}_{T+1} \mid \mathbf{x}_T \sim \mathcal{N}(\mathbf{B}\mathbf{x}_T, \mathbf{Q})$
2. $\mathbf{y}_{T+1} \mid \mathbf{x}_{T+1} := \mathbf{H}\Lambda\mathbf{x}_{T+1}$

where $\mathbf{x}_{T+1}$ can be marginalised over. Multi-step prediction follows in a similar manner.

There are a few immediate limitations of the above methodology, including:

- complete data is required – the above procedure is not designed to work with missing data,
- the d^{th} dimension of each $\mathbf{y}_t$ must correspond to the same measurement location at different points in time for this approach to be reasonable. This means that roaming sensor data cannot be handled, and
- piecewise learning of dynamics, meaning that $\mathbf{B}$, $\mathbf{Q}$, $\mathbf{H}$, and Λ are learned somewhat separately.

However, the following generative model yields the same posterior mean prediction as this procedure:

$$\mathbf{x}_t \mid \mathbf{x}_{t-1} \sim \mathcal{N}(\mathbf{B}\mathbf{x}_{t-1}, \mathbf{S}) \tag{4.1}$$

$$\mathbf{y}_t \mid \mathbf{x}_t \sim \mathcal{N}\big(\mathbf{H}\Lambda\mathbf{x}_t, \sigma^2\mathbf{I}\big)\,. \tag{4.2}$$

This result follows from the orthogonality of $\mathbf{H}$, as discussed by Bruinsma et al. (2020). This probabilistic model differs, however, in terms of the marginal predictive variance, due to the $\sigma^2\mathbf{I}$ term, and the fact that inference in this model will marginalise over plausible values of $\mathbf{x}_{1:T}$, while the LIM procedure produces a point estimate for $\mathbf{x}_T$, and propagates this forward through time. This model-based probabilistic interpretation of LIMs provides the starting point for the model introduced later in this chapter, which addresses all three of the above limitations.

Note that, to the best of my knowledge, there has been minimal attention paid to the decadal prediction problem within the machine learning community. Preliminary work by Rodrigues et al. (2021) investigates replacing the Gauss-Markov model over $\mathbf{x}_{1:T}$ with an RNN, but this seems to be the extent of the work in the area.

4.2 Datasets and their Properties

Ocean-related data has a long history. It has been gathered in many different ways over the last 300 years or so, resulting in a highly complicated array of historical data, the details of which constitute an entire field of study.

There are some fascinating data-related stories. For example, Folland and Parker (1995) discusses how much of the available data was gathered by ships during the course of other business, but the manner in which most ships gathered data changed substantially around 1941. Before this point in time, buckets were used to collect ocean water, and a thermometer used to measure the temperature on the deck of the ship. However, around 1941 practices changed rapidly, and ships started measuring temperature at their engine intake instead.[2] There is an easily-visible jump in the annual average SST measurements in 1941, and extensive work has been conducted to attempt to deal with this.

This is just one well-known example, but there are many more examples. The presence of these difficulties means that ocean data is not straightforward, and it would be extremely challenging to work with raw observational data as a non-expert. This is not to say, however, that the best choice of data set is necessarily heavily pre-processed, or even one which has been gridded – climate scientists have produced numerous high-quality data sets containing oceanographic data with a variety of properties, some of which are not gridded. Since the properties of the available data sets dictate the kinds of models that can be usefully built, I investigate below those that are available, highlighting their important properties.

HadISST Rayner et al. (2003) introduce the HadISST data set. One methodology they utilise in their construction of the data set is that discussed by Kaplan et al. (1997), who construct a simple low-rank linear dynamical system of the form

$$\mathbf{x}_{t+1} \mid \mathbf{x}_t \sim \mathcal{N}(\mathbf{A}\mathbf{x}_t, \mathbf{Q}) \tag{4.3}$$

$$\mathbf{y}_t \mid \mathbf{x}_t \sim \mathcal{N}(\mathbf{H}_t\mathbf{x}_t, \mathbf{S}_t) \tag{4.4}$$

[2]Folland and Parker (1995) suggest that this relates to the risks associated with shining a torch light on deck, which is required in order to take the measurement, in war-time.

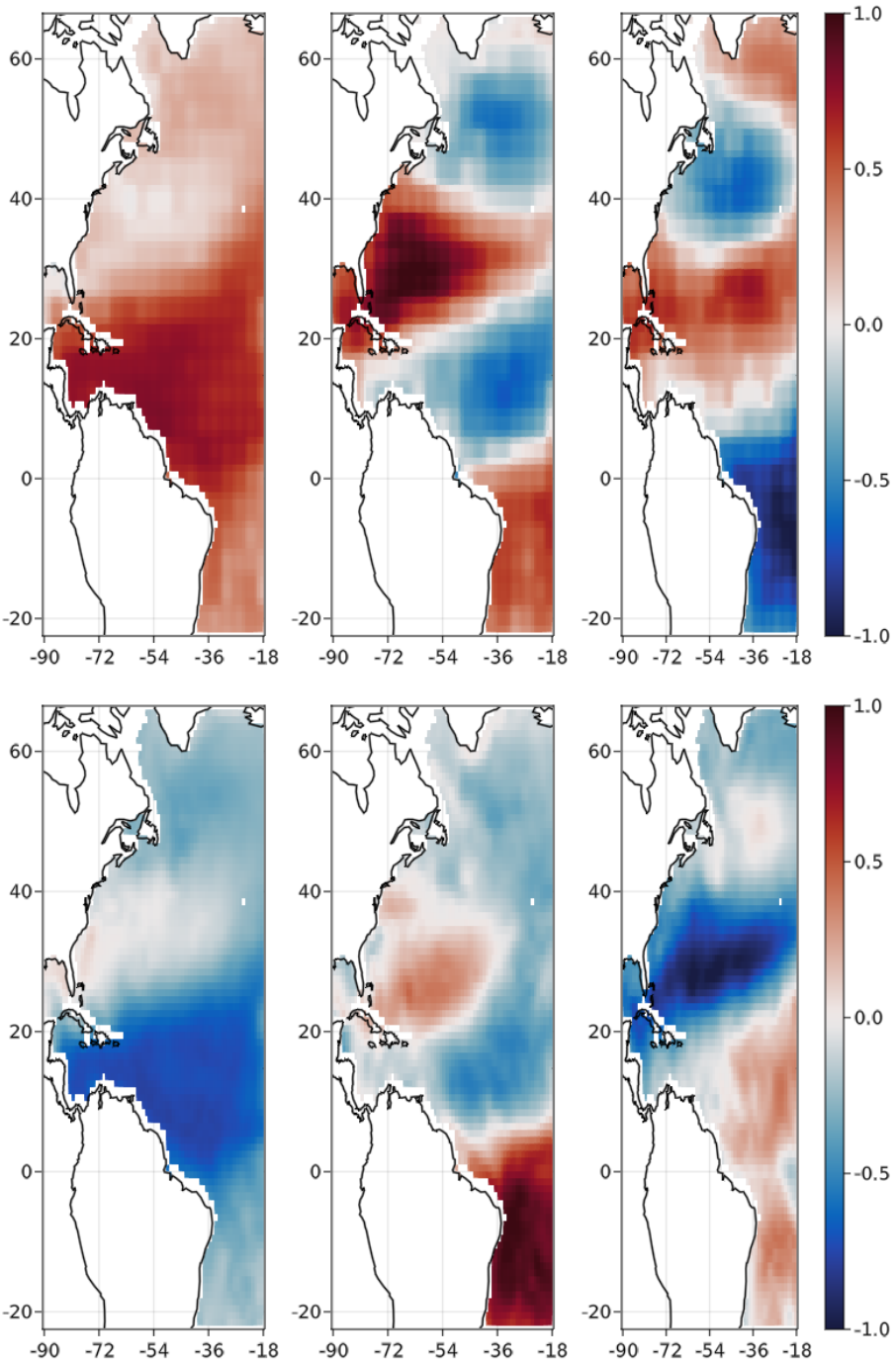


Fig. 4.1 HadISST EOFs pre-1945 (top) and post-1945 (bottom). We see that the pre-1945 HadISST data set is half the resolution of the data set post-1945. This is noted in the paper introducing the data set. Moreover it seems that while the first EOF is fairly stable across time periods, the second and third differ noticeably (after accounting for arbitrary sign changes).

where $\mathbf{y}_t$ is a high-dimensional spatial field at time t, and $\mathbf{x}_t$ a low-dimensional latent state at time t, and $\mathbf{H}_t$ is a mixture of EOFs and some interpolation terms to handle grid-misalignment and missing data. $\mathbf{A}$, $\mathbf{Q}$, and $\mathbf{S}_t$ are all estimated in an ad-hoc manner, and are restricted to be diagonal. The posterior mean of this model is returned as the temperature estimate, and is computed using RTS smoothing. This methodology is interesting, because it is very closely related to LIMs, suggesting that it should perhaps be unsurprising if the HadISST data set is explained well by a small number of EOFs.

Rayner et al. (2003) do not actually use this methodology in isolation – in particular they drop the temporal dependence in the model due to a paucity of training data, separately model and subtract the anthropogenic forcing trend in the temperature, utilise different EOFs at different points in time / space, merge the output of the model with observational data (to account for a loss of variability due to utilising the posterior mean), and apply some other quality controls to avoid incorporating corrupted data into the final analysis. If one were to take a model-based perspective on their procedure, most of these pre-processing steps might be avoided.

Interestingly, Kaplan et al. (1997) are certainly aware of the model-based / Bayesian interpretation of returning the posterior mean of the above model, specifically highlighting that the maximum likelihood solution aligns with their smoothing solution when the observation noise is Gaussian. However, they opt for a frequentist interpretation, citing the well-known result that smoothing solution / posterior mean is optimal under a wide variety of criteria (e.g. least-squares) for a range of measurement error distributions.

For the present use-case, HadISST has a number of potential shortcomings. Firstly, it utilises a substantial amount of pre-processing to produce the data set, and it is unclear how these techniques will affect the outcome of any analysis that we perform. Secondly, it provides no quantification of the uncertainties associated with its creation, which is potentially problematic in data-sparse regions. Moreover, Chelton and Risien (2016) highlight issues in HadISST, particularly around the centre of the Pacific and changes around 1941. Another discrepancy is the apparently substantial change in spatial resolution around 1945 – Fig. 4.1 compares the first three EOFs pre- and post-1945. Arguably the primary advantage of HadISST is that it lacks missing data, but since this work develops a model that is straightforwardly able to handle missing data, this advantage is not hugely pertinent here.

HadSST HadSST4 (Kennedy et al., 2019) is an interesting data set, as it is constructed using only the minimum amount of processing necessary to provide a gridded data set, and leaves missing data where appropriate. Moreover, unlike HadISST, it provides a careful

account of the uncertainties associated with the data, which are especially important at points in time / space with few observations. Sufficient detail is provided in the user-manual about the interpretation of these uncertainties to mean it is plausible to utilise them as part of a measurement model in a probabilistic model.

ICOADS The International Comprehensive Ocean-Atmosphere Data Set (ICOADS) (Freeman et al., 2017) is a vast database, containing surface marine measurements starting in 1662. It is used as a component in the construction of numerous other data sets. Ship records form the bulk of historical measurements, but in recent times many other sources of data have become available. It is quite a low-level data set, with little more than the raw data available.

EN4 Good et al. (2013) introduce EN4, the fourth version of a dataset containing in situ temperature and salinity profiles. It provides two data products: profiles and analysis. The profiles are obtained from a variety of sources, and have a collection of sanity checks applied (removal of duplicates, clearly-erroneous temperature / salinity measurements, large jumps in values, etc).

HadIOD Atkinson et al. (2014) introduced the Hadley Integrated Ocean Database (HadIOD). It is drawn from ICOADS and EN4, and comprises in situ temperature and salinity measurements at a variety of depths, and has a range of quality controls applied (see Ingleby and Huddleston (2007) for details). Version 1 of the dataset spanned 1900 to present, but was extended in version 1.2 to cover 1850 to present (amongst other things) by Brönnimann et al. (2018), and it is updated on a regular basis. The in situ nature of the data necessitates the use of statistical models which can work directly on this off-the-grid roaming sensor data.

While the data is highly detailed, the authors of the data set have gone to great lengths to make it simple for those outside the measurement community to understand and work with, applying a range of quality checks and summarising a large amount of domain expertise about biases in the data and their uncertainties for every single observation. As with HadSST3 and HadSST4, their documentation provides enough detail to construct measurement-specific observation models.[3] It is a truly remarkable data set.

Each measurement y is a small model of the form

$$y = f + \varepsilon_M + \varepsilon_m + \varepsilon \tag{4.5}$$

[3] https://www.metoffice.gov.uk/hadobs/hadiod/HadIOD.1.2.0.0_Product_User_Guide_[1.1].pdf

where f is the true unobserved value, ε_M is an error associated with all measurements made with the same device / platform as y (a particular kind of bucket, engine room inlet, XBT, Argo float, etc), ε_m is an error associated with the particular platform used to make the measurement (a particular ship, XBT, Argo float, etc), and ε is independent noise associated with this particular measurement.

The manner in which the uncertainties are provided varies. ε is characterised by a standard deviation, and is assumed to be a zero-mean Gaussian with this standard deviation. ε_m is characterised by a "correction" (bias) and a standard deviation for this bias – again the authors advise that it can be assumed to be Gaussian in most cases, but will sometimes be a poor approximation to the true uncertainty. ε_M is provided in the form of an ensemble of different corrections (roughly 300 samples) – this is perhaps the most tricky representation of uncertainty to handle in a probabilistic model. One possible probabilistic interpretation of the above is

$$\varepsilon_{M,p} \sim S^{-1} \sum_{s=1}^{S} \delta(c_s) \tag{4.6}$$

$$\varepsilon_{m,q} \sim \mathcal{N}\left(c_q, \nu_q^2\right) \tag{4.7}$$

$$y_n \mid f_n, \varepsilon_{M,1:P}, \varepsilon_{M,1:Q} \sim \mathcal{N}\left(f_n + \varepsilon_{M,p(n)} + \varepsilon_{m,q(n)}, \sigma_n^2\right) \tag{4.8}$$

where

- there are P platform types and $\varepsilon_{M,p}$ denotes the error associated with the p^{th},
- $\delta(x)$ denotes a delta distribution / atom centred at x,
- there are S ensemble members used to express uncertainty about $\varepsilon_{M,p}$,
- c_s is the correction in the s^{th} of them,
- there are Q platforms and $\varepsilon_{m,q}$ denotes the error specific to the q^{th},
- c_q is the bias quoted for the s^{th} platform, and ν_q its standard deviation,
- f_n is the n^{th} true unobserved temperature,
- y_n is the n^{th} temperature measurement,
- $p(n)$ and $q(n)$ are the platform type and platform index associated with the n^{th} observation,
- σ_n is the standard deviation quoted for the independent noise associated with the n^{th} observation.

This model respects the relationship between the biases associated with each observation, and appears tractable – Eq. (4.6) is just a finite discrete mixture, so it can be marginalised via summation. An analogous model could be constructed for the salinity data.

In addition to this error-related data, latitude, longitude and depth are provided for each measurement, and an uncertainty estimate for depth is provided.

4.3 The Infinite Linear Mixing Model

The availability of high-quality in situ data in the HadIOD data set, and the relationship between LIMs and the probabilistic model given in Eq. (4.1) and Eq. (4.2), suggests that one might be able to specify and train / perform inference in a generalisation of this probabilistic model which operates in continuous-space but retains core components of LIMs. The inability of LIMs to handle roaming sensor data and the patchy coverage of historical ocean data suggest that building such a model, and propagating uncertainty information throughout it, might be fruitful. Such an approach would be able to make use of all available data, and the consistent propagation of uncertainty obtained by performing approximate Bayesian inference should result in a model with an appropriate degree of belief in its predictions, up to model mis-specification.

To that end, consider the following probabilistic model:

$$
\begin{aligned}
x_j &\sim \mathcal{GP}\left(0, \kappa_j^{\tau}\right), \quad j \in \{1, ..., J\}, \\
h_j &\sim \mathcal{GP}\left(0, \kappa_j^{\mathbf{r}}\right), \quad j \in \{1, ..., J\} \\
x(\tau) &:= \left(x_1(\tau), ..., x_J(\tau)\right), \\
h(\mathbf{r}) &:= \left(h_1(\mathbf{r}), ..., h_J(\mathbf{r})\right) \\
f((\mathbf{r}, \tau)) &:= \langle h(\mathbf{r}), x(\tau) \rangle \qquad (4.9) \\
y_n \mid f &\sim \mathcal{N}\left(f((\mathbf{r}_n, \tau_n)) + b_n, \sigma^2\right), \quad n \in \{1, ..., N\}. \qquad (4.10)
\end{aligned}
$$

Here, a sample from x_j maps $\mathbb{R} \to \mathbb{R}$, and a sample from h_j maps $\mathbb{R}^2 \to \mathbb{R}$. Consequently, a sample from f maps $\mathbb{T}(\mathbb{R}^2, \mathbb{R}) \to \mathbb{R}$, where $\mathbb{T}(A, B)$ denotes the set of all tuples of the form $\{(a, b) : a \in A, b \in B\}$. b_n is an observation-specific bias obtained from the HadIOD data set, and $\sigma > 0$.

In this model, $x(\tau)$ corresponds to $\mathbf{x}_t$ in the LIM – $x_{1:J}$ gives the evolution of a collection of J *latent* processes through time. Similarly, samples from $h_{1:J}$ provide continuous-in-space *basis* functions, generalising $\mathbf{H}\Lambda$ in the LIM, and enabling predictions to be made anywhere

in space. Thus this model retains the key inductive bias in the LIM, that the data can be explained at any given point in time through a time-varying linear combination of bases. However, by generalising the model to continuous space roaming sensor data in HadIOD can be utilised. Moreover by placing a prior over h and performing approximate inference in it, uncertainty about it can be propagated through to predictions.

The joint distribution over x, h, f and $y_{1:N}$ is not Gaussian, owing to f being the inner product between x and h. However, the conditionals $x, f, y_{1:N} \mid h$ and $h, f, y_{1:N} \mid x$ are both Gaussian, leading to the following two important formulations of the conditionals.

Basis-Given-Latent Formulation The conditional distribution over h and f given x can be expressed as simple two-layer GPPP:

$$h \sim \mathcal{GP}(0, \kappa^{\mathbf{r}}), \tag{4.11}$$

$$f = \mathcal{L}_x h, \tag{4.12}$$

where $\mathcal{L}_x$ is the linear transformation given by

$$(\mathcal{L}_x h)((\mathbf{r}, \tau)) := \langle h(\mathbf{r}), x(\tau) \rangle, \tag{4.13}$$

and, in order to simplify notation, $h_{1:J}$ have been compressed into a single multi-output GP h with kernel $\kappa^{\mathbf{r}} : \mathbb{T}(\{1, ..., J\}, \mathbb{R}^2) \times \mathbb{T}(\{1, ..., J\}, \mathbb{R}^2) \to \mathbb{R}$, given by

$$\kappa^{\mathbf{r}}((j, \mathbf{r}), (j', \mathbf{r}')) = \begin{cases} \kappa_j^{\mathbf{r}}(\mathbf{r}, \mathbf{r}') & j = j' \\ 0 & \text{otherwise} \end{cases}.$$

h is an atomic process, so no further work is required to incorporate it into the GPPP abstraction. However, $\mathcal{L}_x$ is a linear transformation that has yet to be encountered, so it must be analysed.

Recall from $Chapter\ 2$ that the domain of a GPPP comprises tuples. Since the domain of $\mathcal{L}_x h$ itself comprises tuples, if $\mathcal{L}_x h$ is the P^{th} component in a GPPP, then it can be addressed using a nested tuple: $(P, (\mathbf{r}, \tau))$.

Assume that $\mathcal{L}_x$ is applied to the p^{th} process in a GPPP. Assuming that the p^{th} process is multi-output, it has mean function, kernel, and cross-kernels as follows:

$$\begin{aligned} m_p((j, \mathbf{r})) &:= m((p, (j, \mathbf{r}))) \,, \\ \kappa_p((j, \mathbf{r}), (j', \mathbf{r}')) &:= \kappa((p, (j, \mathbf{r})), (p, (j', \mathbf{r}'))) \,, \\ \kappa_{pq}((j, \mathbf{r}), x') &:= \kappa((p, (j, \mathbf{r})), (q, x')) \,. \end{aligned}$$

Since it is a multi-output process, we can further associate to it the following vector-valued mean function and cross-covariance function, and matrix-valued covariance function:

$$\begin{aligned} \mathbf{m} : \mathbb{R}^2 \to \mathbb{R}^J, \text{ where } \mathbf{m}(\mathbf{r})_j &:= m_p((j, \mathbf{r})) \,, \\ \mathbf{c}_q : \mathbb{R}^2 \times \mathcal{X}_q \to \mathbb{R}^J, \text{ where } \mathbf{c}_q(\mathbf{r}, x')_j &:= \kappa_{pq}((j, \mathbf{r}), x') \,, \\ \mathbf{C} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^{J \times J}, \text{ where } \mathbf{C}(\mathbf{r}, \mathbf{r}')_{jj'} &:= \kappa_p((j, \mathbf{r}), (j', \mathbf{r}')) \,. \end{aligned}$$

Given the above, the concrete expressions for Eq. (2.17), Eq. (2.18), and Eq. (2.19) in the context of $\mathcal{L}_x$ are

$$\begin{aligned} m'((P, (\mathbf{r}, \tau))) &= \langle x(\tau) \,, \mathbf{m}(\mathbf{r}) \rangle \,, \\ \kappa'((q, x), (P, (\mathbf{r}', \tau'))) &= \langle \mathbf{c}_q(\mathbf{r}, x) \,, x(\tau) \rangle \,, \\ \kappa'((P, (\mathbf{r}, \tau)), (P, (\mathbf{r}', \tau'))) &= x(\tau)^\top \, \mathbf{C}(\mathbf{r}, \mathbf{r}') \, x(\tau') \,. \end{aligned}$$

The benefit of treating this as a GPPP is precisely as discussed in Chapter 2 – existing software for pseudo-point approximations can be straightforwardly utilised within this GPPP.

One potential problem with this formalism is that implementing it in a generic manner requires access to x at any τ. In practice, this is not a problem as the data occurs in discrete-time, meaning that it is possible restrict τ to be a natural number, and produce samples from x for all relevant times.

Latent-Given-Basis Formulation The conditional distribution over x and f given h can also be expressed as a small GPPP:

$$x \sim \mathcal{GP}(0, \kappa^\tau) \,, \tag{4.14}$$

$$f = \mathcal{L}_h x, \tag{4.15}$$

This model is completely symmetric to the previous, in the sense that simply replacing the κ^τ with $\kappa^{\mathbf{r}}$, x with h etc everywhere produces all of the above results.

Related Models There are a couple of models in the GP literature which are closely related to this model. The first is the Linear Mixing Model (LMM), of which the above can be seen as a generalisation to infinitely many outputs. The LMM is itself a small generalisation of the Linear Model of Coregionalisation introduced by Goovaerts (1997), which is an extensively-utilised multi-output GP model – see Bruinsma et al. (2020) for a review. It has P outputs, and is given by

$$\begin{aligned} x_j &\sim \mathcal{GP}\left(0, \kappa_j^\tau\right), \quad j \in \{1, ..., J\}, \\ x(\tau) &:= (x_1(\tau), ..., x_J(\tau)), \\ f((p, \tau)) &:= \langle h(p), x(\tau) \rangle, \quad p \in \{1, ..., P\}, \\ y_n \mid f &\sim \mathcal{N}\left(f((p_n, \tau_n)) + b_n, \sigma^2\right), \quad n \in \{1, ..., N\}, \end{aligned}$$

for a collection of P basis vectors $h_p \in \mathbb{R}^J$. The Infinite Linear Mixing Model is obtained from the Linear Mixing Model by replacing the integer index p with the continuous spatial index $\mathbf{r}$, and generalising h to be a function which is continuous in space.

While a point estimate of each h_p is typically obtained for the Linear Mixing Model, a prior is placed over h in this work. For one, some form of regularity must be imposed upon h as it comprises infinitely many vectors (one for each point in space), and a GP prior offers that. Additionally it will be important to infer the characteristic length scale of the spatial bases directly from data. This is straightforwardly done using the ELBO if variational inference is performed over h, but will necessitate a tricky hyperparameter search algorithm if a ridge-regression style point estimate is obtained.

Note that while the above discussion focuses on LMMs with multiple outputs but only a single temporal dimension, they trivially extend to higher-dimensional domains by replacing each x_j with processes defined on the higher-dimensional domain. Precisely the same mechanism could be used to extend the Infinite Linear Mixing Model to higher dimensions. For example, for the domain $\mathbb{R}^4$, two dimensions could be associated with x_j and two with h_j. This would induce an interesting restricted class of conditionally-Gaussian model, which may be of general interest, but which is not considered here.

The Infinite Linear Mixing Model is a special case of the Gaussian process Regression Network (GPRN) developed by Wilson et al. (2012). The GPRN is given by

$$\begin{aligned} x_j &\sim \mathcal{GP}\left(0, \kappa_j^{\tau}\right), \quad j \in \{1, ..., J\}, \\ h_{pj} &\sim \mathcal{GP}\left(0, \kappa_{pj}^{\mathbf{r}}\right), \quad p \in \{1, ..., P\} \\ x(\mathbf{x}) &:= \left(x_1(\mathbf{x}), ..., x_J(\mathbf{x})\right), \\ h((p, \mathbf{x})) &:= \left(h_{p1}(\mathbf{x}), ..., h_{pJ}(\mathbf{x})\right) \\ f((p, \mathbf{x})) &:= \left\langle h((p, \mathbf{x})), x(\mathbf{x}) \right\rangle, \\ y_n \mid f &\sim \mathcal{N}\left(f((p_n, \tau_n)) + b_n, \sigma^2\right), \quad n \in \{1, ..., N\}. \end{aligned}$$

In effect, it is an LMM in which the basis vectors vary as a function of the input. Letting $P = 1$, $\mathbf{x} := (\mathbf{r}, \tau)$, if one restricts x to depend only on τ, and h only on $\mathbf{r}$, the Infinite Linear Mixing Model is recovered. Inference in the GPRN is known to be tricky, however, since the LMM is substantially simpler, one would hope that approximate inference will prove to be more straightforward. For example, the partition of the dimensions should mean that the number of pseudo-points required to provide good coverage of the spatial bases does not grow as more data is gathered through time. Conversely, the number of random variables in which inference must be performed in x does not grow if the spatial extent increases. Neither of these is true in the general GPRN case.

From the perspective of approximation inference, a key feature of the model considered here is the presence of the product of GPs. This basic kind of conditionally-Gaussian structure seen in the Infinite Linear Mixing Model occurs in a couple of other places in the literature. Tobar et al. (2015) and Bruinsma et al. (2022) consider a model in which both processes range through time, and convolve one against the other to produce a new process. They show that a Gibbs sampling procedure can be developed which samples from the optimal variational approximate posterior over the two processes, from which an estimator for the gradient of the log marginal likelihood w.r.t. the hyperparameters can be obtained. They note that this works extremely well in the problems they consider, but they only utilise a small number of data points. Consequently, given the need for the use of mini batches of data in the present work, their results are not obviously immediately transferable.

One important lesson from their work is that a mean-field approximation, in which the approximate posterior factorises across processes (i.e., $q(x, h) := q(x)\, q(h)$), is unlikely to provide a good approximation.

The rest of this section explores the various approaches one could take.

4.3.1 Approximate Inference

Exact inference in all of the components of the Infinite LMM is intractable and, since the HadIOD data set comprises hundreds of millions of observations in total, exact inference will not be feasible in the conditionals. Rather, it is necessary to find a posterior approximation in a manner which permits utilising mini batches of the observational data. The best approach available for this remains the sparse variational pseudo-point approximation discussed in Chapter 1. Consequently, any approximate inference algorithm will need to be built around this kind of approximation in some way.

Having chosen to utilise the variational pseudo-point approximation, there are numerous choices to be made about the details which have a substantial impact on the performance, and are discussed below.

Mean Field The most naïve parametrisation considered here for the approximate posterior is to factorise it across the two latent GPs: $q(x, h, f) := q(x)\, q(h)\, p(f \mid x, h)$. Each factor is separately approximated using a pseudo-point approximation. In particular $q(x)$ utilises the Conjugate Computation Variational Inference (CVI) technique developed by Khan and Lin (2017), recently considered in the context of pseudo-point GP approximations by Adam et al. (2021), while $q(h)$ utilises the centred parametrisation introduced in Sec. 1.3.2, with the pseudo-observation covariance matrix restricted to be diagonal. The ADAM optimiser (Kingma and Ba, 2015) with a learning rate of 10^{-2} is utilised to optimise the parameters of $q(h)$ and all kernel parameters, while natural gradient ascent is utilised for the parameters of $q(x)$, as per Adam et al. (2021).

The ELBO is computed making use of the latents-given-bases GPPP formulation of the model, as this was easy to implement as it minimised the amount of code which needed to be written. A differentiable Monte Carlo estimator for the ELBO is constructed. It specifically makes use of

$$\mathcal{L}(\phi, \theta) = \hat{\mathcal{L}}_{\text{inner}}(h, \phi, \theta) - \mathcal{KL}[q(\mathbf{u}; \phi, \theta) \,\|\, p(\mathbf{u}; \theta)]\,, \quad h \sim q(h)\,, \tag{4.16}$$

where $\hat{\mathcal{L}}_{\text{inner}}$ is the standard estimator for the unsaturated ELBO, presented in Eq. (1.17), applied to the latents-given-bases conditional model shown in Eq. (4.14) and Eq. (4.15). Pseudo-inputs are located in x, the component of the GPPP, but not the second, f.

Latents-given-Bases Coupled Approximation A more sophisticated approximation utilises the same $q(h)$ as above, but conditions the posterior over x on h. Since the prior

distribution over x given h is a GP, and the observation model is Gaussian, for small data sets it is possible to perform exact inference. In this situation, it is possible to simply let $q(x, f \mid h) := p(x, f \mid h, \mathbf{y})$ (i.e., the exact posterior of the first GPPP above), which is the best possible choice. However, the complexity of ELBO computation under this choice for $q(x, f \mid h)$ is linear in the number of observations, which is prohibitive for large data sets such as the one under consideration here. Consequently, this choice provides only a good baseline against which to compare other methods in small scale settings.

So instead of conditioning on $\mathbf{y}$, one could condition on a small pseudo-observational data set, and let $q(x, f \mid h)$ be the exact posterior given this pseudo-observational data set. The existing formulations for pseudo-observation approximations discussed in Sec. 1.3, however, tie the location of the pseudo-observations to those of the pseudo-points, which is problematic in this setting for the following reasons.

On the one hand, if pseudo-points are placed in x, the useful size of the pseudo-observation data set in this particular setting is strictly limited. To see this, consider that there are a total of JT unique dimensions of x of which any indirect observation is made owing to making observations in discrete time at a total of T unique time points. Therefore, the number of pseudo-points which can usefully be placed in x is JT. Worse still, placing pseudo-observations and pseudo-points in x yields a mean-field approximation, because the prior over x is not a function of h. The only other place to locate the pseudo-points is in f. However, this is again sub-optimal for precisely the computational reasons discussed in the context of additive GPs in Sec. 2.6.1.

Instead, one could decouple the inputs associated with the pseudo-observations from the inputs associated with the pseudo-points. Placing pseudo-observations in f and pseudo-points in x should achieve the best of both worlds, yielding an approximate posterior in which x, f, and h are all coupled, but with better computational properties than placing both pseudo-points *and* pseudo-observations in f.

To see how this is achieved in general, abstract away the details of this particular problem, and instead consider the standard GP regression problem. Let f be a GP with domain $\mathcal{X}$, $\mathbf{y} \in \mathbb{R}^N$ a vector of observations made under Gaussian observation noise with variance σ^2 at locations $\mathbf{x} \in \mathcal{X}^N$, and $\mathbf{z} \in \mathcal{X}^M$ a collection of M pseudo-inputs. Letting $\mathbf{u} := f(\mathbf{z})$ as usual, we face the question of how to parametrise $q(\mathbf{u})$. The pseudo-observation parametrisation discussed in Sec. 1.3.2 ties the location of the pseudo-observations to line up precisely with the pseudo-points.

Consider an additional set of L pseudo-observation inputs $\mathbf{w} \in \mathcal{X}^L$ and associated random variables $\mathbf{v} := f(\mathbf{v})$, pseudo-observations $\hat{\mathbf{y}} \in \mathbb{R}^L$, and a pseudo-observation covariance $\mathbf{S}_{\hat{\mathbf{y}}} \in \mathbb{S}_+^L$. Let $q(\mathbf{u})$ be the optimal approximate posterior given this pseudo-observation data set, which by substituting the above into Eq. (1.22) is

$$q(\mathbf{u}) \propto \mathcal{N}(\hat{\mathbf{y}}; \mathbf{C}_{\mathbf{vu}}\Lambda_{\mathbf{u}}\mathbf{u}, \mathbf{S}_{\hat{\mathbf{y}}})\, p(\mathbf{u})\,. \tag{4.17}$$

If $L > M$, this parametrisation introduces some redundancy if $\hat{\mathbf{y}}$ and $\mathbf{S}_{\hat{\mathbf{y}}}$ are not further constrained. In practice, $\mathbf{S}_{\hat{\mathbf{y}}}$ will be constrained to be diagonal, as this means that computing $q(\mathbf{u})$ requires only $\mathcal{O}(LM^2 + M^3)$ operations. I refer to this approximation as a *Decoupled Pseudo-Observation (DPO) Approximation*, because the location and number of pseudo-observations is not tied to the location and number of pseudo-points.

In the present work, $\mathbf{z}$ reside in x, the first component process of the GPPP $p(x, f \mid h)$, while $\mathbf{w}$ reside in f.

Having established the parametrisation of the approximate posterior, an optimisation scheme must be devised for it. The preferred option is natural gradient ascent. Recall that natural gradients w.r.t. the natural parameters of an exponential family are simply the Euclidean gradient w.r.t. the expectation parameters, so if a given q is an exponential family, and it is straightforward to convert between the natural and exponential parameters, it is straightforward to compute the natural gradient w.r.t. the natural parameters.

However, while the conditional distribution $q(x \mid h)$ and marginal distribution $q(h)$ are Gaussian, the joint approximate posterior $q(x, h)$ does not form an exponential family, and it is the joint for which the natural gradient is needed. This makes it very hard to compute the natural gradient, as the entire Fisher information matrix is required. For all intents and purposes, this rules out performing natural gradient ascent w.r.t. the variational parameters in $q(x, h)$.

One might consider performing natural gradient ascent w.r.t. just the variational parameters of the marginal distribution $q(h)$, however, some difficulties are again encountered owing to the requirement that the covariance matrix associated with the pseudo-points be positive definite. To see this, recall that in the Gaussian case it is straightforward to perform natural gradient descent provided that the reconstruction term, or estimator thereof, of the ELBO is log-concave in the latent variable over which q is distributed. This property ensures that the natural gradient is always positive definite, so provided that the second natural parameter is initialised to be positive definite, it will stay that way. However, if it is not log-concave, it is possible to take gradient steps which do not yield a positive-definite result. This necessitates

some additional complications to the gradient ascent procedure (Salimbeni et al., 2018) which are only known to work well in the case of GP models with simple non-Gaussian likelihoods, and in that case seems to requires careful learning rate scheduling. It is unclear, therefore, whether it would be possible to get it to work well in this much more complicated setting. In this setting $\hat{\mathcal{L}}_{\text{inner}}$ acts as the reconstruction term for $q(h)$, and it is not log-concave.

In the end, I opt to utilise the ADAM optimiser with learning rate 10^{-2} with this parametrisation, for all variational and model parameters.

The structure of this approximation family is intimately related to that utilised by Ober and Aitchison (2021) in the context of Deep GPs. Let $f_{lj} \sim \mathcal{GP}(0, \kappa_{lj})$ be the GP producing the j^{th} activation at the l^{th} hidden layer of a Deep GP. The approximate posterior over the pseudo-points $\mathbf{u}$ associated with this GP, given all of the previous layers, is

$$q(\mathbf{u}) \propto \mathcal{N}(\hat{\mathbf{y}}; \mathbf{u}, \mathbf{S}_{\hat{\mathbf{y}}})\, \mathcal{N}(\mathbf{u}; \mathbf{0}, \mathbf{C}_{\mathbf{u}}(\mathbf{u}_{l-1,1:J}))$$

where $\hat{\mathbf{y}} \in \mathbb{R}^M$ and $\mathbf{S}_{\hat{\mathbf{y}}} \in \mathbb{S}_+^M$, and $\mathbf{u}_{l-1,1:J}$ are all of the pseudo-points at the $(l-1)^{th}$ layer. Note the explicit dependency of the pseudo-point covariance matrix $\mathbf{C}_{\mathbf{u}}$ on the pseudo-points from the previous layer. In this case, this dependency is one of composition – the inputs to the kernel used to construct $\mathbf{C}_{\mathbf{u}}$ *are* the pseudo-points in the previous layer. On the other hand, the model presented in this chapter has a multiplicative dependence. Moreover, note that the number of pseudo-observations in this approximation are observations of the pseudo-points, as opposed to being decoupled from them.

Bases-given-Latents Coupled Approximation The bases-given-latents formulation has precisely the same structure as the latents-given-bases formulation, so a conditional parametrisation in the opposite direction can also be made, in which $q(x)$ and $q(h, f \mid x)$ are directly parametrised. To achieve this for $q(h, f \mid x)$, a pseudo-point approximation can be applied to the GPPP specified in Eq. (4.14) and Eq. (4.15). The same considerations as before apply to the placement of pseudo-points, so a DPO approximation is again used. A standard pseudo-point approximation using the non-centred parametrisation is made to x.

4.4 Results

This section contains a collection of proof-of-concept experiments on synthetic data generated from the Infinite LMM, and a subset of HadIOD.

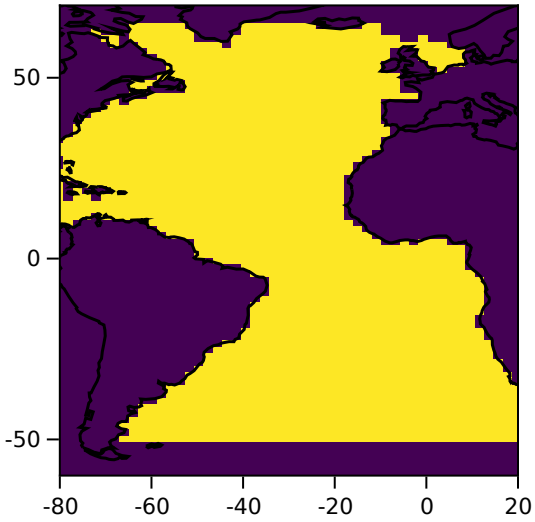


Fig. 4.2 Spatial mask applied to remove any data that lives outside of the Atlantic, or outside of $[-60, 70]$ degrees latitude, and $[-80, 20]$ degrees longitude. Observe that the Mediteranian Sea, South East Pacific, and Southern Ocean are excluded.

General Experimental Details The following details the conditions used in all experiments in this chapter. Firstly, only observations located between $[-60, 70]$ degrees latitude, $[-80, 20]$ degrees longitude are retained, and which reside within the Atlantic Basin are retained. Fig. 4.2 depicts this region.

In all experiments, Exponentiated Quadratic kernels are used in the priors over both x and h. $J = 3$ is used for all experiments. Initial length scales for (x_1, x_2, x_3) are $(10, 5, 1)$ respectively. The kernels for h are ARD, and have two length scales each – the initial length scales for (h_1, h_2, h_3) are $((\frac{1}{2}\Delta_{\text{lat}}, \frac{1}{2}\Delta_{\text{lon}}), (\frac{1}{4}\Delta_{\text{lat}}, \frac{1}{4}\Delta_{\text{lat}}), (\frac{1}{8}\Delta_{\text{lat}}, \frac{1}{8}\Delta_{\text{lat}}))$ where $\Delta_{\text{lat}} := 130 = 70 + 60$ and $\Delta_{\text{lon}} := 100 = 20 + 80$. The variance of each x_j is initialised to $\frac{1}{J}$ and learned, while that of each h_j is fixed to 1 to avoid redundancy in the parametrisation.

100 pseudo points are used per-basis and are located on a fixed regular Cartesian grid, covering $[-60, 70]$ degrees latitude and $[-80, 20]$ degrees longitude. This means that there are a total of $100J$ pseudo-points in the bases. This introduces some redundancy in the pseudo-input placement as data is only present over the ocean, and roughly half of these pseudo-inputs are located over land. For the sake of the experiments in this chapter, the most important thing is to have enough pseudo-inputs – careful optimisation of their placement is left for future work.

Where DPO approximations are made, a total of $10T$ pseudo-observations are utilised, 10 at each point in time. The spatial locations of the inputs associated to these pseudo-observations are sampled uniformly at random from the grid of pseudo-points, independently at each point in time. The pseudo-observation covariance matrix is restricted to be diagonal.

Wherever a standard pseudo-point approximation is utilised to either $q(h)$ or $q(x)$, the non-centred parametrisation is adopted, and the covariance matrix restricted to be diagonal.

Wherever pseudo-points are placed in x, there are a total of JT of them – one for each process at each point in time that it is observed. Observation variance σ^2 is initialised to $\frac{1}{10}$.

4.4.1 Synthetic Data Experiments

This section comprises some basic checks on the performance of the various approximate inference algorithms in a situation where the ground truth is known.

Minature Data Set, No Learning This first synthetic experiment utilises a total of 1500 observations, 100 at each of 15 time points. Spatial inputs are chosen randomly. In this experiment, the data is sampled from the prior described above, and the model parameters (length scales and observation variance) are kept fixed. This is to isolate the variational inference from learning the model parameters.

Five approaches to approximate inference are investigated:

- *Mean-Field*: this is included as a baseline.
- *Exact-Latent-given-Bases* (Exact-LGB): this approximation utilises $q(f, x \mid h) := p(f, x, \mathbf{y} \mid h)$.
- *DPO-Latent-given-Bases* (DPO-LGB): same as Exact-Latent-given-Bases, but the exact posterior is replaced with a DPO approximation.
- *Exact-Bases-given-Latent* (Exact-BGL: this approximations utilises $q(f, h \mid x) := p(f, h, \mathbf{y} \mid x)$.
- *DPO-Bases-given-Latent* (DPO-BGL): same as Exact-Bases-given-Latent, but the exact posterior is replaced with a DPO approximation.

The Exact-LGB and Exact-BGL approximations are only possible in this example because of the small size of the data set. This makes it possible to isolate the quality of $q(f, x \mid h)$ and $q(f, h \mid x)$ relative to the best possible choice – the exact posterior.

15000 steps of ADAM were performed in each method, and Fig. 4.3 indicates that convergence has been achieved for all approximate inference algotithms. As expected *Mean-Field* attains the lowest ELBO, and the various coupled methods all perform better. In particular Exact-LGB and DPO-LGB appear to perform very similarly, while there is a small gap in performance between Exact-BGL and DPO-BGL. The former suggests that the DPO approximation is providing a good approximation to the exact conditional distribution over $f, x \mid h, \mathbf{y}$, while the latter approximation is less accurate relative to the exact distribution $f, h \mid x, \mathbf{y}$.

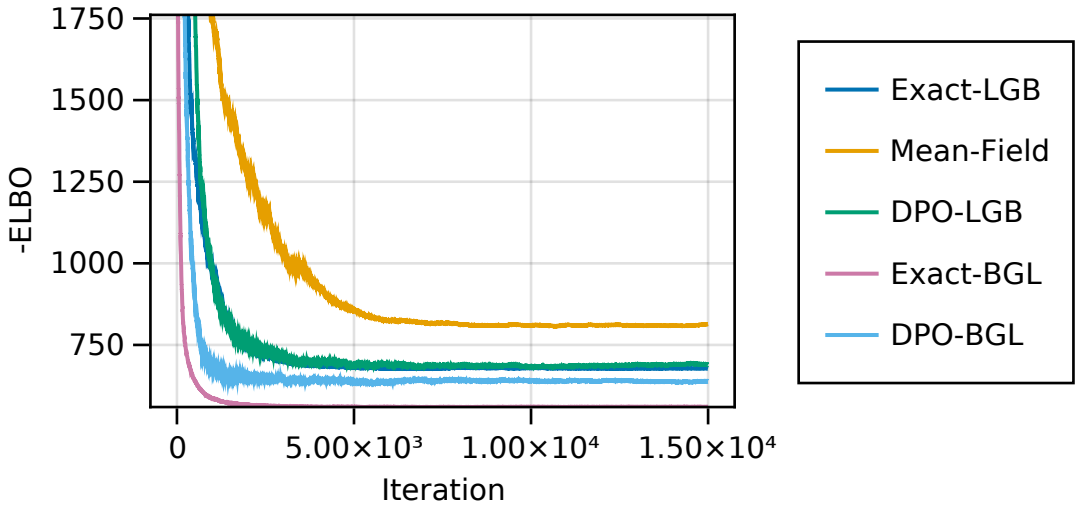


Fig. 4.3 Learning curves for variational inference with *no* parameter learning. Convergence is attained in all cases well before 15000 iterations have occured. *Mean-Field* takes the longest to converge and has the lowest ELBO at convergence.

Fig. 4.4 depicts the samples drawn from h in order to generate the data, and Fig. 4.5 shows the posterior mean of the bases inferred by a subset of the approximate inference algorithms: *Mean-Field*, DPO-BGL, and DPO-LGB. In all cases, approximate inference appears to have done a good job of recovering the sample from h via its posterior mean, up to a change of sign. It is partcularly noteworthy that the third basis has been recovered, as this has the highest-frequency component, presumably being the hardest to recover.

Minature Data Set, With Learning This second synthetic experiment utilises precisely the same set up as the previous, but optimises both the variational parameters and model parameters. Fig. 4.6 shows the learning curves in this case. The qualitative features are the same as before, albeit all algorithms take longer to converge.

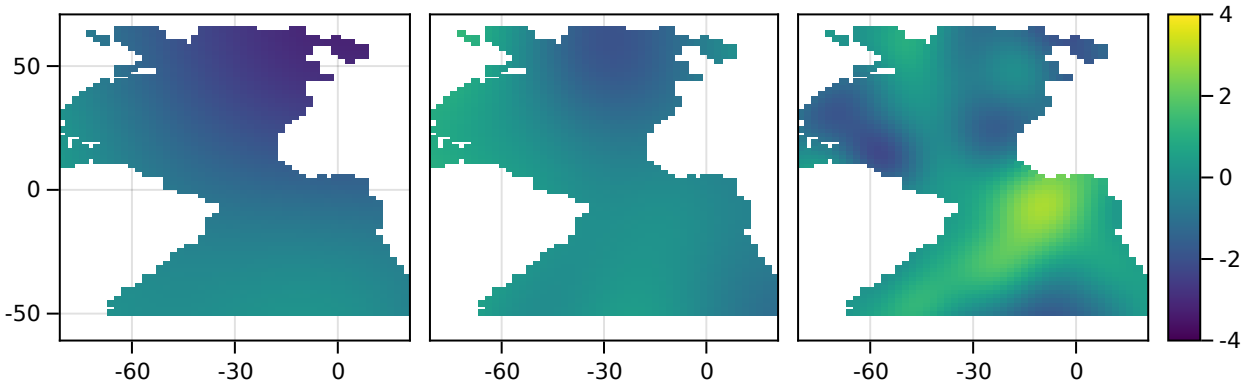


Fig. 4.4 The true unobserved bases sampled from the prior.

Similar results are obtained for the bases as in the previous experiment. That the bases are accurately recovered when the model parameters are learned provides some confidence that approximate inference is working reasonably well.

4.4.2 HadIOD

The following experiments were conducted in order to demonstrate that some basic qualitative properties of the climate are recovered by performing approximate inference in the model. A subset of 10^6 data were chosen uniformly at random without replacement from the HadIOD dataset between the years 2000 to 2010, and grouped by month. A single approximate inference scheme was utilised, the DPO-LGB. This is because in practice it seems to be substantially faster to execute, and this experiment is already pushing the limits of the scalability of the algorithm. For example, with the subset of HadIOD utilised and $J = 3$, roughly 2×10^5 iterations were required to attain convergence, taking roughly 12 hours to run. These scalability issues are discussed further in the next section.

Fig. 4.7 shows the progress of training. The ELBO appears to converge after around 10^5 iterations, although owing to the non-convexity of the problem it is not clear whether this is the global minimum.

Figure Fig. 4.8 again shows the posterior mean of h, while Fig. 4.9 shows samples from the posterior distribution over x. They suggest that the first two components of the model account for slowly varying high-level features of variability in the climate over time, while the third component appears to account for annual periodicity. It is interesting that the approximate posterior over the latents is as concentrated as it is. There are two possible explanations for this – either the exact posterior is highly concentrated, and the concentration

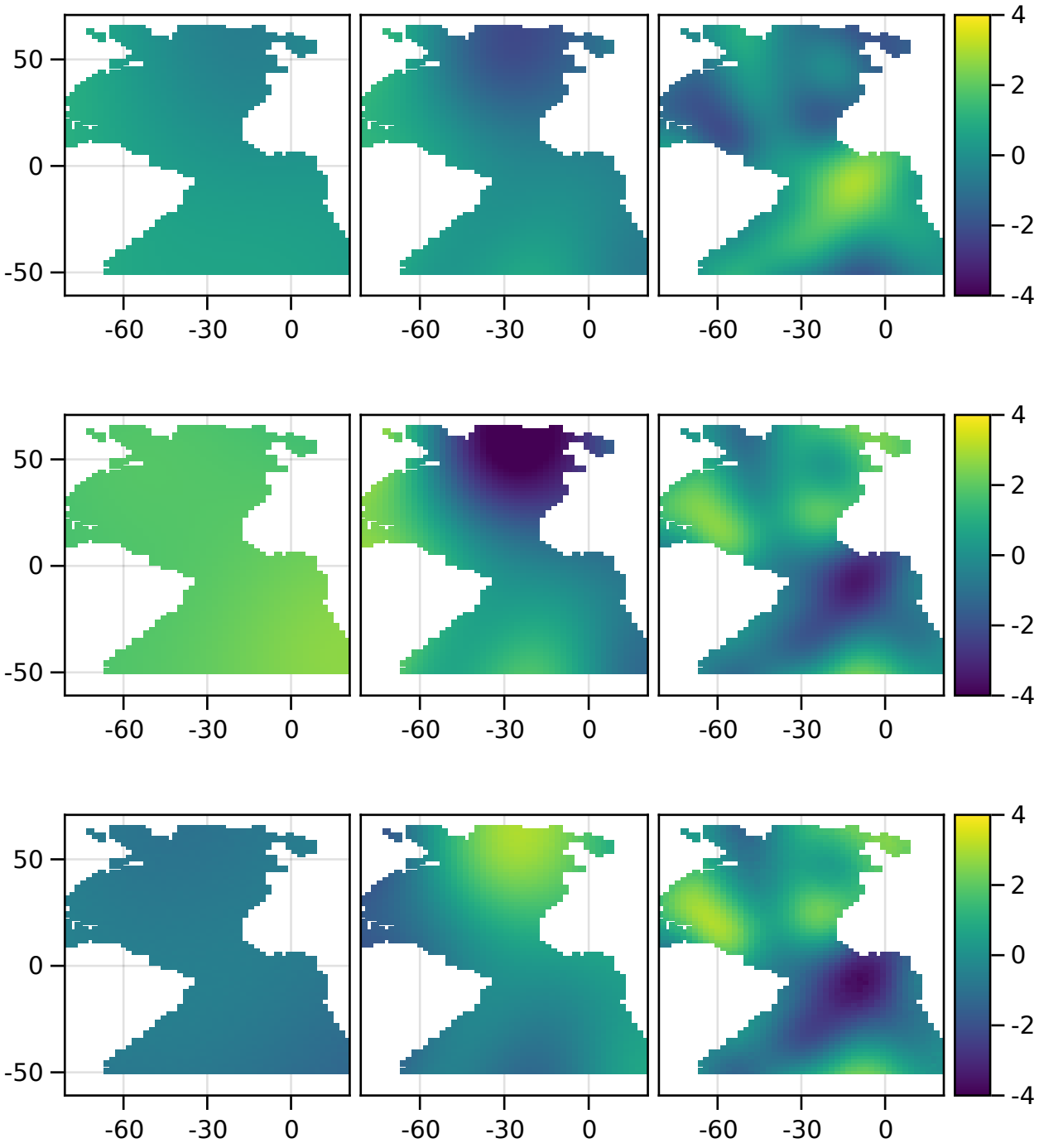


Fig. 4.5 Bases inferred using approximate inference. There is some variation in the first basis, but the structured recovered is broadly the same for all of them, up to a change of sign. Top: *Mean-Field*. Middle: DPO-LGB. Bottom: DPO-BGL. Similar results are obtained for Exact-LGB and Exact-BGL as for DPO-LGB and DPO-BGL respectively.

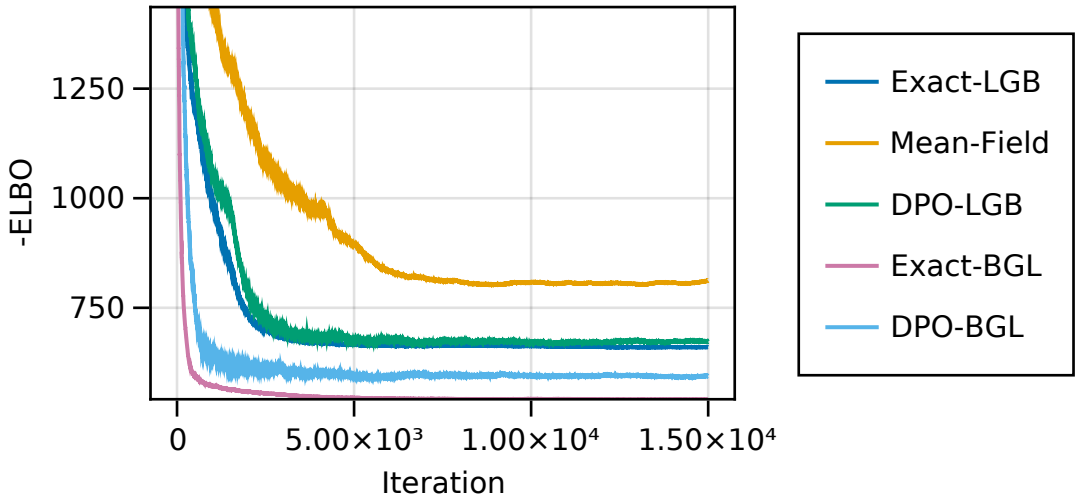


Fig. 4.6 Learning curves for variational inference *and* parameter learning. Convergence is attained in all cases well before 15000 iterations have occured. *Mean-Field* again takes longer than before to converge, and has the lowest ELBO at convergence. All methods take longer to converge than when the model parameters are fixed, as expected.

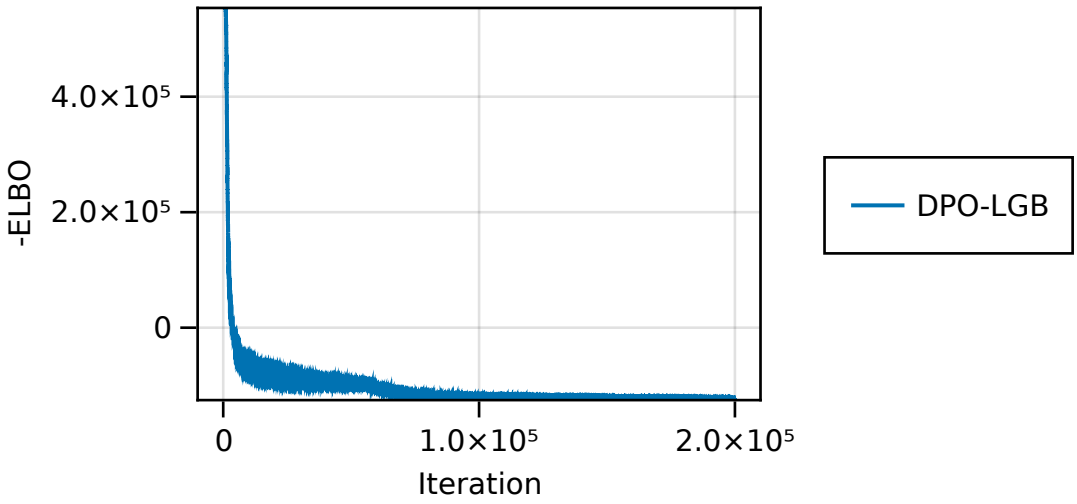


Fig. 4.7 DPO-LGB learning curve.

of the latents under the approximate posterior accurately reflects this, or the approximate posterior is overly concentrated as a conseqeuence of some types of approximation error. Given that there are 10^6 observations utilised, it seems quite plausible that the exact posterior would be this concentrated, although more investigation is required to know for sure.

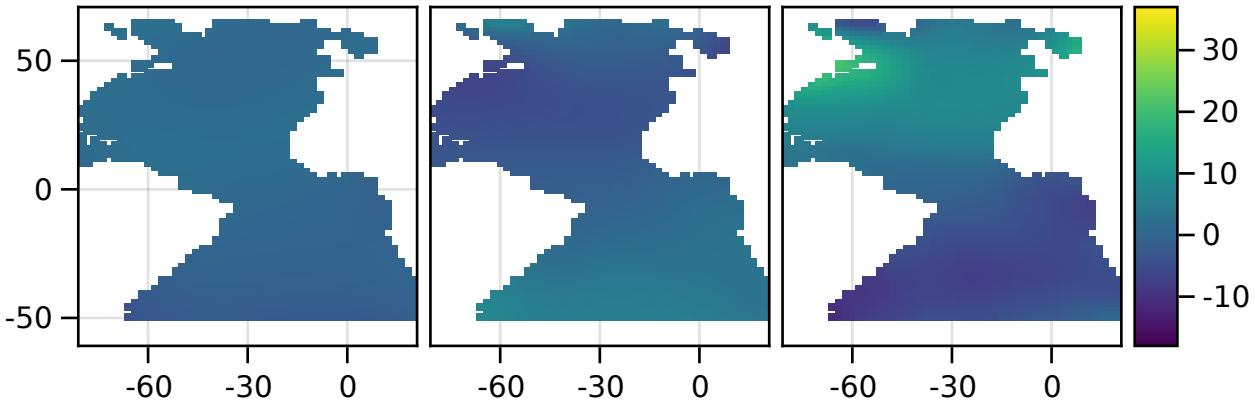


Fig. 4.8 Posterior mean over h. Scale is in degrees celsius.

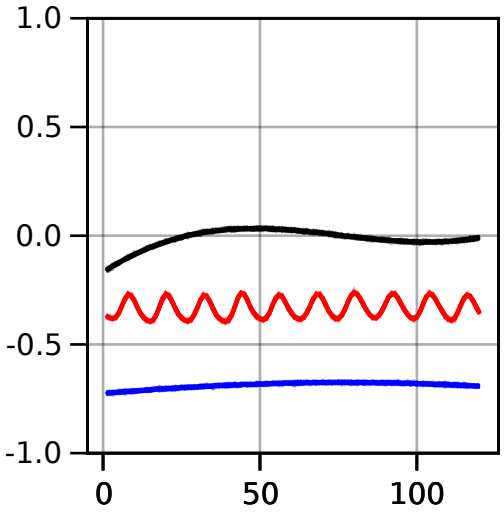


Fig. 4.9 Samples from the posterior over x. $j = (1, 2, 3)$ correspond to black, blue, and red respectively. Note the periodicity in the x_3, and the degree to which the posterior is concentrated.

Fig. 4.10 gives a sense of the extent to which the model has picked up on annual periodicities and longer-term fluctuations in climate, in addition to how well it is able to reproduce some areally-averaged temperatures. For example, the time series associated with regions 1 and 5 have annual periodicities, but are entirely out of phase with one another. This is to be expected, as the regions 1 and 5 reside in opposite hemispheres. Regions 2 and 4 show a similar pattern, although the amplitudes of their peiodicities are smaller. Their average temperature is also higher, which is also to be expected as they reside in the tropics.

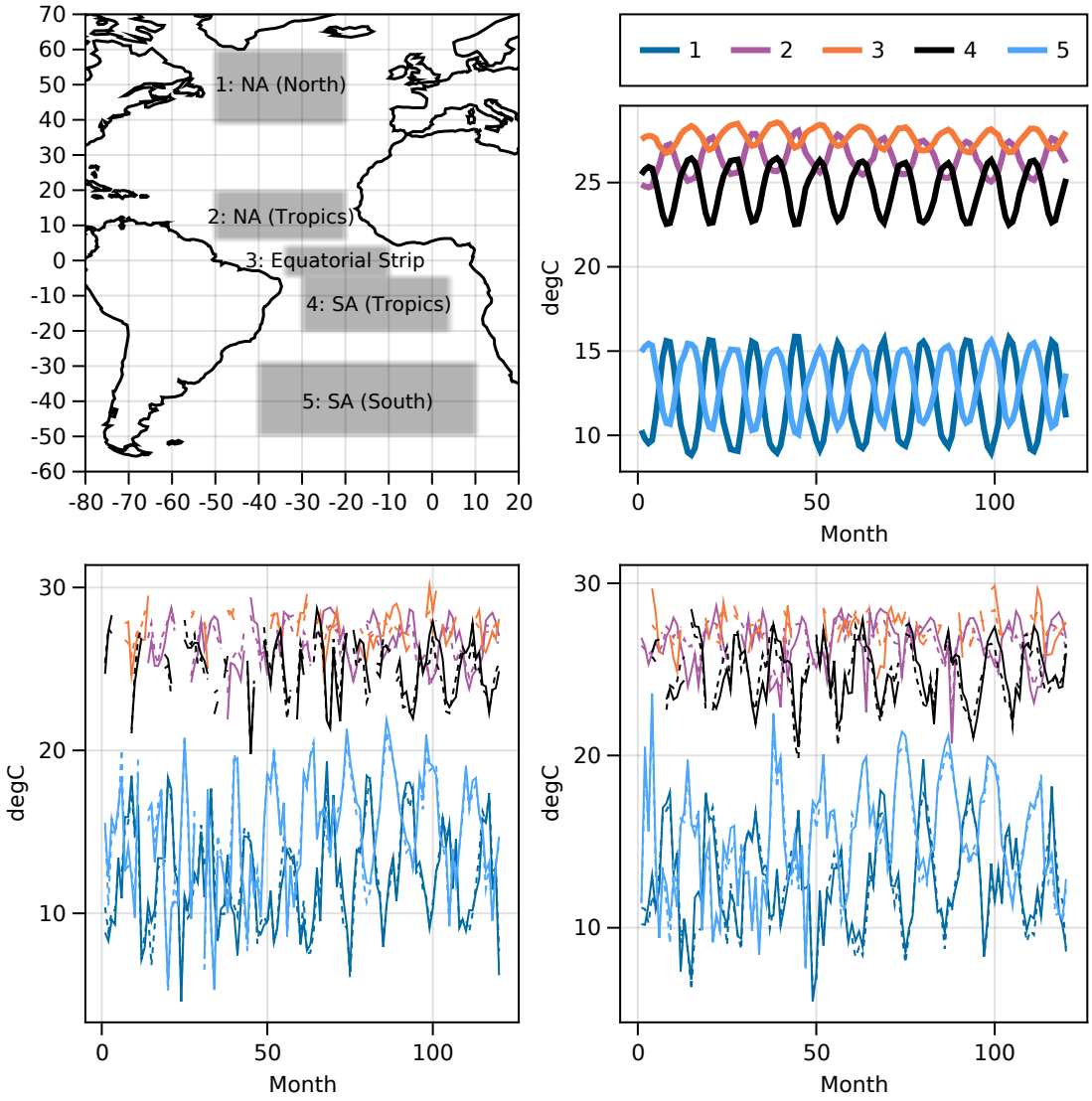


Fig. 4.10 Time series of performance of the model in 5 regions. Top left: the five regions of the Atlantic considered. Top right: Spatially-averaged prediction in each region. Bottom left: average prediction at training data (solid line) and observed mean of training data (dashed line) in each region in each month. Bottom right: same as bottom left, but for test data.

Fig. 4.10 also gives a sense of the model's ability to reproduce the average temperature in a given region, when compared to real data. This is shown in the bottom half of the figure – the graphs are noisier due to sampling variability, but they tell a clear story. The model's central belief about regionally averaged temperature is broadly consistent with what is observed.

4.5 Conclusion

This chapter has developed a probabilistic model – the *Infinite Linear Mixing Model* – which removes some of the limitations of existing statistical models utilised for decadal prediction, while retaining the central assumption that the observed data can be explained by a time-varying linear combination of spatial basis functions. However, approximate inference in this model is challenging, particularly at the scale required in order to fully utilise the available oceanographic data. To this end, a *Decoupled Pseudo-Observation* approximation was developed which goes part of the way to providing an accurate and scalable approximate inference algorithm for this model. In particular, it enables the use of an estimator requiring only mini-batches of observations in order to perform stochastic gradient descent in the variational and model parameters, while retaining strong dependencies between x and h. A collection of simple experiments were conducted on synthetic and real-world sea surface temperature data, to qualitatively validate the approach to approximate inference introduced. The results suggest that the model is able to pick up on some important high-level structure in the climate, and reproduce spatially-averaged temperatures well.

While this approximation is scalable in terms of the total number of observations, it does not currently scale well to time periods longer than around ten years, and takes a long time to train on even ten years. In order to test the proposed model on HadIOD in a sufficiently rigorous manner for decadal prediction, it will need to be trained on at least a hundred years of data, as much less than this will make it hard to pick up on sufficiently long-term signals in the data. Consequently, given that the model appears promising, these scalability issues should be addressed. It may be sufficient simply to both utilise state-space approximations to perform inference in x and to make use of hardware acceleration – both of these will reduce the time taken per step of gradient descent in the ELBO. Once these scalability problems have been addressed, it will be possible to utilise the model to perform hindcast experiments, and compare the predictions made by this model to those of competing approaches.

Chapter 5

Discussion

This thesis presents some new approaches for scaling up GPs to large spatio-temporal settings, and a new approach to the design of software to support these. Each of the preceding chapters provided a localised summary, so I take the opportunity below to step back and take a broader view of the themes tackled in this thesis, and what I believe the large open problems to be.

The Role of Software in Machine Learning and Probabilistic Modelling As alluded to in Chapter 1, most research in machine learning and probabilistic modelling necessarily focuses on methodological development. However, I do not believe it to be controversial to assert that composable software abstractions are a vitally-important factor in the successful adoption of new techniques. For example, it is hard to imagine how Deep Learning would have reached its present level of success without the array of composable tools available to researchers today.[1] For example, without algorithmic differentiation it would take a much greater deal of time for a Deep Learning research to go from idea to implementation. Similarly, domain experts often need to tweak architectures to suit their particular needs, which would be extremely difficult for most to achieve if hand-coded code for differentiating their network were required.

It is this type of reasoning that motivated Chapter 1 – a desire to narrow the gap between what ought to be easily achievable in practice and what actually is. That most GPs are built from affine transformations, and that one can in principle condition on observations, perform inference, or place pseudo-points in any part of a model, is well understood by GP researchers. However, it has remained somewhat difficult to utilise all of these facts in

[1] This is of course not the only reason for the widespread adoption of Deep Learning – it is also happens to be able to solve a variety of important problems for which no good solutions existed a decade ago.

practice, despite their conceptual simplicity. I believe that `Stheno.jl` does indeed narrow this gap somewhat, although there are plenty of practical (improving the robustness and performance of the software) and algorithmic problems left to solve.

GPs in Spatio-Temporal Models There remains a large amount of work to do in order to enable the routine use of GPs in large messy real-world spatio-temporal problems. While it is the case that approximations exist for simple large-scale spatio-temporal problems, with domains comprising at most 2 or 3 dimensions and certain choices of kernel, they do not scale to high-dimensional input spaces, or compose well in a hierarchical model. For example, the R-INLA software (Lindgren and Rue, 2015) works very well for a very restricted set of problems, but is not easy to compose with other tools or embed inside another model.

Conversely, pseudo-point approximations alone do not scale to truly large spatio-temporal settings. The methodology developed in Chapter 3 allows them to scale to problems involving a large number of time points, and high-dimensional "space", but it simply doesn't extend easily to problems involving a large low-dimensional spatial domain, in which many pseudo-points would be required to properly cover the spatial domain. For example, the models to which this approximation applies could not easily be scaled to produce a global model for temperature with high spatial resolution – some additional approximation is required. Furthermore, there is often information available at multiple scales, meaning that entirely different approximations may be best for different components of the model.

A large amount of work goes into scaling up existing GP models. This line of work is based on the premise that the existing models are adequate, and it is our ability to perform approximate inference in them that is lacking. While this may be the case, it might also be that we need to consider more restricted model classes, such as was done in Chapter 4. Doing this could enable a trade off to be made between flexibility and scalability, that could be well-suited to a wide array of problems. It might even be the case that a class of GP-based models exists which is both better suited to a wide range of problems, and more scalable, than e.g., simple stationary GPs.

Supposing that it turns out to be the case that some combination of existing types of approximations and restrictions on the class of models considered are able to solve these scalability problems, it will then be necessary produce software which enables domain experts to easily combine these approximations and models in just the right way for their particular problem, if they are to be widely adopted. It will be necessary to abstract away enough detail, and provide a good enough mixture of practical guidelines and automation, to make it possible for domain experts to know how to combine approximations in the correct way, with minimal

involvement from GP researchers. What this software will look like is not possible to say currently, as the abstractions that will need to be developed will depend strongly upon what solutions turn out to be important.

References

Adam, V. (2017). Structured variational inference for coupled Gaussian processes. *Workshop on Advances in Approximate Bayesian Inference*.

Adam, V., Chang, P., Khan, M. E. E., and Solin, A. (2021). Dual parameterization of sparse variational gaussian processes. *Advances in Neural Information Processing Systems*, 34.

Adam, V., Eleftheriadis, S., Artemev, A., Durrande, N., and Hensman, J. (2020). Doubly sparse variational Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 2874–2884. PMLR.

Adams, R. P., Murray, I., and MacKay, D. J. (2008). The gaussian process density sampler. In *NIPS*, pages 9–16.

Alvarez, M. A. and Lawrence, N. D. (2008). Sparse convolved gaussian processes for multi-output regression. In *NIPS*, volume 21, pages 57–64.

Alvarez, M. A., Rosasco, L., and Lawrence, N. D. (2011). Kernels for vector-valued functions: A review. *arXiv preprint arXiv:1106.6251*.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276.

Archambeau, C., Cornford, D., Opper, M., and Shawe-Taylor, J. (2007). Gaussian process approximations of stochastic differential equations. In *Gaussian Processes in Practice*, pages 1–16. PMLR.

Artemev, A., Burt, D. R., and van der Wilk, M. (2021). Tighter bounds on the log marginal likelihood of gaussian process regression using conjugate gradients. *arXiv preprint arXiv:2102.08314*.

Ashman, M., So, J., Tebbutt, W., Fortuin, V., Pearce, M., and Turner, R. E. (2020). Sparse Gaussian process variational autoencoders. *arXiv preprint arXiv:2010.10177*.

Atkinson, C. P., Rayner, N. A., Kennedy, J. J., and Good, S. A. (2014). An integrated database of ocean temperature and salinity observations. *Journal of Geophysical Research: Oceans*, 119(10):7139–7163.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–153.

Betancourt, M. (2017). A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*.

Bezanson, J., Karpinski, S., Shah, V. B., and Edelman, A. (2012). Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*.

Boer, G. J., Smith, D. M., Cassou, C., Doblas-Reyes, F., Danabasoglu, G., Kirtman, B., Kushnir, Y., Kimoto, M., Meehl, G. A., Msadek, R., et al. (2016). The decadal climate prediction project (dcpp) contribution to cmip6. *Geoscientific Model Development*, 9(10):3751–3777.

Borovitskiy, V., Azangulov, I., Terenin, A., Mostowsky, P., Deisenroth, M., and Durrande, N. (2021). Matérn gaussian processes on graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 2593–2601. PMLR.

Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth (he/him), M. (2020). Matérn gaussian processes on riemannian manifolds. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12426–12437. Curran Associates, Inc.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., and Wanderman-Milne, S. (2018). JAX: composable transformations of Python+NumPy programs.

Brönnimann, S., Allan, R., Atkinson, C., Buizza, R., Bulygina, O., Dahlgren, P., Dee, D., Dunn, R., Gomes, P., John, V. O., et al. (2018). Observations for reanalyses. *Bulletin of the American Meteorological Society*, 99(9):1851–1866.

Bruinsma, W., Perim, E., Tebbutt, W., Hosking, S., Solin, A., and Turner, R. (2020). Scalable exact inference in multi-output gaussian processes. In *International Conference on Machine Learning*, pages 1190–1201. PMLR.

Bruinsma, W. P., Tegnér, M., and Turner, R. E. (2022). Modelling non-smooth signals with complex spectral structure. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research. PMLR.

Bui, T. D. and Turner, R. E. (2014). Tree-structured Gaussian process approximations. In *Advances in Neural Information Processing Systems 27*, pages 2213–2221. Curran Associates, Inc.

Bui, T. D., Yan, J., and Turner, R. E. (2017). A unifying framework for Gaussian process pseudo-point approximations using power expectation propagation. *Journal of Machine Learning Research*, 18(1):3649–3720.

Bunker, J. and Turner, R. E. (2019). Extending and applying the gaussian process autoregressive regression model.

Burt, D., Rasmussen, C. E., and van der Wilk, M. (2019). Rates of convergence for sparse variational Gaussian process regression. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 862–871. PMLR.

Burt, D. R., Artemev, A., and van der Wilk, M. (2021). Barely biased learning for gaussian process regression. *arXiv preprint arXiv:2109.09417*.

Burt, D. R., Rasmussen, C. E., and van der Wilk, M. (2020a). Convergence of sparse variational inference in gaussian processes regression. *Journal of Machine Learning Research*, 21:1–63.

Burt, D. R., Rasmussen, C. E., and van der Wilk, M. (2020b). Variational orthogonal features. *arXiv preprint arXiv:2006.13170*.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. (2016). Manifold Gaussian processes for regression. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 3338–3345. IEEE.

Camden, O. D. (2015). National Statistics Postcode Lookup UK Coordinates. `https://opendata.camden.gov.uk/Maps/National-Statistics-Postcode-Lookup-UK-Coordinates/77ra-mbbn`. [Online; accessed January-2021].

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1).

Chang, P. E., Wilkinson, W. J., Khan, M. E., and Solin, A. (2020). Fast variational learning in state-space Gaussian process models. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

Chelton, D. B. and Risien, C. M. (2016). Zonal and meridional discontinuities and other issues with the hadisst1. 1 dataset.

Chen, J. and Revels, J. (2016). Robust benchmarking in noisy environments. *arXiv e-prints*.

Csató, L. and Opper, M. (2002). Sparse On-Line Gaussian Processes. *Neural computation*, 14(3):641–668.

Cusumano-Towner, M. F., Saad, F. A., Lew, A. K., and Mansinghka, V. K. (2019). Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th acm sigplan conference on programming language design and implementation*, pages 221–236.

Dahl, A. and Bonilla, E. V. (2019). Grouped gaussian processes for solar power prediction. *Machine Learning*, 108(8):1287–1306.

Damianou, A. and Lawrence, N. D. (2013). Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR.

Damianou, A. C., Titsias, M. K., and Lawrence, N. (2016). Variational inference for latent variables and uncertain inputs in gaussian processes.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer.

Delworth, T., Manabe, S., and Stouffer, R. J. (1993). Interdecadal variations of the thermohaline circulation in a coupled ocean-atmosphere model. *Journal of Climate*, 6(11):1993–2011.

Delworth, T. L. and Mann, M. E. (2000). Observed and simulated multidecadal variability in the northern hemisphere. *Climate Dynamics*, 16(9):661–676.

Deser, C., Phillips, A. S., and Hurrell, J. W. (2004). Pacific interdecadal climate variability: Linkages between the tropics and the north pacific during boreal winter since 1900. *Journal of Climate*, 17(16):3109–3124.

Doucet, A. (2010). A note on efficient conditional simulation of Gaussian distributions. *Departments of Computer Science and Statistics, University of British Columbia*, 4.

Duffin, C., Cripps, E., Stemler, T., and Girolami, M. (2021). Statistical finite elements for misspecified models. *Proceedings of the National Academy of Sciences*, 118(2).

Duncker, L., Bohner, G., Boussard, J., and Sahani, M. (2019). Learning interpretable continuous-time models of latent stochastic dynamical systems. In *International Conference on Machine Learning*, pages 1726–1734. PMLR.

Dutordoir, V., Durrande, N., and Hensman, J. (2020). Sparse gaussian processes with spherical harmonic features. In *International Conference on Machine Learning*, pages 2793–2802. PMLR.

Duvenaud, D., Lloyd, J., Grosse, R., Tenenbaum, J., and Zoubin, G. (2013). Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174. PMLR.

Duvenaud, D., Nickisch, H., and Rasmussen, C. E. (2011). Additive Gaussian Processes. *Advances in Neural Information Processing Systems*, 24:226–234.

Duvenaud, D., Rippel, O., Adams, R., and Ghahramani, Z. (2014). Avoiding pathologies in very deep networks. In *Artificial Intelligence and Statistics*, pages 202–210. PMLR.

Fairbrother, J., Nemeth, C., Rischard, M., Brea, J., and Pinder, T. (2021). Gaussianprocesses. jl: A nonparametric bayes package for the julia language. *Journal of Statistical Software (to appear)*.

Fiedler, T., Pitman, A. J., Mackenzie, K., Wood, N., Jakob, C., and Perkins-Kirkpatrick, S. E. (2021). Business risk and the emergence of climate analytics. *Nature Climate Change*, pages 1–8.

Foland, C., Palmer, T., and Parker, D. (1986). Sahel rainfall and worldwide sea temperatures. *Nature*, 320:602–607.

Folland, C., Owen, J., Ward, M. N., and Colman, A. (1991). Prediction of seasonal rainfall in the sahel region using empirical and dynamical methods. *Journal of forecasting*, 10(1-2):21–56.

Folland, C. K. and Parker, D. (1995). Correction of instrumental biases in historical sea surface temperature data. *Quarterly Journal of the Royal Meteorological Society*, 121(522):319–367.

Foster, D., Comeau, D., and Urban, N. M. (2020). A bayesian approach to regional decadal predictability: Sparse parameter estimation in high-dimensional linear inverse models of high-latitude sea surface temperature variability. *Journal of Climate*, 33(14):6065–6081.

Freeman, E., Woodruff, S. D., Worley, S. J., Lubker, S. J., Kent, E. C., Angel, W. E., Berry, D. I., Brohan, P., Eastman, R., Gates, L., et al. (2017). Icoads release 3.0: a major update to the historical marine climate record. *International Journal of Climatology*, 37(5):2211–2232.

Frigola, R., Chen, Y., and Rasmussen, C. E. (2014). Variational gaussian process state-space models. In *Advances in neural information processing systems*, pages 3680–3688.

Gardner, J., Pleiss, G., Wu, R., Weinberger, K., and Wilson, A. (2018a). Product Kernel Interpolation for Scalable Gaussian Processes. In *International Conference on Artificial Intelligence and Statistics*, pages 1407–1416. PMLR.

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. (2018b). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*.

Ge, H., Xu, K., and Ghahramani, Z. (2018). Turing: Composable inference for probabilistic programming. In *International Conference on Artificial Intelligence and Statistics*, pages 1682–1690.

Gelfand, A. E., Sahu, S. K., and Carlin, B. P. (1995). Efficient parametrisations for normal linear mixed models. *Biometrika*, pages 479–488.

Ghahramani, Z. and Rasmussen, C. E. (2003). Bayesian monte carlo. In *Advances in neural information processing systems*, pages 505–512.

Gibbs, M. and MacKay, D. J. (1997). Efficient implementation of gaussian processes.

Gilboa, E., Saatçi, Y., and Cunningham, J. (2013). Scaling multidimensional gaussian processes using projected additive approximations. In *International Conference on Machine Learning*, pages 454–461. PMLR.

Girolami, M. and Calderhead, B. (2011). Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214.

Good, S. A., Martin, M. J., and Rayner, N. A. (2013). En4: Quality controlled ocean temperature and salinity profiles and monthly objective analyses with uncertainty estimates. *Journal of Geophysical Research: Oceans*, 118(12):6704–6716.

Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. (2012). Church: a language for generative models. *arXiv preprint arXiv:1206.3255*.

Goovaerts, P. (1997). *Geostatistics for Natural Resources Evaluation*. Oxford University Press, 1 edition.

Gorinova, M., Moore, D., and Hoffman, M. (2020). Automatic reparameterisation of probabilistic programs. In *International Conference on Machine Learning*, pages 3648–3657. PMLR.

GPy (since 2012). GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

Grigorievskiy, A., Lawrence, N., and Särkkä, S. (2017). Parallelizable sparse inverse formulation Gaussian processes (SpInGP). In *International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

Ham, Y.-G., Kim, J.-H., and Luo, J.-J. (2019). Deep learning for multi-year enso forecasts. *Nature*, 573(7775):568–572.

Hamelijnck, O., Wilkinson, W., Loppi, N., Solin, A., and Damoulas, T. (2021). Spatio-temporal variational Gaussian processes. *Advances in Neural Information Processing Systems*, 34.

Hartikainen, J., Riihimäki, J., and Särkkä, S. (2011). Sparse spatio-temporal Gaussian processes with general likelihoods. In *International Conference on Artificial Neural Networks*, pages 193–200. Springer.

Hawkins, E., Robson, J., Sutton, R., Smith, D., and Keenlyside, N. (2011). Evaluating the potential for statistical decadal predictions of sea surface temperatures with a perfect model approach. *Climate dynamics*, 37(11):2495–2509.

Hawkins, E. and Sutton, R. (2009a). Decadal predictability of the atlantic ocean in a coupled gcm: Forecast skill and optimal perturbations using linear inverse modeling. *Journal of Climate*, 22(14):3960–3978.

Hawkins, E. and Sutton, R. (2009b). The potential to narrow uncertainty in regional climate predictions. *Bulletin of the American Meteorological Society*, 90(8):1095–1108.

Hensman, J., Durrande, N., and Solin, A. (2017). Variational fourier features for Gaussian processes. *The Journal of Machine Learning Research*, 18(1):5537–5588.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 282–290. AUAI Press.

Hensman, J., Matthews, A., and Ghahramani, Z. (2015). Scalable variational Gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360.

Hernández-Lobato, J. M., Requeima, J., Pyzer-Knapp, E. O., and Aspuru-Guzik, A. (2017). Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space. In *International conference on machine learning*, pages 1470–1479. PMLR.

HM Land Registry (2014). Price Paid Data. `https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads`. [Online; accessed January-2021].

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623.

Hoffman, Y. and Ribak, E. (1991). Constrained realizations of Gaussian fields-a simple algorithm. *The Astrophysical Journal*, 380:L5–L8.

Huddart, B., Subramanian, A., Zanna, L., and Palmer, T. (2017). Seasonal and decadal forecasts of atlantic sea surface temperatures using a linear inverse model. *Climate Dynamics*, 49(5):1833–1845.

Ialongo, A. D., Van Der Wilk, M., Hensman, J., and Rasmussen, C. E. (2019). Overcoming mean-field approximations in recurrent gaussian process models. In *International Conference on Machine Learning*, pages 2931–2940. PMLR.

Ialongo, A. D., van der Wilk, M., and Rasmussen, C. E. (2018). Closed-form inference and prediction in gaussian process state-space models. *arXiv preprint arXiv:1812.03580*.

Ingleby, B. and Huddleston, M. (2007). Quality control of ocean temperature and salinity profiles—historical and real-time data. *Journal of Marine Systems*, 65(1-4):158–175.

Innes, M. (2018). Don't unroll adjoint: Differentiating ssa-form programs. *CoRR*, abs/1810.07951.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45.

Kaplan, A., Kushnir, Y., Cane, M. A., and Blumenthal, M. B. (1997). Reduced space optimal analysis for historical data sets: 136 years of atlantic sea surface temperatures. *Journal of Geophysical Research: Oceans*, 102(C13):27835–27860.

Karpinski, S. (2019). The unreasonable effectiveness of multiple dispatch. `https://www.youtube.com/watch?v=kc9HwsxE1OY`. Accessed: 2022-01-22.

Kennedy, J., Rayner, N., Smith, R., Parker, D., and Saunby, M. (2011). Reassessing biases and other uncertainties in sea surface temperature observations measured in situ since 1850: 2. biases and homogenization. *Journal of Geophysical Research: Atmospheres*, 116(D14).

Kennedy, J. J., Rayner, N., Atkinson, C., and Killick, R. (2019). An ensemble data set of sea surface temperature change from 1850: The met office hadley centre hadsst. 4.0. 0.0 data set. *Journal of Geophysical Research: Atmospheres*, 124(14):7719–7763.

Khan, M. and Lin, W. (2017). Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models. In *Artificial Intelligence and Statistics*, pages 878–887.

Khan, M. E. and Nielsen, D. (2018). Fast yet simple natural-gradient descent for variational inference in complex models. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*, pages 31–35. IEEE.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kirtman, B., Shukla, J., Balmaseda, M., Graham, N., Penland, C., Xue, Y., and Zebiak, S. (2002). Current status of enso forecast skill. *A report to the Climate Variability and Predictability (CLIVAR) Numerical Experimentation Group (NEG), CLIVAR Working Group on Seasonal to Interannual Prediction*.

Kok, M. and Solin, A. (2018). Scalable magnetic field slam in 3d using gaussian process maps. In *2018 21st international conference on information fusion (FUSION)*, pages 1353–1360. IEEE.

Kucukelbir, A., Ranganath, R., Gelman, A., and Blei, D. (2015). Automatic variational inference in stan. In *Advances in neural information processing systems*, pages 568–576.

Kushnir, Y. (1994). Interdecadal variations in north atlantic sea surface temperature and associated atmospheric conditions. *Journal of Climate*, 7(1):141–157.

Lalchand, V. and Rasmussen, C. E. (2020). Approximate inference for fully bayesian gaussian process regression. In *Symposium on Advances in Approximate Bayesian Inference*, pages 1–12. PMLR.

Lawrence, N. and Hyvärinen, A. (2005). Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(11).

Lawrence, N. D. (2003). Gaussian process latent variable models for visualisation of high dimensional data. In *Nips*, volume 2, page 5. Citeseer.

Lazaro-Gredilla, M. and Figueiras-Vidal, A. (2009). Inter-domain Gaussian processes for sparse inference using inducing features. In *Advances in Neural Information Processing Systems*, pages 1087–1095.

Lindgren, F. and Rue, H. (2015). Bayesian spatial modelling with r-inla. *Journal of statistical software*, 63:1–25.

Lindgren, F., Rue, H., and Lindström, J. (2011). An Explicit Link Between Gaussian Fields and Gaussian Markov Random Fields: The Stochastic Partial Differential Equation Approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498.

Loper, J., Blei, D., Cunningham, J. P., and Paninski, L. (2020). General linear-time inference for Gaussian processes on one dimension. *arXiv preprint arXiv:2003.05554*.

Lunn, D., Spiegelhalter, D., Thomas, A., and Best, N. (2009). The bugs project: Evolution, critique and future directions. *Statistics in medicine*, 28(25):3049–3067.

MacKay, D. J. (1998). Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Mahsereci, M. and Hennig, P. (2017). Probabilistic line searches for stochastic optimization. *The Journal of Machine Learning Research*, 18(1):4262–4320.

Mantua, N. J. and Hare, S. R. (2002). The pacific decadal oscillation. *Journal of oceanography*, 58(1):35–44.

Mantua, N. J., Hare, S. R., Zhang, Y., Wallace, J. M., and Francis, R. C. (1997). A pacific interdecadal climate oscillation with impacts on salmon production. *Bulletin of the american Meteorological Society*, 78(6):1069–1080.

Matthews, A. G. d. G., Hensman, J., Turner, R. E., and Ghahramani, Z. (2016). On sparse variational methods and the Kullback-Leibler divergence between stochastic processes. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 231–239. PMLR.

Matthews, D. G., Alexander, G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). Gpflow: A Gaussian process library using TensorFlow. *The Journal of Machine Learning Research*, 18(1):1299–1304.

Meehl, G. A., Goddard, L., Boer, G., Burgman, R., Branstator, G., Cassou, C., Corti, S., Danabasoglu, G., Doblas-Reyes, F., Hawkins, E., et al. (2014). Decadal climate prediction: an update from the trenches. *Bulletin of the American Meteorological Society*, 95(2):243–267.

Meehl, G. A., Goddard, L., Murphy, J., Stouffer, R. J., Boer, G., Danabasoglu, G., Dixon, K., Giorgetta, M. A., Greene, A. M., Hawkins, E., et al. (2009). Decadal prediction: Can it be skillful? *Bulletin of the American Meteorological Society*, 90(10):1467–1486.

Menne, M. J., Durre, I., Vose, R. S., Gleason, B. E., and Houston, T. G. (2012). An overview of the global historical climatology network-daily database. *Journal of Atmospheric and Oceanic Technology*, 29(7):897–910.

Minka, T. (2004). Power EP. Technical report, Technical report, Microsoft Research, Cambridge.

Mogensen, P. K. and Riseth, A. N. (2018). Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

NASA-JPL (2020). NASADEM Merged DEM Global 1 arc second V001 [Data set]. NASA EOSDIS Land Processes DAAC.

Neal, R. M. et al. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2.

Newman, M. (2007). Interannual to decadal predictability of tropical and north pacific sea surface temperatures. *Journal of climate*, 20(11):2333–2356.

Newman, M. (2013). An empirical benchmark for decadal forecasts of global surface temperature anomalies. *Journal of Climate*, 26(14):5260–5269.

Nicholson, G., Blangiardo, M., Briers, M., Diggle, P. J., Fjelde, T. E., Ge, H., Goudie, R. J., Jersakova, R., King, R. E., Lehmann, B. C., et al. (2021). Interoperability of statistical models in pandemic preparedness: principles and reality. *arXiv preprint arXiv:2109.13730*.

Nocedal, J. and Wright, S. J. (1999). *Numerical optimization*. Springer.

Ober, S. W. and Aitchison, L. (2021). Global inducing point variational posteriors for bayesian neural networks and deep gaussian processes. In *International Conference on Machine Learning*, pages 8248–8259. PMLR.

O'Hagan, A. (1978). Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(1):1–24.

O'Hagan, A. (1987). Monte carlo is fundamentally unsound. *The Statistician*, pages 247–249.

O'Hagan, A. (1991). Bayes–hermite quadrature. *Journal of statistical planning and inference*, 29(3):245–260.

Opper, M. and Archambeau, C. (2009). The variational Gaussian approximation revisited. *Neural Computation*, 21(3):786–792.

Osborne, M. A., Garnett, R., and Roberts, S. J. (2009). Gaussian processes for global optimization. In *3rd international conference on learning and intelligent optimization (LION3)*, pages 1–15.

O'Hagan, A. (1998). A Markov property for covariance structures. *Statistics Research Report*, 98:13.

Panos, A., Dellaportas, P., and Titsias, M. K. (2018). Fully scalable Gaussian processes using subspace inducing inputs. *arXiv preprint arXiv:1807.02537*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Penland, C. (1989). Random forcing and forecasting using principal oscillation pattern analysis. *Monthly Weather Review*, 117(10):2165–2185.

Penland, C. and Magorian, T. (1993). Prediction of niño 3 sea surface temperatures using linear inverse modeling. *Journal of Climate*, 6(6):1067–1076.

Penland, C. and Sardeshmukh, P. D. (1995). The optimal growth of tropical sea surface temperature anomalies. *Journal of climate*, 8(8):1999–2024.

Perdikaris, P., Raissi, M., Damianou, A., Lawrence, N. D., and Karniadakis, G. E. (2017). Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2198):20160751.

Plummer, M. et al. (2003). Jags: A program for analysis of Bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, volume 124. Vienna, Austria.

Power, S., Casey, T., Folland, C., Colman, A., and Mehta, V. (1999). Inter-decadal modulation of the impact of enso on australia. *Climate Dynamics*, 15(5):319–324.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959.

Rasmussen, C. E. and Nickisch, H. (2010). Gaussian processes for machine learning (GPML) toolbox. *Journal of machine learning research*, 11(Nov):3011–3015.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Rauch, H. E., Striebel, C., and Tung, F. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450.

Rayner, N., Parker, D. E., Horton, E., Folland, C. K., Alexander, L. V., Rowell, D., Kent, E. C., and Kaplan, A. (2003). Global analyses of sea surface temperature, sea ice, and night marine air temperature since the late nineteenth century. *Journal of Geophysical Research: Atmospheres*, 108(D14).

Requeima, J., Tebbutt, W., Bruinsma, W., and Turner, R. E. (2019). The gaussian process autoregressive regression model (gpar). In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1860–1869. PMLR.

Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-mode automatic differentiation in julia. *arXiv:1607.07892 [cs.MS]*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Rodrigues, E., Zadrozny, B., Watson, C., and Gold, D. (2021). Decadal forecasts with resdmd: a residual dmd neural network. *arXiv preprint arXiv:2106.11111*.

Saatçi, Y. (2012). *Scalable Inference for Structured Gaussian Process Models*. PhD thesis, University of Cambridge.

Salimbeni, H., Dutordoir, V., Hensman, J., and Deisenroth, M. (2019). Deep gaussian processes with importance-weighted variational inference. In *International Conference on Machine Learning*, pages 5589–5598. PMLR.

Salimbeni, H., Eleftheriadis, S., and Hensman, J. (2018). Natural gradients in practice: Non-conjugate variational inference in Gaussian process models. *arXiv preprint arXiv:1803.09151*.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55.

Särkkä, S. and García-Fernández, Á. F. (2020). Temporal parallelization of Bayesian smoothers. *IEEE Transactions on Automatic Control*.

Särkkä, S. and Solin, A. (2019). *Applied Stochastic Differential Equations*. Cambridge University Press.

Särkkä, S., Solin, A., and Hartikainen, J. (2013). Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering. *IEEE Signal Processing Magazine*, 30(4):51–61.

Scherrer, C. and Zhao, T. (2020). Soss: Declarative probabilistic programming via runtime code generation. doi:10.5281/zenodo.5520061.

Schober, M., Särkkä, S., and Hennig, P. (2019). A probabilistic model for the numerical solution of initial value problems. *Statistics and Computing*, 29(1):99–122.

Seeger, M. (1999). Bayesian methods for support vector machines and gaussian processes. Technical report, Univeristy of Edinburgh.

Seeger, M., Williams, C., and Lawrence, N. (2003). Fast forward selection to speed up sparse Gaussian process regression. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics.

Shah, A., Wilson, A., and Ghahramani, Z. (2014). Student-t processes as alternatives to gaussian processes. In *Artificial intelligence and statistics*, pages 877–885. PMLR.

Simpson, F., Lalchand, V., and Rasmussen, C. E. (2020). Marginalised gaussian processes with nested sampling. *arXiv preprint arXiv:2010.16344*.

Snelson, E. and Ghahramani, Z. (2005). Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264. MIT Press.

Snelson, E. and Ghahramani, Z. (2012). Variable noise and dimensionality reduction for sparse Gaussian processes. *arXiv preprint arXiv:1206.6873*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Solin, A. (2016). *Stochastic differential equation methods for spatio-temporal Gaussian process regression*. PhD thesis, Aalto University.

Solin, A., Hensman, J., and Turner, R. E. (2018). Infinite-horizon Gaussian processes. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3490–3499.

Steinruecken, C., Smith, E., Janz, D., Lloyd, J., and Ghahramani, Z. (2019). The automatic statistician. In *Automated Machine Learning*, pages 161–173. Springer, Cham.

Tanaka, Y., Tanaka, T., Iwata, T., Kurashima, T., Okawa, M., Akagi, Y., and Toda, H. (2019). Spatially aggregated gaussian processes with multivariate areal outputs. *arXiv preprint arXiv:1907.08350*.

Thomas, A., Spiegelhalter, D. J., and Gilks, W. (1992). Bugs: A program to perform bayesian inference using gibbs sampling. *Bayesian statistics*, 4(9):837–842.

Titsias, M. and Lawrence, N. D. (2010). Bayesian gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 844–851. JMLR Workshop and Conference Proceedings.

Titsias, M. and Lázaro-Gredilla, M. (2014). Doubly stochastic variational bayes for non-conjugate inference. In *International Conference on Machine Learning*, pages 1971–1979.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 567–574. PMLR.

Tobar, F. (2019). Band-limited Gaussian processes: The sinc kernel. In *Advances in Neural Information Processing Systems*, pages 12749–12759. Curran Associates, Inc.

Tobar, F., Bui, T. D., and Turner, R. E. (2015). Learning stationary time series using gaussian processes with nonparametric kernels. *Advances in neural information processing systems*, 28.

Tolpin, D., van de Meent, J.-W., and Wood, F. (2015). Probabilistic programming in anglican. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 308–311. Springer.

Turner, R. E. (2010). *Statistical Models for Natural Sounds*. PhD thesis, UCL (University College London).

Turner, R. E. and Sahani, M. (2011). Demodulation as probabilistic inference. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(8):2398–2411.

van der Wilk, M., Rasmussen, C. E., and Hensman, J. (2017). Convolutional Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 2849–2858.

Whittle, P. (1963). Stochastic-processes in several dimensions. *Bulletin of the International Statistical Institute*, 40(2):974–994.

Wilkinson, W., Andersen, M., Reiss, J. D., Stowell, D., and Solin, A. (2019a). End-to-end probabilistic inference for nonstationary audio analysis. In *International Conference on Machine Learning*, pages 6776–6785. PMLR.

Wilkinson, W., Solin, A., and Adam, V. (2021). Sparse algorithms for Markovian Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 1747–1755. PMLR.

Wilkinson, W. J., Andersen, M. R., Reiss, J. D., Stowell, D., and Solin, A. (2019b). Unifying probabilistic models for time-frequency analysis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3352–3356. IEEE.

Wilkinson, W. J., Chang, P. E., Andersen, M. R., and Solin, A. (2020). State space expectation propagation: Efficient inference schemes for temporal Gaussian processes. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*. PMLR.

Williams, C. K. and Rasmussen, C. E. (1996). Gaussian processes for regression.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International Conference on Machine Learning*, pages 1775–1784.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Stochastic variational deep kernel learning. *arXiv preprint arXiv:1611.00336*.

Wilson, A. G., Knowles, D. A., and Ghahramani, Z. (2012). Gaussian process regression networks. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, pages 1139–1146.

Wilson, J., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. (2020). Efficiently sampling functions from gaussian process posteriors. In *International Conference on Machine Learning*, pages 10292–10302. PMLR.

Wilson, J. T., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. P. (2021). Pathwise conditioning of gaussian processes. *Journal of Machine Learning Research*, 22(105):1–47.

Wu, A., Aoi, M. C., and Pillow, J. W. (2017a). Exploiting gradients and hessians in bayesian optimization and bayesian quadrature. *arXiv preprint arXiv:1704.00060*.

Wu, J., Poloczek, M., Wilson, A. G., and Frazier, P. (2017b). Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems*, pages 5267–5278.

Xi, X., Briol, F.-X., and Girolami, M. (2018). Bayesian quadrature for multiple related integrals. In *International Conference on Machine Learning*, pages 5373–5382. PMLR.

Zanna, L. (2012). Forecast skill and predictability of observed atlantic sea surface temperatures. *Journal of Climate*, 25(14):5047–5056.

Appendix A

A.1 Multiple Dispatch

Multiple Dispatch enables functions to provide specialised implementations depending upon the type of their arguments, and is central to the Julia programming language. Consider `f` in Fig. A.1. In Julia-parlance, `f` is a *function* with two *methods*. In general, if `f` is called on some argument `x`, the first method will be called, since it applies to any data type. If, however, `x` is a real number, the second method will be called. The act of *dispatch* is that of choosing which method of `f` to use for a particular set of arguments. The most specialised method which is applicable will be chosen.

This same concept extends to functions with multiple arguments. `g` is another function, this time with three methods. The first applies to any argument types, the second when `x` is a real number, and the last when both `x` and `y` are real numbers.

```
# Unary case.
f(x::Any) = ...
f(x::Real) = ...

# Binary case.
g(x::Any, y::Any) = ...
g(x::Real, y::Any) = ...
g(x::Real, y::Real) = ...
```

Fig. A.1 Two examples of functions with multiple methods.

It is this mechanism that we employ to provided specialise implementations of the functions discussed in the interfaces in Chapter 2. For example, Fig. A.2 shows four methods of the

function `cov`, each of which is specialised to a different type of GP. Each method of `cov` computes the same thing, they just go about it in different ways.

```
cov(f::GP, x) = ...
cov(f::PosteriorGP, x) = ...
cov(f::BayesianLinearRegressor, x) = ...
cov(f::MyNewGPType, x) = ...
```

Fig. A.2 Implementations of `cov` which specialise on the particular kind of GP they encounter.

Multiple Dispatch is a strict generalisation of *Single Dispatch*, which is used widely in Object Oriented programming languages. In particular methods of classes in Object Oriented languages specialise on their first argument, often called `self`, and do not specialise on the other methods. They do not, however, allow for specialisation based on the type of their other arguments.

Furthermore, note that multiple dispatch is distinct from function overloading – Karpinski (2019) provides a careful discussion of the distinction between the two.

Appendix B

B.1 Conditional Independence Properties of Optimal Approximate Observation Models

Conditional independence structure in the observation model is reflected in the optimal approximate posterior, regardless the precise form of the observation model (Gaussian, Bernoulli, etc). Moreover, for Gaussian observation models, it is possible to find the model in which performing exact inference yields the optimal approximate posterior. These two properties are derived in the following two subsections.

B.1.1 Conditional Independence Structure

This is only a slight extension of the result of Seeger (1999) and Opper and Archambeau (2009), which generalises from reconstruction terms which depend on only a one dimensional marginal of the Gaussian in question, to non-overlapping multi-dimensional marginals of arbitrary size. Let

$$p(\bar{\mathbf{u}}) := \mathcal{N}(\mathbf{u}; \mathbf{m}, \mathbf{C}) \,, \quad q(\bar{\mathbf{u}}) := \mathcal{N}(\bar{\mathbf{u}}; \mathbf{m}^{\mathrm{q}}, \mathbf{C}^{\mathrm{q}}) \,, \tag{B.1}$$

for $\mathbf{m}, \mathbf{m}^{\mathrm{q}} \in \mathbb{R}^N$ and positive-definite matrices $\mathbf{C}, \mathbf{C}^{\mathrm{q}} \in \mathbb{S}_+^N$. Partition $\bar{\mathbf{u}}$ into a collection of T sets, $\bar{\mathbf{u}}_1, ..., \bar{\mathbf{u}}_T$, such that

$$\bar{\mathbf{u}} = \begin{bmatrix} \bar{\mathbf{u}}_1 \\ \vdots \\ \bar{\mathbf{u}}_T \end{bmatrix} . \tag{B.2}$$

Let $\mathbf{m}_t$ and $\mathbf{m}_t^{\mathrm{q}}$ be the blocks of $\mathbf{m}$ and $\mathbf{m}^{\mathrm{q}}$ corresponding to $\bar{\mathbf{u}}_t$. Similarly let $\mathbf{C}_t$ and $\mathbf{C}_t^{\mathrm{q}}$ the on-diagonal blocks of $\mathbf{C}$ and $\mathbf{C}^{\mathrm{q}}$ corresponding to $\bar{\mathbf{u}}_t$. Then the prior and approximate

posterior marginals over $\bar{\mathbf{u}}_t$ are

$$p(\bar{\mathbf{u}}_t) = \mathcal{N}(\bar{\mathbf{u}}_t; \mathbf{m}_t, \mathbf{C}_t)\,, \quad q(\bar{\mathbf{u}}_t) = \mathcal{N}(\bar{\mathbf{u}}_t; \mathbf{m}_t^{\mathrm{q}}, \mathbf{C}_t^{\mathrm{q}})\,, \tag{B.3}$$

due to the marginalisation property of Gaussians.

Assume that the reconstruction term can be written as a sum over terms specific to each $\bar{\mathbf{u}}_t$,

$$r(\mathbf{m}^{\mathrm{q}}, \mathbf{C}^{\mathrm{q}}) = \sum_{t=1}^{T} r_t(\mathbf{m}_t^{\mathrm{q}}, \mathbf{C}_t^{\mathrm{q}})\,, \tag{B.4}$$

for functions $r_1, ..., r_T$. This is a useful assumption because it is satisfied for the model class considered in this work. Under this assumption, the optimal Gaussian approximate posterior density is proportional to

$$p(\bar{\mathbf{u}}) \prod_{t=1}^{T} \mathcal{N}\big(\mathbf{y}_t^{\mathrm{q}}; \bar{\mathbf{u}}_t, [\mathbf{G}_t]^{-1}\big) \tag{B.5}$$

for appropriately-sized surrogate observations $\mathbf{y}_1^{\mathrm{q}}, ..., \mathbf{y}_T^{\mathrm{q}}$ and positive-definite precision matrices $\mathbf{G}_1, ..., \mathbf{G}_T$, which is to say that the optimal approximate posterior is equivalent to the exact posterior under a "surrogate" Gaussian observation model whose density factorises across $\bar{\mathbf{u}}_1, ..., \bar{\mathbf{u}}_T$.

A straightforward way to arrive at this result is via a standard result involving exponential families. Consider an exponential family prior

$$p(\bar{\mathbf{u}}) = h(\bar{\mathbf{u}}) \exp(\langle \eta, \phi(\bar{\mathbf{u}}) \rangle - A(\eta))\,, \tag{B.6}$$

where h is the base measure, ϕ the sufficient-statistic function, A the log partition function, and η the natural parameters. and approximate posterior in the same family,

$$q(\bar{\mathbf{u}}) = h(\bar{\mathbf{u}}) \exp(\langle \eta^{\mathrm{q}}, \phi(\bar{\mathbf{u}}) \rangle - A(\eta^{\mathrm{q}}))\,, \tag{B.7}$$

which differs from p only in its natural parameters η^{q}. Let

$$p(\bar{\mathbf{u}} \,|\, \mathbf{y}) \propto p(\bar{\mathbf{u}})\, p(\mathbf{y} \,|\, \bar{\mathbf{u}}) \tag{B.8}$$

be the posterior over $\bar{\mathbf{u}}$ given observations $\mathbf{y}$ under an arbitrary observation model $p(\mathbf{y} \,|\, \bar{\mathbf{u}})$. It is well-known (for example see Khan and Nielsen (2018)) that the η^{q} minimising $\mathcal{KL}[q(\bar{\mathbf{u}}) \,\|\, p(\bar{\mathbf{u}} \,|\, \mathbf{y})]$ satisfies

$$\eta^{\mathrm{q}} = \eta + (\nabla_{\mu} r)(\mu(\eta^{\mathrm{q}}))\,. \tag{B.9}$$

where μ denotes the expectation parameters $\mu := \mathbb{E}_q[\phi(\bar{\mathbf{u}})]$ and, in an abuse of notation, $\mu(\eta)$ denotes the function computing the mean parameter for any particular natural parameter. Given the canonical parameters $\mathbf{m}$ and $\mathbf{C}$ of a Gaussian, and letting $\Lambda := [\mathbf{C}]^{-1}$, its natural parameters and mean parameters are

$$\eta = (\eta_1, \eta_2) = (\Lambda\mathbf{m}, -\frac{1}{2}\Lambda), \quad \mu = (\mu_1, \mu_2) = (\mathbf{m}, \mathbf{m}\mathbf{m}^\top + \mathbf{C}). \tag{B.10}$$

Let $\Lambda^{\mathrm{q}} := [\mathbf{C}^{\mathrm{q}}]^{-1}$ be the precision of q, $\Lambda := \mathbf{C}^{-1}$ the precision of p, and recall that the optimal Gaussian approximate posterior satisfies

$$\Lambda^{\mathrm{q}} = \Lambda - 2\left(\nabla_{\mathbf{C}^{\mathrm{q}}} r\right)(\mathbf{m}^{\mathrm{q}}, \mathbf{C}^{\mathrm{q}})\,. \tag{B.11}$$

Due to the assumed structure in r, $\nabla_{\mathbf{C}^{\mathrm{q}}} r$ is block-diagonal:

$$\left(\nabla_{\mathbf{C}^{\mathrm{q}}} r\right)(\mathbf{m}^{\mathrm{q}}, \mathbf{C}^{\mathrm{q}}) = \begin{bmatrix} (\nabla_{\mathbf{C}_1^{\mathrm{q}}} r_1)(\mathbf{m}_1^{\mathrm{q}}, \mathbf{C}_1^{\mathrm{q}}) & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & (\nabla_{\mathbf{C}_T^{\mathrm{q}}} r_T)(\mathbf{m}_T^{\mathrm{q}}, \mathbf{C}_T^{\mathrm{q}}) \end{bmatrix}. \tag{B.12}$$

Observe that each on-diagonal block involves only the corresponding term in r, i.e. the t^{th} block is only a function of r_t.

Equating the exact posterior precision under the approximate model in Eq. (B.5) with the optimal approximate posterior precision yields

$$\Lambda + \begin{bmatrix} \Lambda_1^{\mathrm{q}} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \Lambda_T^{\mathrm{q}} \end{bmatrix} = \Lambda + \begin{bmatrix} -2\left(\nabla_{\mathbf{C}_1^{\mathrm{q}}} r_1\right)(\mathbf{m}_1^{\mathrm{q}}, \mathbf{C}_1^{\mathrm{q}}) & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & -2\left(\nabla_{\mathbf{C}_T^{\mathrm{q}}} r_T\right)(\mathbf{m}_T^{\mathrm{q}}, \mathbf{C}_T^{\mathrm{q}}) \end{bmatrix} \tag{B.13}$$

From the above we deduce that letting $\Lambda_t^{\mathrm{q}} := -2\left(\nabla_{\mathbf{C}_t^{\mathrm{q}}} r_t\right)(\mathbf{m}_t^{\mathrm{q}}, \mathbf{C}_t^{\mathrm{q}})$ ensures that the posterior precision under the surrogate model and the precision of the approximate posterior coincide for the optimal q.

Similarly, the optimal approximate posterior mean satisfies

$$\Lambda^{\mathrm{q}}\mathbf{m}^{\mathrm{q}} = \Lambda\mathbf{m} + [\nabla_\mu r]_1, \tag{B.14}$$

where $[\nabla_\mu r]_1$ denotes the component of the gradient of r w.r.t. μ corresponding to μ_1. Equating the optimal posterior mean under the approximate model in Eq. (B.5) with that of

the optimal approximate posterior yields

$$\begin{bmatrix} \mathbf{G}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{G}_T \end{bmatrix} \begin{bmatrix} \mathbf{y}_1^{\mathrm{q}} \\ \vdots \\ \mathbf{y}_T^{\mathrm{q}} \end{bmatrix} = \begin{bmatrix} [\nabla_\mu r_1]_1 \\ \vdots \\ [\nabla_\mu r_T]_1 \end{bmatrix}. \tag{B.15}$$

This implies that $\mathbf{y}_t^{\mathrm{q}} := \mathbf{G}_t^{-1}[\nabla_\mu r]_{1t}$.

B.1.2 Approximate Inference via Exact Inference

Recall the standard saturated bound introduced by Titsias (2009), that is obtained at the optimal approximate posterior:

$$\mathcal{L} = \log \mathcal{N}(\mathbf{y}; \mathbf{m}_\mathbf{f}, \mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\mathbf{C}_{\bar{\mathbf{u}}\mathbf{f}} + \mathbf{S}) - \frac{1}{2}\mathrm{tr}\big(\mathbf{S}^{-1}[\mathbf{C}_\mathbf{f} - \mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\mathbf{C}_{\bar{\mathbf{u}}\mathbf{f}}]\big). \tag{B.16}$$

The first term is simply to log marginal likelihood of the LGSSM defined in Eq. (3.44) if f is separable, or Eq. (3.45) if it is sum-separable.

Recall from Eq. (3.40) that

$$\mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}} = \begin{bmatrix} \mathbf{B}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{B}_T \end{bmatrix},$$
$$\mathbf{B}_t := \mathbf{C}_{\mathbf{f}_t\mathbf{u}_t}\Lambda_{\mathbf{u}_t}\mathbf{H}_{M_\tau D},$$

so the trace term in Eq. (B.16) can be written as

$$\begin{aligned} \mathrm{tr}\big(\mathbf{S}^{-1}[\mathbf{C}_\mathbf{f} - \mathbf{C}_{\mathbf{f},\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\mathbf{C}_{\bar{\mathbf{u}},\mathbf{f}}]\big) &= \mathrm{tr}\big(\mathbf{S}^{-1}[\mathbf{C}_\mathbf{f} - \mathbf{C}_{\mathbf{f},\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\mathbf{C}_{\bar{\mathbf{u}}}\Lambda_{\bar{\mathbf{u}}}\mathbf{C}_{\bar{\mathbf{u}},\mathbf{f}}]\big) \\ &= \sum_{t=1}^{T} \mathrm{tr}\big(\mathbf{S}_t^{-1}[\mathbf{C}_{\mathbf{f}_t} - \mathbf{B}_t\mathbf{C}_{\bar{\mathbf{u}}_t}\mathbf{B}_t]\big) \qquad \text{(B.17)} \\ &= \sum_{t=1}^{T} \mathrm{tr}\big(\mathbf{S}_t^{-1}[\mathbf{C}_{\mathbf{f}_t} - \mathbf{C}_{\mathbf{f}_t\mathbf{u}_t}\Lambda_{\mathbf{u}_t}\mathbf{C}_{\mathbf{u}_t}\Lambda_{\mathbf{u}_t}\mathbf{C}_{\mathbf{u}_t\mathbf{f}_t}]\big) \\ &= \sum_{t=1}^{T} \mathrm{tr}\big(\mathbf{S}_t^{-1}[\mathbf{C}_{\mathbf{f}_t} - \mathbf{C}_{\mathbf{f}_t\mathbf{u}_t}\Lambda_{\mathbf{u}_t}\mathbf{C}_{\mathbf{u}_t\mathbf{f}_t}]\big) \qquad \text{(B.18)} \end{aligned}$$

These quantities can be computed either by running the approximate model forwards through time and computing the marginal statistics using Eq. (B.17), or via the $\kappa^{\mathbf{r}}$ and κ^{τ} directly using Eq. (B.18).

Observe that, as with any $\mathbf{f}_t$ from the training data, the marginal distribution over some $\mathbf{f}_{*t}$ under the approximate posterior only involves $\bar{\mathbf{u}}_t$ as $\mathbf{f}_{*t} \perp\!\!\!\perp \bar{\mathbf{u}}_{\setminus t} \mid \bar{\mathbf{u}}_t$:

$$
\begin{aligned}
q(\mathbf{f}_{*t}) &= \mathcal{N}\big(\mathbf{f}_{*t}; \hat{\mathbf{m}}_{\mathbf{f}_{*t}}, \mathbf{C}^{\mathrm{q}}_{\mathbf{f}_{*t}}\big) \text{ where} \\
\hat{\mathbf{m}}_{\mathbf{f}_{*t}} &:= \mathbf{m}_{\mathbf{f}_{*t}} + \mathbf{C}_{\mathbf{f}_{*t}\mathbf{u}_t} \Lambda_{\mathbf{u}_t} \mathbf{H}_{\mathbf{u}_t} (\hat{\mathbf{m}}_{\bar{\mathbf{u}}_t} - \mathbf{m}_{\bar{\mathbf{u}}_t}), \\
\mathbf{C}^{\mathrm{q}}_{\mathbf{f}_{*t}} &:= \mathbf{C}_{\mathbf{f}_{*t}} - \mathbf{C}_{\mathbf{f}_{*t}\mathbf{u}_t} \Lambda_{\mathbf{u}_t} \mathbf{H}_{\mathbf{u}_t} \left[\mathbf{C}_{\bar{\mathbf{u}}_t} - [\hat{\Lambda}^{*}_{\bar{\mathbf{u}}_t}]^{-1} \right] \mathbf{H}^{\top}_{\mathbf{u}_t} \Lambda_{\mathbf{u}_t} \mathbf{C}_{\mathbf{u}_t \mathbf{f}_{*t}}.
\end{aligned}
$$

Performing smoothing in the approximate model provides $\hat{\mathbf{m}}_{\bar{\mathbf{u}}_t}$ and $[\hat{\Lambda}^{*}_{\bar{\mathbf{u}}_t}]^{-1}$, from which the optimal approximate posterior marginals are straightforwardly obtained via the above.

B.1.3 Block-Diagonal Structure

Furthermore, this conditional independence property implies that

$$
\mathbf{C}_{\mathbf{f}_t \bar{\mathbf{u}}} \mathbf{C}^{-1}_{\bar{\mathbf{u}}} = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{C}_{\bar{\mathbf{f}}_t \bar{\mathbf{u}}_t} \mathbf{C}^{-1}_{\bar{\mathbf{u}}_t} & \dots & \mathbf{0} \end{bmatrix}. \tag{B.19}
$$

This is easily proven by considering that, were it not the case, then

$$
\mathbb{E}[\mathbf{f}_t \mid \mathbf{u}] \neq \mathbb{E}[\mathbf{f}_t \mid \mathbf{u}_t], \tag{B.20}
$$

for any non-zero $\mathbf{u}$, which is a contradiction. It follows from repeated application of Eq. (B.19) that the larger matrix $\mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}} \mathbf{C}^{-1}_{\bar{\mathbf{u}}}$ is block-diagonal, and is given by

$$
\mathbf{C}_{\mathbf{f}\bar{\mathbf{u}}} \mathbf{C}^{-1}_{\bar{\mathbf{u}}} := \begin{bmatrix} \mathbf{C}_{\mathbf{f}_1 \bar{\mathbf{u}}} \mathbf{C}^{-1}_{\bar{\mathbf{u}}} \\ \vdots \\ \mathbf{C}_{\mathbf{f}_T \bar{\mathbf{u}}} \mathbf{C}^{-1}_{\bar{\mathbf{u}}} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{\mathbf{f}_1 \bar{\mathbf{u}}_1} \mathbf{C}^{-1}_{\bar{\mathbf{u}}_1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{C}_{\mathbf{f}_T \bar{\mathbf{u}}_T} \mathbf{C}^{-1}_{\bar{\mathbf{u}}_T}. \end{bmatrix} \tag{B.21}
$$

Similar manipulations reveal that the same property holds in the sum-separable case.

B.2 Additional Experiment Details

B.2.1 Benchmarking Experiment

The kernel of the GP used in all experiments is

$$\kappa((\mathbf{r}, \tau), (\mathbf{r}', \tau')) = \kappa^{\mathbf{r}}(\mathbf{r}, \mathbf{r}')\, \kappa^{\tau}(\tau, \tau') \tag{B.22}$$

where $\kappa^{\mathbf{r}}$ is an Exponentiated Quadratic kernel with length scale 0.9 and amplitude 0.92, and κ^{τ} is a Matern-3/2 kernel with length scale 1.2. The particular values of the length scales / amplitudes are of little importance to the proof-of-concept experiments presented in this work – they were chosen pseudo-randomly.

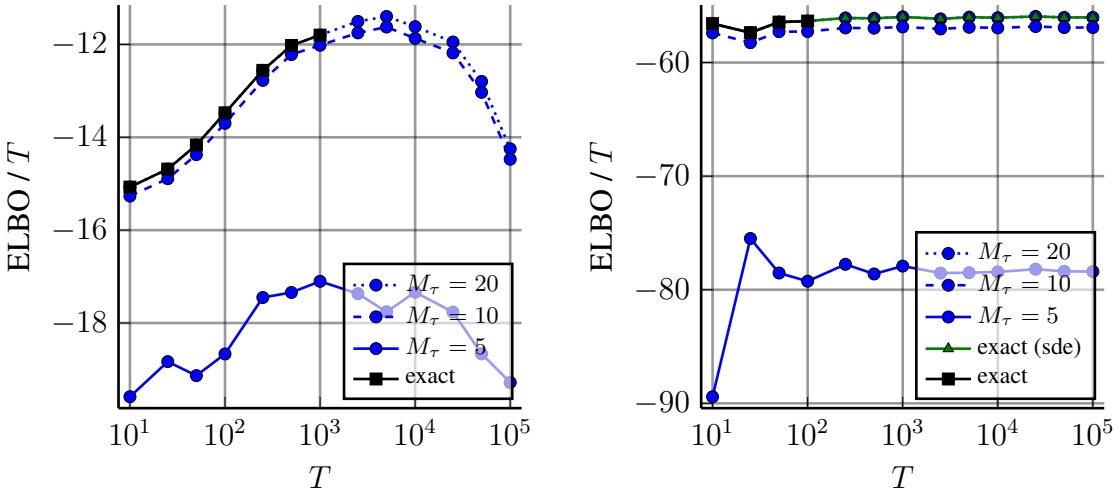


Fig. B.1 The ELBO obtained vs the exact LML. The bound appears reasonably tight when $M_\tau = 10$ are used per time point, and very tight for $M_\tau = 20$. $M_\tau = 5$ is clearly insufficient.

These experiments were conducted using a single thread on a 2019 MacBook Pro with 2.6 GHz CPU. Timings produced using benchmarking functionality provided by Chen and Revels (2016).

Sum-Separable Experiments

Similar experiments to those in section Sec. 3.7 were performed with the sum-separable kernel given by adding two separable kernels of the form in Eq. (3.1), although with similar length-scales and amplitudes. The results are broadly similar, although the state space approximations and state space + pseudo-point approximations take a bit longer to run as there are twice as many latent dimensions for a given number of pseudo-points than in the

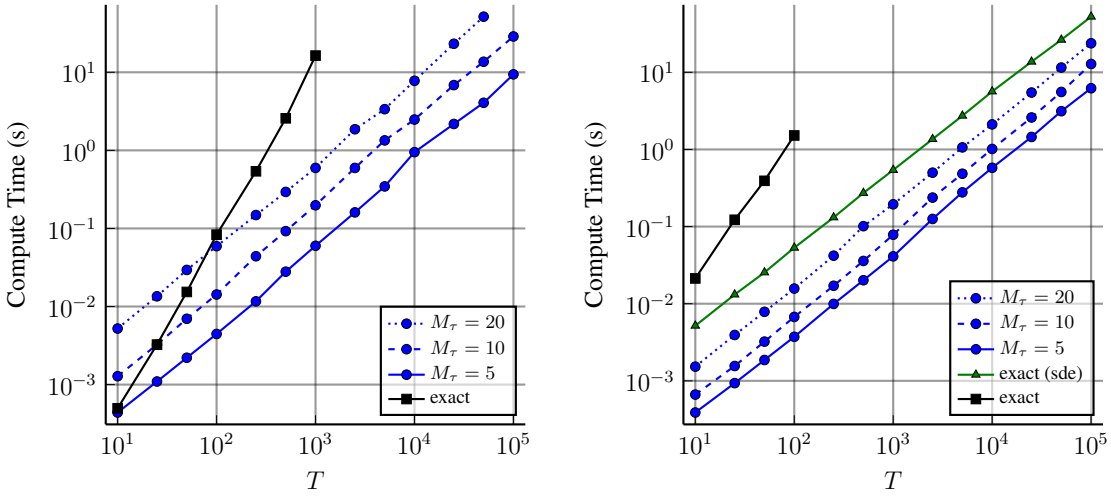


Fig. B.2 Time to compute LML exactly vs ELBO with a sum of two separable kernels. Left: irregular samples as per Fig. 3.7. Right: regular samples with missing data as per Fig. 3.8. Observe that, due to the increased latent dimensionality of the sum-separable model, it takes longer to compute the ELBO (and LML using the vanilla state space approximation) than in the separable case.

separable model. As before, these experiments should be thought of purely as a proof of concept.

B.2.2 Climatology Data

The spatial locations of the pseudo-points were chosen via k-means clustering of lat-lon coordinates of sold apartments, using Clustering.jl, and were not optimised beyond that.[1]

The separable kernel was

$$\kappa((\mathbf{r}, \tau), (\mathbf{r}', \tau')) = s\, \kappa^{\mathbf{r}}(\Lambda\mathbf{r}, \Lambda\mathbf{r}')\, \kappa^{\tau}(\lambda\tau, \lambda\tau') \tag{B.23}$$

where κ_τ is a standardised Matérn-$\frac{5}{2}$, $\kappa_{\mathbf{r}}$ is a standardised Exponentiated Quadratic, Λ is a diagonal matrix with positive elements, $\lambda > 0$, $s > 0$. Initialisation: $\lambda = 10^{-2}$, $\Lambda_{d,d} = 1$, $s = 1$, $d \in \{1, 2, 3\}$. Observation noise variance initialised to 0.5.

The L-BFGS implementation provided by Mogensen and Riseth (2018) was utilised to optimise the ELBO, with memory $M = 50$ iterations, and gradients computed using the Zygote.jl algorithmic differentiation tool (Innes, 2018). All kernel parameters constrained to be positive by optimising the log of their value, and observation noise variance constrained

[1] `https://github.com/JuliaStats/Clustering.jl`

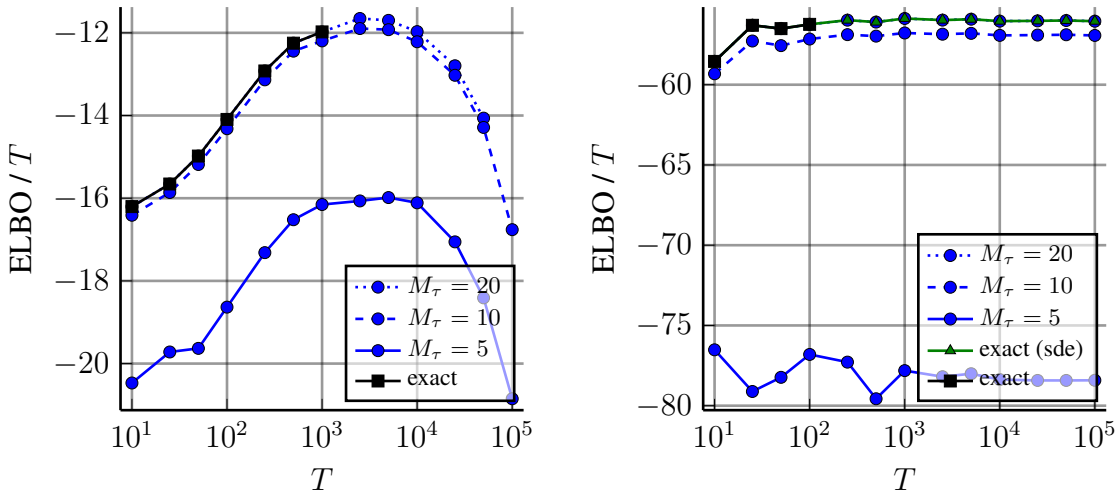


Fig. B.3 Analogue of Fig. B.1 for Fig. B.2. As before, $M_\tau = 5$ is clearly insufficient for accurate inference, while $M_\tau = 20$ is very close to the LML.

to be in $[10^{-2}, 2]$ via a re-scaled logit transformation – this is justifiable as the data itself was standardised to have unit variance.

The sum-separable model comprises a sum of two GPs with kernels of this form. Initialisation: $\lambda = \{10^{-3}, 10^{-1}\}$, $\Lambda_{d,d} = \{1.0, 5.0\}$, $s = \{0.7, 0.3\}$. The same optimisation procedure was used.

B.2.3 Apartment Data

The spatial locations of the pseudo-points were chosen via k-means clustering of lat-lon coordinates of sold apartments, using Clustering.jl, and were not optimised beyond that. $M_\tau = 75$ pseudo-points used per time point.

The separable kernel was

$$\kappa((\mathbf{r}, \tau), (\mathbf{r}', \tau')) = s\, \kappa^{\mathbf{r}}(\Lambda\mathbf{r}, \Lambda\mathbf{r}')\, \kappa^{\tau}(\lambda\tau, \lambda\tau') \tag{B.24}$$

where κ_τ is a standardised Matérn-$\frac{3}{2}$, $\kappa_{\mathbf{r}}$ is a standardised Exponentiated Quadratic, Λ is a diagonal matrix with positive elements, $\lambda > 0$, $s > 0$. Initialisation: $\lambda = 10^{-2}$, $\Lambda_{d,d} = 1$, $s = 1$, $d \in \{1, 2\}$. Observation noise variance initialised to 0.5.

The L-BFGS implementation provided by Mogensen and Riseth (2018) was utilised to optimise the ELBO, with memory $M = 50$ iterations, and gradients computed using the Zygote.jl algorithmic differentiation tool (Innes, 2018). All kernel parameters constrained to

be positive by optimising the log of their value, and observation noise variance constrained to be in $[10^{-2}, 2]$ via a re-scaled logit transformation – this is justifiable as the data itself was standardised to have unit variance.

The sum-separable model comprises a sum of two GPs with kernels of this form. Initialisation: $\lambda = \{10^{-3}, 10^{-1}\}$, $\Lambda_{d,d} = \{1.0, 5.0\}$, $s = \{0.7, 0.3\}$. The same optimisation procedure was used.

B.3 Efficient Inference in Linear Latent Gaussian Models

Consider the linear-Gaussian model

$$\begin{aligned} \mathbf{x} &\sim \mathcal{N}(\mathbf{m}_\mathbf{x}, \mathbf{C}_\mathbf{x}) \\ \mathbf{y} \mid \mathbf{x} &\sim \mathcal{N}(\mathbf{A}\mathbf{x} + \mathbf{a}, \mathbf{Q}) \end{aligned}$$

where $\mathbf{m}_\mathbf{x} \in \mathbb{R}^{D_x}$ and $\mathbf{a} \in \mathbb{R}^{D_y}$ are vectors, $\mathbf{C}_\mathbf{x}$ is a $D_x \times D_x$ positive-definite matrix, $\mathbf{Q}$ is a $D_y \times D_y$ diagonal positive definite matrix, and $\mathbf{A}$ is a $D_y \times D_x$ matrix. We need to

1. generate samples from the marginal distribution over $p(\mathbf{y})$,
2. compute the marginals $p(\mathbf{y}_n)$, $n = 1, ..., D_y$,
3. compute the LML $p(\mathbf{y})$, and
4. compute the posterior distribution $p(\mathbf{x} \mid \mathbf{y})$.

All of these operations can be performed exactly in polynomial-time since $\mathbf{x}$ and $\mathbf{y}$ are jointly Gaussian distributed. However, there are two approaches for computing 1, 3, and 4, one of which will be faster depending upon D_x and D_y.

In this section we analyse these approaches. We do this to prepare for deriving the additional algorithms needed to perform the above operations efficiently when $\mathbf{A} := \mathbf{B}\mathbf{C}$, for tall $\mathbf{B}$ and wide $\mathbf{C}$.

B.3.1 Preliminaries

The marginal distibution over $\mathbf{y}$ is

$$\mathbf{y} \sim \mathcal{N}(\mathbf{m}_\mathbf{y}, \mathbf{C}_\mathbf{y}), \quad \mathbf{m}_\mathbf{y} := \mathbf{A}\mathbf{m}_\mathbf{x} + \mathbf{a}, \quad \mathbf{C}_\mathbf{y} := \mathbf{A}\mathbf{C}_\mathbf{x}\mathbf{A}^\top + \mathbf{Q}.$$

Computing $\mathbf{AC_x}$ requires $\mathcal{O}(D_y D_x^2)$ operations, and $(\mathbf{AC_x})\mathbf{A}^\top$ requires $\mathcal{O}\big(D_y^2 D_x\big)$, so constructing the marginals takes roughly $\mathcal{O}\big(D_y D_x^2 + D_y^2 D_x\big)$ operations.

By computing $p(\mathbf{y})$ we will mean computing $\mathbf{m_y}$ and $\mathbf{C_y}$.

B.3.2 Sampling

First consider sampling from the marginal distribution over $p(\mathbf{y})$. The two approaches to this are:

1. Ancestral sampling: first sample from $p(\mathbf{x})$ then from $p(\mathbf{y} \mid \mathbf{x})$.
2. Direct marginal sampling: compute the marginal distribution $p(\mathbf{y})$ and sample from it directly.

Ancestral sampling requires computing the Cholesky factorisation of $\mathbf{C_x}$, thus the overall algorithm requires $\mathcal{O}(D_x^3 + D_x D_y)$ scalar operations. Conversely, computing $p(\mathbf{y})$ and sampling from it requires $\mathcal{O}\big(D_y D_x^2 + D_y^2 D_x + D_y^3\big)$ scalar operations. So if D_x is much smaller than D_y we are better off using ancestral sampling, but if D_x is much larger than D_y then direct marginal sampling is better.

B.3.3 Computing Marginal Probabilities

To compute all $p(\mathbf{y}_n)$ we must compute both $\mathbf{m_y}$ and the diagonal of $\mathbf{C_y}$. $\mathbf{m_y}$ requires only $\mathcal{O}(D_x D_y)$ scalar operations. Once $\mathbf{AC_x}$ has been computed, obtaining the diagonal of $\mathbf{C_y}$ requires only an additional $\mathcal{O}(D_x D_y)$ scalar operations, so the whole operation requires roughly $\mathcal{O}(D_y D_x^2)$ operations.

B.3.4 Computing the Log Marginal Likelihood and Posterior

These two operations can be performed separately, but it typically makes sense to perform them together as the majority of computational work is shared between them.

Algorithm 1 LML by and posterior by factorising $p(\mathbf{y})$. Approx. number of scalar operations on the right of each line.

1: **Naive-Inference**: $\mathbf{m_x}$, $\mathbf{C_x}$, $\mathbf{A}$, $\mathbf{a}$, $\mathbf{Q}$, $\mathbf{y}$
2: $\mathbf{V} \leftarrow \mathbf{AC_x}$ $\mathcal{O}(D_y D_x^2)$
3: $\mathbf{C_y} \leftarrow \mathbf{VA}^\top + \mathbf{Q}$ $\mathcal{O}\big(D_y^2 D_x\big)$
4: $\mathbf{U} \leftarrow \text{cholesky}(\mathbf{C_y})$ $\mathcal{O}\big(D_y^3\big)$
5: $\mathbf{B} \leftarrow \mathbf{U}^{-\top}\mathbf{V}$ $\mathcal{O}\big(D_y^2 D_x\big)$
6: $\alpha \leftarrow \mathbf{U}^{-\top}(\mathbf{y} - (\mathbf{Am_x} + \mathbf{a}))$ $\mathcal{O}\big(D_y^2\big)$
7: $\text{lml} \leftarrow -\frac{1}{2}\left[D_y \log 2\pi + 2 \log\det \mathbf{U} + \alpha^\top \alpha\right]$ $\mathcal{O}(D_y)$
8: $\mathbf{m_{x|y}} \leftarrow \mathbf{m_x} + \mathbf{B}^\top \alpha$ $\mathcal{O}(D_y D_x)$
9: $\mathbf{C_{x|y}} \leftarrow \mathbf{C_x} + \mathbf{B}^\top \mathbf{B}$ $\mathcal{O}(D_x^2 D_y)$
10: **return** $\mathbf{m_{x|y}}, \mathbf{C_{x|y}}, \text{lml}$

Algorithm 2 LML and posterior by utilising the matrix inversion and determinant lemmas. Approx. number of scalar operations on the right of each line. Note that since $\mathbf{Q}$ is diagonal, its Cholesky factorisation is also diagonal.

1: **Low-Rank-Inference**: $\mathbf{m_x}$, $\mathbf{C_x}$, $\mathbf{A}$, $\mathbf{a}$, $\mathbf{Q}$, $\mathbf{y}$
2: $\mathbf{U_Q} \leftarrow \text{cholesky}(\mathbf{Q})$ $\mathcal{O}(D_y)$
3: $\mathbf{U_x} \leftarrow \text{cholesky}(\mathbf{C_x})$ $\mathcal{O}(D_x)$
4: $\mathbf{B} \leftarrow \mathbf{U_x A}^\top \mathbf{U_Q}^{-1}$ $\mathcal{O}(D_x^2 D_y)$
5: $\mathbf{U} \leftarrow \text{cholesky}\big(\mathbf{BB}^\top + \mathbf{I}\big)$ $\mathcal{O}(D_x^3 + D_x^2 D_y)$
6: $\mathbf{G} \leftarrow \mathbf{U}^{-\top}\mathbf{U_x}$ $\mathcal{O}(D_x^3)$
7: $\mathbf{C_{x|y}} \leftarrow \mathbf{G}^\top \mathbf{G}$ $\mathcal{O}(D_x^3)$
8: $\delta \leftarrow \mathbf{U_Q}^{-\top}(\mathbf{y} - (\mathbf{Am_x} + \mathbf{a}))$ $\mathcal{O}(D_y)$
9: $\beta \leftarrow \mathbf{B}\delta$ $\mathcal{O}(D_y D_x)$
10: $\mathbf{m_{x|y}} \leftarrow \mathbf{m_x} + \mathbf{G}^\top(\mathbf{U}^{-\top}\beta)$ $\mathcal{O}(D_x^2)$
11: $\text{lml} \leftarrow -\frac{1}{2}\left[\delta^\top \delta - (\mathbf{U}^{-\top}\beta)^\top \mathbf{U}^{-\top}\beta + D_y \log 2\pi + 2\log\det \mathbf{U} + 2\log\det \mathbf{Q}\right]$ $\mathcal{O}(D_x^2)$
12: **return** $\mathbf{m_{x|y}}, \mathbf{C_{x|y}}, \text{lml}$

Note that Algorithm 1 and Algorithm 2 are locally scoped, so the symbols don't necessarily correspond to the same quantities. For example, $\mathbf{B}$ is different in each algorithm.

B.3.5 Bottleneck Linear-Gaussian Observation Models

The above inference methods assume no particular structure in $\mathbf{A}$, however, recall that Eq. (3.44) gives $\mathbf{A}$ at time t to be

$$\mathbf{A} = \mathbf{C}_{\mathbf{f}_{n,t}\mathbf{u}_t}\Lambda_{\mathbf{u}_t}\mathbf{H}_{\mathbf{u}} \tag{B.25}$$

where $\mathbf{H}_{\mathbf{u}}$ is $M \times MD$ and $\mathbf{C}_{\mathbf{f}_{n,t}\mathbf{u}_t}$ is $N \times M$. Most kernels have $D > 1$, so it's worth determining whether we can utilise this structure to accelerate inference.

To this end consider a model given by

$$\mathbf{z} := \mathbf{H}\mathbf{x} + \mathbf{h}, \quad \mathbf{x} \sim \mathcal{N}(\mathbf{m}_{\mathbf{x}}, \mathbf{C}_{\mathbf{x}}) \tag{B.26}$$

$$\mathbf{y} := \mathbf{B}\mathbf{z} + \mathbf{b} + \varepsilon, \quad \varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \tag{B.27}$$

where $\mathbf{H} \in \mathbb{R}^{M\times DM}$, $\mathbf{h} \in \mathbb{R}^M$, $\mathbf{B} \in \mathbb{R}^{N\times M}$, $\mathbf{b}, \varepsilon \in \mathbb{R}^N$, and $\mathbf{Q} \in \mathbb{R}^{N\times N}$ is a positive-definite diagonal matrix. We call this a *bottleneck* model, since $\mathbf{z}$ carries all of the information in $\mathbf{x}$ needed to perform inference in $\mathbf{y}$, its dimension is less than that of $\mathbf{x}$ and $\mathbf{y}$ in the problems that we consider.

Observe that this model forms a two-state degenerate Markov chain. Algorithm 3 utilises this Markov structure, and is able to recycle Algorithm 2 as a consequence. It comprises three broad components: computing the marginals over $\mathbf{z}$, computing the posterior $\mathbf{z}|\mathbf{y}$, and finally computing the posterior $\mathbf{x}|\mathbf{y}$. This last step is equivalent to performing a single step of RTS smoothing.

Algorithm 3 LML and posterior. Utilises the matrix inversion and determinant lemmas, and the bottlenecked structure of the model. Approx. number of scalar operations on the right of each line.

Bottleneck-Inference: $\mathbf{m}_{\mathbf{x}}$, $\mathbf{C}_{\mathbf{x}}$, $\mathbf{B}$, $\mathbf{b}$, $\mathbf{Q}$, $\mathbf{H}$, $\mathbf{h}$, $\mathbf{y}$

$\mathbf{m}_{\mathbf{z}} \leftarrow \mathbf{H}\mathbf{m}_{\mathbf{x}} + \mathbf{h}$ $\quad \mathcal{O}(D_z D_x)$

$\mathbf{C}_{\mathbf{z}} \leftarrow \mathbf{H}\mathbf{C}_{\mathbf{x}}\mathbf{H}^\top + \mathbf{h}$ $\quad \mathcal{O}(D_z D_x^2 + D_z^2 D_x)$

$\mathbf{m}_{\mathbf{z}|\mathbf{y}}, \mathbf{C}_{\mathbf{z}|\mathbf{y}}, \text{lml} \leftarrow$ **Low-Rank-Inference**$(\mathbf{m}_{\mathbf{z}}, \mathbf{C}_{\mathbf{z}}, \mathbf{B}, \mathbf{b}, \mathbf{Q})$ $\quad \mathcal{O}(D_z^2 D_y + D_z^3)$

$\mathbf{U} \leftarrow \text{cholesky}\big(\mathbf{C}_{\mathbf{z}|\mathbf{y}}\big)$ $\quad \mathcal{O}(D_z^3)$

$\mathbf{G} \leftarrow \mathbf{C}_{\mathbf{x}}\mathbf{H}^\top\mathbf{U}^{-1}\mathbf{U}^{-\top}$ $\quad \mathcal{O}(D_x^2 D_z)$

$\mathbf{m}_{\mathbf{x}|\mathbf{y}} \leftarrow \mathbf{m}_{\mathbf{x}} + \mathbf{G}^\top(\mathbf{m}_{\mathbf{z}|\mathbf{y}} - \mathbf{m}_{\mathbf{z}})$ $\quad \mathcal{O}(D_z D_x)$

$\mathbf{C}_{\mathbf{x}|\mathbf{y}} \leftarrow \mathbf{C}_{\mathbf{x}} + \mathbf{G}^\top(\mathbf{C}_{\mathbf{z}|\mathbf{y}} - \mathbf{C}_{\mathbf{z}})\mathbf{G}$ $\quad \mathcal{O}(D_z^2 D_x + D_x^2)$

return $\mathbf{m}_{\mathbf{x}|\mathbf{y}}, \mathbf{C}_{\mathbf{x}|\mathbf{y}}, \text{lml}$

Observe that this algorithm exchanges $\mathcal{O}(D_x^3)$ for $\mathcal{O}(D_z^3)$ operations. Recalling that $D_x = MD$ and $D_z = M$, we expect that this algorithm will produce better performance than Algorithm 2 for some value of $D > 1$. Since the exact value of D at which this change will occur is unclear, and the optimal choice of algorithm for the experiments in this work depends on this, we investigate the effect of D, M, and N on the performance of each algorithm in the next subsection.

B.3.6 Benchmarking Inference

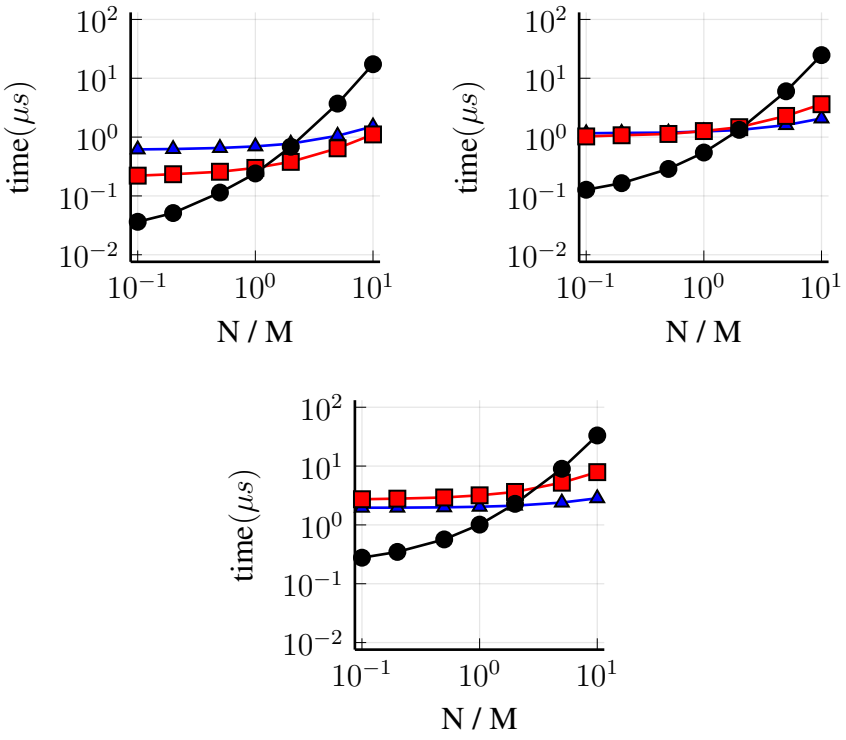


Fig. B.4 Black-circle=naive, red square=low rank, blue triangle=bottleneck. All experiments conducted using $M = 100$ pseudo-points. Left: $D = 1$, Middle: $D = 2$, Right: $D = 3$.

Fig. B.4 shows the performance of the three different algorithms for computing the LML and posterior distribution as the dimensions of the linear-Gaussian model's dimensions change. Black lines with circles use Algorithm 1 (labelled *naive*), red lines with squares used Algorithm 2 (labelled *low rank*), and blue lines with triangles use Algorithm 3 (labelled *bottleneck*). The experiments are set up to match situations encountered in the spatio-temporal models discussed in this paper – they are parametrised in terms of the number of M, N, and D. M is fixed to 100 across all three graphs, the total number of latent dimensions is MD

for $D \in \{1, 2, 3\}$, corresponding to the Matérn-$1/2$, Matérn-$3/2$, and Matérn-$5/2$ kernels respectively. The number of observations N range between $0.1M = 10$ and $10M = 1000$.

As expected the *naive* algorithm performs better when $N < M$, but this quickly changes when $N > M$. For the kinds of problems encountered in this work we generally have that $N > M$, which would suggest that correct choice in our work is typically the *low-rank* algorithm. For $M = N$ the *naive* algorithm tends to be faster, owing to the smaller number of operations used – it's just a shorter algorithm than the *low-rank* algorithm.

Algorithm 3, *bottleneck*, performs similarly or better than Algorithm 2 for $D = 2$ and $D = 3$. The gap between the two grows as N grows, suggesting that Algorithm 3 will typically be a better choice for large M. Indeed, even for $D = 1$ the difference between the two becomes close for large N.

Given that N, M and D determine which of the three algorithms is optimal, one must choose appropriately for any given application. We adopt the bottleneck algorithm in all experiments in this work because we consistently work in regimes where $N > M$ at most points in time, and we do not make use of any kernels for which $D = 1$.

Also note that these results highlight that using Algorithm 2 to perform the second step in Algorithm 3 is only optimal if $N > M$. If a problem were encountered for which $N < M$ it would be prudent to consider replacing it with Algorithm 1.