Large Linear Multi-output Gaussian Process Learning

Vladimir Yurivich Feinberg

A Thesis

Presented to the Faculty

of Princeton University

in Candidacy for the Degree

of Bachelor of Science in Engineering

Department of Computer Science

Advisers

Professor Kai Li

Professor Barbera Engelhardt

June 2017

Abstract

Gaussian process (GP) models, which put a distribution over arbitrary functions in a continuous domain, can be generalized to the multi-output case; a common way of doing this is to use a linear model of coregionalization [2]. Such models can learn correlations across the multiple outputs, which can then be exploited to share knowledge across the multiple outputs. For instance, temperature data from disparate regions over time can contribute to a predictive weather model that is more accurate than the same model applied to a single region. While model learning can be performed efficiently for single-output GPs, the multi-output case still requires approximations for large numbers of observations across all outputs. In this work, we propose a new method, Large Linear GP (LLGP), which estimates covariance hyperparameters for multi-dimensional outputs and one-dimensional inputs. Our approach learns GP kernel hyperparameters at an asymptotically faster rate than the current state of the art. When applied to real time series data, we find this theoretical improvement is realized with LLGP being generally an order of magnitude faster while improving or maintaining predictive accuracy. Finally, we discuss extensions of our approach to multidimensional inputs.

Acknowledgements

I would like to thank my two advisers, Professors Kai Li and Barbara Engelhardt, for their consistent guidance throughout the research process. Both advisers had a crucial role in taking the project into a promising direction, and both also helped me maintain course to a successful outcome. In addition, Li-Fang Cheng also played a formative role in this work: she has helped me conduct my experiments and also advised me during the course of my research.

The Princeton University Computer Science Department also graciously offered time and space in several senses; first, the department reserved a location for seniors like myself to have a dedicated work environment, and second, the department provided computational resources on which I could perform validation experiments.

Finally, I'd like to mention Jeffrey Dwoskin, whose publicly available Princeton LaTeX template spared me many headaches in manual typesetting.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Gaussian processes (GPs) are a popular nonlinear regression method that innately capture function smoothness across inputs as defined by their covariance function [29]. GPs seamlessly extend to multi-output learning, picking up on latent cross-output processes, where in multi-output learning the objective is to build a probabilistic regression model over vector-valued observations.

The covariance kernel describes the interaction between data points: data points that are closer together (as specified by a higher kernel value at that pair) will have expected values for regressor variables that are closer together [25]. A multiple output GP behaves much like a single-output one, where in addition to the input features we add a tag indicating which output the data point is associated to. In this way, a multi-output GP learns multiple regressions at once, aided by learning the

cross-covariance across outputs—this typically results in an improvement in accuracy [2].

This multi-output extension of GPs deserves careful attention. First, the departure from a single-output setting makes GP modeling typical approaches tractable, so alternate methods need to be developed. Second, there are a variety of use cases that benefit from efficient methods to apply the multi-output extension to GPs.

1.1.1 Applications

Multiple-output GPs appear in a variety of contexts, demonstrating the need for efficient model learning. Faster training times will enable researchers to explore more models.

Genton and Kleiber discuss the use of multi-output GPs in geostatistics [11]. They review several kernels with different cross-covariance models and their applicability to multivariate statistical analysis for prediction related to geographic events (Figure 1.1). The authors describe the linear model of coregionalization, the focus of our work, as a popular yet restrictive cross-covariance model [11].

Multi-output GPs are found in finance as well. Just like in geostatistics, complicated interactions between several variables over time can be model led with ease by relying on statistical characterizations through realistic covariance functions. For instance, foreign exchange and commodity data can be used to re-create artificially removed values the value of other commodities over time (Figure 1.2). We will revisit this example in our work.

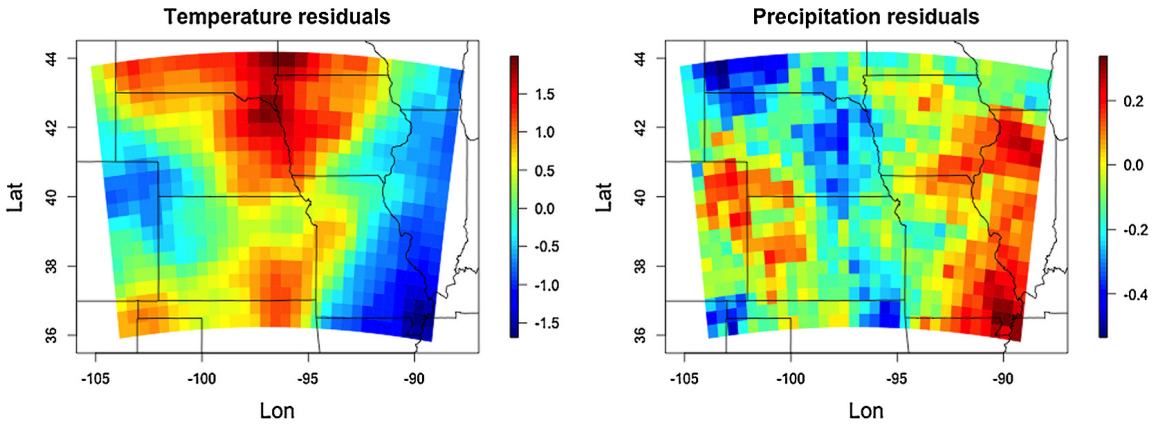
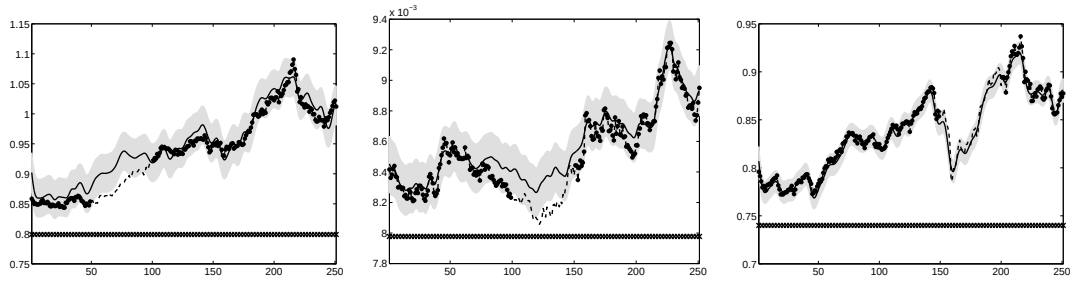


Figure 1.1: This is Figure 1, replicated from Genton and Kleiber's analysis of cross-covariance functions in geostatistics [11]. Here, the authors depict residuals for average summer temperature and precipitation prediction for the year 1989, which are outputs whose regression is simultaneously learned in a multi-output GP climate model. Units in the left and right diagrams are degrees Celsius and centimeters, respectively.

With the growing Internet of Things and large sensor network deployments, one may find use cases for GPs in data imputation with error bars and outlier detection, as GPs offer a probabilistic model. Again, the use of multi-output GPs is crucial, since information used across sensors can detect when data for a particular sensor is misleading (Figure 1.3).

As a final indicator of the utility of multi-GPs, we turn the reader's attention to medical diagnostics. Such models are used to both impute and predict lab covariates for medical patients—these predictions can in turn allow earlier and more accurate diagnosis of diseases such as sepsis [6].



(a) CAD (b) JPY (c) AUD

Figure 1.2: This is Figure 4, replicated from Alvarez et al, depicting two foreign currency exchanges CAD and JPY and a precious metal commodity AUD [1]. Test data was held out in these indices, and training data included other indices which are not shown. True values are dotted, multi-output GP predictions are solid, and a two standard deviation error bar is greyed out. The GP learning method used here relies on inducing points, which form an approximation for the real GP model by sampling the covariance kernel at the inducing points, depicted as crosses on the horizontal axis. Notice the AUD reconstruction indicates that a multiple-output approach may have predictive power.

1.1.2 Lack of Existing Methods

Medical covariate prediction was particularly inspiring for this project, since it dealt with multi-output GP model learning at a large scale with many data points [6]. With each patient needing their own model, the bottleneck in the research pipeline was training GP models under various different Bayesian prior configurations. This work hopes to address use cases such as this one, where training multi-output GPs is the bottleneck.

In particular, we will focus on a specific family of multi-output GPs, but nonetheless one that has found frequent use in the literature; namely, the linear model of

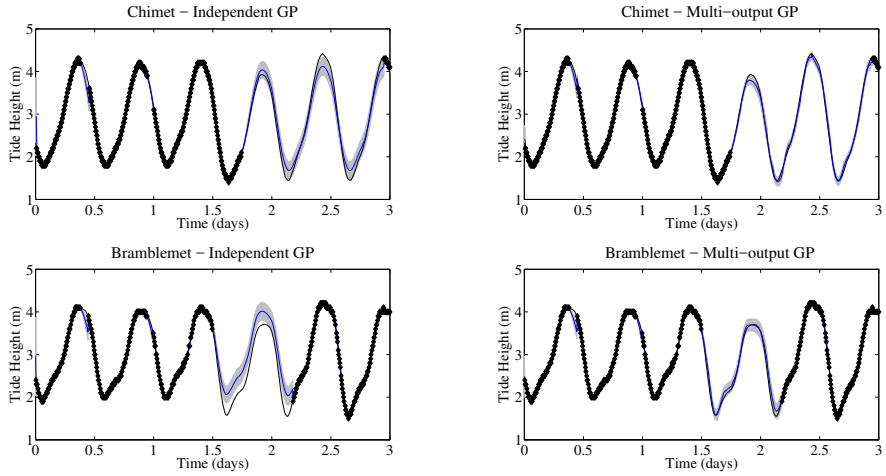


Figure 1.3: We replicate Figure 3 from Osborne et al, who demonstrate the increased accuracy from a single-output GP to a multi-output one in predicting tide heights over time from two sensors Bramblemet and Chimet on the south coast of England [22]. The solid black line represents the actual held-out data, the dots represent the training data. The blue line is the expectation from the GP model and the grey shaded region is the 2 standard deviation error bar.

coregionalization. Even this simple cross-covariance model violates stationarity, a key kernel property usually exploited for fast training in single-output GPs.

Falling back to an exact GP model comes at a cost: to compute the gradients for kernel hyperparameters, an exact computation requires an $O(n^3)$ decomposition of an $O(n^2)$ matrix in memory for n samples [25]. Each hyperparameter's scalar partial derivative then requires another $O(n^2)$ computation. While these operations may be performed in parallel, the memory use and runtime is prohibitive for problems with a large number of outputs, since the number of hyperparameters grows accordingly. For this reason, an accurate but fast approximate approach has been an important goal of machine learning research.

1.2 Gaussian Processes

1.2.1 Single-output Gaussian Processes

First, we give an overview of the theoretical foundation for GPs. A GP is a set of marginally normal random variables (RVs) $\{f_{\mathbf{x}}\}_{\mathbf{x}}$ indexed by $\mathbf{x} \in \mathcal{X}$, with the property that for any finite collection $X \subset \mathcal{X}$, the RVs are jointly Gaussian with a prescribed mean function $\boldsymbol{\mu} : \mathcal{X} \to \mathbb{R}$ and covariance function (kernel) $K : \mathcal{X}^2 \to \mathbb{R}$, i.e., $f_X \sim N(\boldsymbol{\mu}_X, K_{X,X})$ [29]. K is parameterized by a set of parameters $\boldsymbol{\theta}$; in sections of our work where we would like to emphasize a particular parameterization we will use $K_{|\boldsymbol{\theta}}$. As usual, we drop the mean with $\mathbf{y} \triangleq f_X - \boldsymbol{\mu}_X$. We introduce the notation vectorization $f_X = \left(f_{\mathbf{x}}\right)_{\mathbf{x} \in X}$ or matricization $K_{X,Z} = \left(K(\mathbf{x}, \mathbf{z})\right)_{\mathbf{x} \in X, \mathbf{z} \in Z}$ with subscripts.

Given a set of n observations at each X of the $\mathbb{R}^n$-valued RV $\mathbf{y}$, the aforementioned model assumes:

$$\mathbf{y} \sim N(\mathbf{0}, K_{X,X}).$$

For a fixed index $* \in \mathcal{X}$ of and corresponding $\mathbb{R}$-valued RV y_*, we have a corresponding property:

$$\begin{pmatrix} \mathbf{y} \\ y_* \end{pmatrix} \sim N\left(\mathbf{0}, \begin{pmatrix} K_{X,X} & K_{X,*} \\ K_{*,X} & K_{*,*} \end{pmatrix}\right)$$

If we fix a particular set of observations of $\mathbf{y}$ at X, then inference of y_* at any query index $*$ is performed by marginalization [25]:

$$y_* | \mathbf{y} \sim N\left(K_{*,X} K_{X,X}^{-1} \mathbf{y}, K_{*,*} - K_{*,X} K_{X,X}^{-1} K_{X,*}\right).$$

Accuracy is often sensitive to a particular parameterization of our kernel, so model estimation is performed by maximizing data log likelihood with respect to parameters $\boldsymbol{\theta}$ of K:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \log p(\mathbf{f}_X | X, \boldsymbol{\theta}) \\ &= -\frac{1}{2}\mathbf{y}^\top K_{|\boldsymbol{\theta}}^{-1}\mathbf{y} - \frac{1}{2}\log\left|K_{|\boldsymbol{\theta}}\right| - \frac{n}{2}\log 2\pi.\end{aligned} \tag{1.1}$$

As such, the heart of GP learning lies in optimization of $\mathcal{L}$. Gradient-based optimization methods then require the gradient with respect to every parameter θ_j using $\boldsymbol{\alpha} \triangleq K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$:

$$\partial_{\theta_j}\mathcal{L} = \frac{1}{2}\boldsymbol{\alpha}^\top \partial_{\theta_j} K_{|\boldsymbol{\theta}}\boldsymbol{\alpha} - \frac{1}{2}\operatorname{tr}\left(K_{|\boldsymbol{\theta}}^{-1}\partial_{\theta_j}K_{|\boldsymbol{\theta}}\right). \tag{1.2}$$

1.2.2 Multiple-output Gaussian Processes

We build multi-output GP models as instances of general GPs, where a multi-output GP model identifies correlations between outputs through a shared input space [2].

Here, for D outputs, we write our indexing set as $\mathcal{X}' = [D] \times \mathcal{X}$: an input is in a point from a shared domain coupled with an output tag. Then, if we make observations at $X_d \subset \mathcal{X}$ for each individual output $d \in [D]$, we can set the global set of inputs across all the outputs in our multiple regression as:

$$\mathbf{X} = \{(d, x) \,|\, d \in [D], x \in X_d\} \subset \mathcal{X}';\;\; n = |\mathbf{X}|\,.$$

An LMC kernel K is of the form

$$K([i, \mathbf{x}_i], [j, \mathbf{x}_j]) = \sum_{q=1}^{Q} b_{ij}^{(q)} k_q(\|\mathbf{x}_i - \mathbf{x}_j\|) + \epsilon_i \delta_{ij}, \tag{1.3}$$

where $k_q : \mathbb{R} \to \mathbb{R}$ is a stationary kernel on $\mathcal{X}$. Typically, the positive semi-definite (PSD) matrices $B_q \in \mathbb{R}^{D \times D}$ formed by $b_{ij}^{(q)}$ are parameterized as $A_q A_q^\top + \boldsymbol{\kappa}_q I_D$, with $A_q \in \mathbb{R}^{D \times R_q}, \boldsymbol{\kappa}_q \in \mathbb{R}_+^D$ and R_q a preset rank. The entries of these matrices as well as domain-specific parameters of each k_q comprise the parameter vector $\boldsymbol{\theta}$.

Importantly, even though each k_q is stationary, K is only stationary on $\mathcal{X}'$ if B_q is Toeplitz. Settings of B_q that arise in practice, where we wish to capture covariance across outputs as an arbitrary D^2-dimensional latent process, are not Toeplitz. Therefore, these matrices prevent stationarity in K and therefore prohibit direct application of typical methods of accelerating GP learning, such as structured kernel interpolation.

1.3 Contributions

Variational approaches that work on $m \ll n$ inducing points, or locations at which the true kernel is sampled, let users trade off accuracy for efficiency [17].

Nguyen and Bonilla observed in Collaborative Multi-output Gaussian Processes (COGP) that multi-output GP algorithms can further share an internal representation of the covariance structure among all outputs at once [21]. We analogously exploit sharing in the covariance structure.

Our approach makes the following contributions to approximate inference in multi-output GPs. First, we identify a block-Toeplitz structure induced by the linear model of coregionalization (LMC) kernel on a grid. Next, we adapt a previously identified kernel structure based on Semiparametric Latent Factor Models (SLFM) to the multi-output setting [27]. Both of these structures coordinate for fast matrix-vector multiplication $K\mathbf{y}$ with the covariance matrix K, which we demonstrate has better asymptotic complexity for $\nabla\mathcal{L}$. Moreover, these guarantees are realized in practice.

When inputs do not lie on a uniform grid (i.e., for time series, the length of time between observations differs), we apply a technique proposed in Massively Scalable Gaussian Processes (MSGP) [30]. For single-output kernels, MSGP leverages the structured kernel interpolation (SKI) framework to make inputs act as if they are on a grid [32]. In a single-output case, this opens up the possibility for a matrix-free GP, avoiding construction of the explicit Gram covariance matrix. Because LMC kernels exhibit the previously-identified block-Toeplitz structure, they harmonize with SKI. While direct application of SKI is not possible, we can make an extension that makes matrix-free LMC kernel learning viable even on non-grid inputs.

1.4 Outline

This work is organized as follows. The next section, Section 2, gives an extensive overview of currently available methods for GPs. We will review hierarchical constructions (Section 2.1), which we describe as not viable for multi-output GPs. Next,

we provide an overview of inducing point approaches (Section 2.2), which are useful for understanding the methods that we build upon and those that are an essential background to the current state-of-the-art multi-output GP method, COGP. Finally, we cover interpolating point approaches in Section 2.3, which are also essential pre-requisites for our method.

Section 3 then describes the LLGP model learning method in detail and Section 4 provides its performance on both synthetic and real benchmarks. In particular, Section 4.3 compares LLGP to COGP on financial data for foreign exchange returns in 2007 and Section 4.4 compares LLGP to COGP on weather data imputation.

Finally, we conclude in Section 5, discussing the implications of our work and future directions to explore.

Chapter 2

Related Work

Gaussian Process (GP) model learning is the procedure of modifying the kernel hyperparameters $\boldsymbol{\theta}$ to maximize the data log likelihood (Equation 1.1). Sometimes, hyperpriors to the parameters $\boldsymbol{\theta}$ are added, so that maximizing the data log likelihood becomes a problem of maximizing the posterior density. Even with such additions, the likelihood, Equation 1.1, and the expression for its gradients, Equation 1.2, require the $n \times n$ matrix inversion $K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$, where n is the number of data points, $K_{|\boldsymbol{\theta}}$ is the kernel with the current hyperparameters, and $\mathbf{y}$ are the output observations.

We review existing approaches for approximations of the likelihood $\mathcal{L}$, which accelerate GP learning because at each step in the GP log likelihood optimization the approximations to the terms in the two aforementioned equations take less time and space to compute.

2.1 Hierarchical Approaches

2.1.1 Linear Conjugate Gradients

Hierarchical approaches, like the interpolating approaches that will be explored in Section 2.3, build off a method for solving linear systems with large matrices.

In the context of GPs, Gibbs and MacKay noticed that it suffices to solve the linear system $K_{|\boldsymbol{\theta}}\mathbf{z} = \mathbf{y}$ for $\mathbf{z}$ in an iterative manner, through successive matrix-vector multiplications (MVMs) with by the matrix $K_{|\boldsymbol{\theta}}$ [12]. The authors used *linear conjugate-gradient* (LCG) descent to solve the linear system. LCG has several amiable properties. First, its stability implies that even with increasing error over iterations through an approximation of the Gram matrix, LCG will converge to a solution of desired accuracy [26].

In effect, LCG reduces the problem of covariance matrix inversion to finding a sparse representation of $K_{|\boldsymbol{\theta}}$ that is fast to multiply with. In the case of [12], the benefits of LCG were convincing enough that it was used with dense matrices.

As put by briefly by [28], the k-th iteration of LCG yields a minimizer of $\left\|\mathbf{y} - K_{|\boldsymbol{\theta}}\mathbf{z}\right\|^2$ with $\mathbf{z}$ constrained to the subspace $\text{span}\{\mathbf{z}_0, K_{|\boldsymbol{\theta}}\mathbf{z}_0, \cdots, K^k_{|\boldsymbol{\theta}}\mathbf{z}_0\}$, where $\mathbf{z}_0$ is some initial guess. With exact arithmetic, this would converge in n iterations, since $K_{|\boldsymbol{\theta}}$ is non-singular and the span of the n-th subspace would be $\mathbb{R}^n$.

2.1.2 Tree-structured Kernels

Tree-structured kernels operate by assuming that the kernel k is stationary and decreasing as the distance between points gets larger. Then, the matrix entries of

the kernel are fully determined by the pairwise Euclidean distance matrix induced by the data points.

Tree-structured kernel approaches build KD trees [28]; this type of work can be traced back to a general approach of kernel density estimation using dual trees [15]. The KD tree is built over the input space [28]. Each node of the KD tree carries a centroid representing the subtree. The i-th element of $K_{|\boldsymbol{\theta}}\mathbf{z}$, $\sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j) z_j$, can then be computed by a smaller weighted sum $\sum_k k(\mathbf{x}_i, \mathbf{c}_k) z'_k$ from centroids $\mathbf{c}_k$ and an induced weighing $\mathbf{z}'$. The centroids and weighing chosen to represent the dataset are, critically, selected to be close to the point $\mathbf{x}_i$. Here, we see the stationarity assumption at work—the adaptive kd-tree structure finds a vicinity of $\mathbf{x}_i$, and chooses many centroids near there, but the contribution of points far away from $\mathbf{x}_i$ are more crudely summarized by fewer centroids. Altogether, we get an accurate reconstruction of the kernel for stationary kernels that also decrease as the distance between the points the kernel is being evaluated on increases.

2.1.3 Tree-structured Covariance

Along a similar vein, one form acceleration directly speeds up the computation of the inverse matrix by applying hierarchical inversion methods. For matrices of the form $\sigma^2 I + K$ for low-rank K that are also Hierarchical Off-Diagonal Low-Rank (HODLR) $n \times n$ matrices of logarithmic depth, which are formed by common kernels, can be inverted in $O(n \log^2 n)$ time [3].

2.1.4 Inappropriateness of Tree-based Methods for LMC

Unfortunately, in the LMC context, the kernel (Equation 1.3) is not stationary. On the subspace of the input between two fixed outputs, it is stationary if all component kernels k_q are. In this case, one could apply these approximations to matrix blocks corresponding to stationary subspaces.

Nonetheless, neither KD-methods nor HODLR directly apply. We can explain the failure of these methods on more complex multi-output or non-stationary kernels kernels by noting the efficacy of these methods depends on the extent to which summaries of points (entries in the Gram matrix) far away from a query point are accurate representations for their whole cluster. In other words, these methods tacitly require a monotonic, isotropic, and fairly smooth kernel [20]:

$$\alpha_1 k(\mathbf{x}_*, \mathbf{x}_1) + \alpha_2 k(\mathbf{x}_*, \mathbf{x}_2) \approx (\alpha_1 + \alpha_2) k\left(\mathbf{x}_*, \frac{\mathbf{x}_1 + \mathbf{x}_2}{2}\right)$$

Even for a single-output GP with a periodic, stationary kernel such as $k(\mathbf{x}, \mathbf{z}) = \sin \exp - \|\mathbf{x} - \mathbf{z}\|^2$ the above does not hold.

2.2 Inducing Points

To cope with the lack of tractable GP inference methods, inducing point approaches create a tractable model to use instead of the classic GP. Such approaches fix or estimate inducing points $\mathbf{U}$ and claim that the data f_X is conditionally deterministic (deterministic training conditional, or DTC), independent (fully independent train-

ing conditional, or FITC), or partially independent (partially independent training conditional, or PITC) given random variables (RVs) $f_{\mathbf{U}}$ [23]. These approaches are agnostic to kernel stationarity, and their quality can be improved by increasing the number of inducing points at the cost of a longer runtime. Setting the inducing points $\mathbf{U} = \mathbf{X}$ recovers the original exact GP.

2.2.1 A New Graphical Model

In a review paper, Quiñonero-Candela and Rasmussen define sparse approximation methods over GPs as methods which make a conditional independence assumption over the training and test GP function values [23]:

$$p(f_*, \mathbf{f}_X) \approx q(f_*, f) = \mathbb{E}_{\mathbf{u}} \left(q(f_*|\mathbf{u}) q(\mathbf{f}_X|\mathbf{u}) \right)$$

Here, $\mathbf{u}$ is a latent rv, with $\mathbf{u} \sim \mathcal{N}(0, K_{U,U})$ for a set U of values, called inducing points, where $m = |U|$ is usually smaller than $|X|$. Assumptions additional to the conditional independence above determine the inducing point method.

Taking $q(\cdot|\mathbf{u}) = \mathcal{N}(K_{\cdot,\mathbf{u}} K_{U,U}^{-1} \mathbf{u}, \operatorname{diag}(K_{\cdot,\cdot} - Q_{\cdot,\cdot})$ with $Q_{X,Z} = K_{X,U} K_{U,U}^{-1} K_{U,Z}$ for $\cdot$ replaced with both f_* and $\mathbf{f}_X$ gives rise to the FITC prior. It is equivalent to using the kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} k(\mathbf{x}_i, \mathbf{x}_j) & i = j \\ Q_{\mathbf{x}_i, \mathbf{x}_j} & \text{o/w} \end{cases}$$

Using the exact kernel for the diagonal maintains covariance rank. This is important to make sure that the span of basis functions for our GP is sufficiently large;

otherwise, our GP may not be expressive enough (it would only have the rank of $K_{U,U}$) [23].

In effect, the smaller latent space of $\mathbf{u}$ is used to represent the covariances. In our context, the shared $\mathcal{X}'$ space looks tempting to use for $\mathbf{u}$, but it doesn't contain enough information to capture cross-covariances between the multiple outputs, and will require some modification.

In the single output context, for m inducing points, the MVM is reduced to an $O(nm^2)$ operation, and storage decreases to $O(nm)$ from n^2.

Other inducing point approaches are either similar to FITC, such as the partially independent conditional or the sparse multiscale GP. Furthermore, some use other inducing-point methods, such as sparse spectrum Gaussian Processes (SSGP), but these require stationarity [24].

2.2.2 Collaborative Multi-output Gaussian Processes

By sharing the m inducing points across all outputs, Collaborative Multi-output Gaussian Processes (COGP) improves runtime per iteration to $O(m^3)$. Moreover, COGP only requires $O(Dnm)$ memory [21]. To ensure that $O(m^3)$ is computationally tractable for COGP, m must be sufficiently small. COGP makes large multi-output GP estimation feasible through a variational approximation, the evidence lower bound, to the log likelihood.

2.3 Interpolation Points

2.3.1 Structured Covariance Matrices

If we can identify structure in the covariance matrices K, then we can recover fast in-memory representations and efficient matrix-vector multiplications (MVMs) for finding products $K\mathbf{z}$, which will prove to be the fundamental computational operation of GP inference.

The Kronecker product $A \otimes B$ of matrices of order a, b is a block matrix of order ab with ij-th block $A_{ij}B$. We can represent it by keeping representations of A and B separately, rather than the product. Then, the corresponding MVMs can be computed in time $O(a\text{MVM}(B) + b\text{MVM}(A))$, where $\text{MVM}(\cdot)$ is the runtime of a MVM. For GPs on uniform dimension-P grids, this approximately reduces the runtime and representation costs by a $(1/P)$-th power [13].

Symmetric Toeplitz matrices T are constant along their diagonal and fully determined by their top row $\{T_{1j}\}_j$, yielding an $O(n)$ representation. Such matrices arise naturally when we examine the covariance matrix induced by a stationary kernel k applied to a one-dimensional grid of inputs. Since the difference in adjacent inputs $t_{i+1} - t_i$ is the same for all i, we have the Toeplitz property that:

$$T_{(i+1)(j+1)} = k(|t_{i+1} - t_{j+1}|) = k(|t_i - t_j|) = T_{ij}$$

Furthermore, we can embed T in the upper-left corner of a circulant matrix C of twice its size, which enables MVMs $C \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} T\mathbf{x} \\ \mathbf{0} \end{pmatrix}$ in $O(n \log n)$ time. This approach has

been used for fast inference in single-output GP time series with uniformly spaced inputs [7].

2.3.2 Structured Kernel Interpolation

SKI abandons the inducing-point approach: instead of using an intrinsically sparse model, SKI approximates the original $K_{X,X}$ directly [32]. To do this efficiently, SKI relies on the differentiability of K. For $\mathbf{x}, \mathbf{z}$ within a grid U, $|U| = m$, and $W_{\mathbf{x},U} \in \mathbb{R}^{1\times m}$ as the cubic interpolation weights [19], $|K_{\mathbf{x},\mathbf{z}} - W_{\mathbf{x},U}K_{U,\mathbf{z}}| = O(m^{-3})$. The simultaneous interpolation $W \triangleq W_{X,U} \in \mathbb{R}^{n\times m}$ then yields the SKI approximation: $K_{X,X} \approx WK_{U,U}W^\top$. Importantly, W has only $4^d n$ nonzero entries, with $\mathcal{X} = \mathbb{R}^d$.

2.3.3 Massively Scalable Gaussian Processes

Massively Scalable Gaussian Processes (MSGP) observes that the kernel $K_{U,U}$ on a grid exhibits Kronecker and Toeplitz matrix structure [30]. Drawing on previous work on structured GPs [7, 13], MSGP uses linear conjugate gradient descent as a method for evaluating $K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$ efficiently for Equation 1.1. In addition, [31] mentions an efficient eigendecomposition that carries over to the SKI kernel for the remaining $\log\left|K_{|\boldsymbol{\theta}}\right|$ term in Equation 1.1.

While evaluating $\log\left|K_{|\boldsymbol{\theta}}\right|$ is not feasible in the LMC setting (because the LMC sum breaks Kronecker and Toeplitz structure), the general notion of creating structure with SKI carries over to LLGP.

Chapter 3

Methods

We propose a linear model of coregionalization (LMC) method based on recent structure-based optimizations for Gaussian Process (GP) estimation instead of variational approaches. Critically, the accuracy of the method need not be reduced by keeping m low because its reliance on structure allows better asymptotic performance. For simplicity, our work focuses on multi-dimensional outputs, one-dimensional inputs, and Gaussian likelihoods.

3.1 Matrix-free GP Learning

As discussed in Section 2.1.1, Gibbs and MacKay describe the algorithm for GP model learning in terms of only matrix-vector multiplications (MVMs) with the covariance matrix [12]. Critically, we cannot access $\mathcal{L}$ itself, only $\nabla\mathcal{L}$, so we choose AdaDelta as the high-level optimization routine [33]. Algorithm 1 summarizes the high-level operation of LLGP.

Algorithm 1 The content corresponding to each line is detailed below. $\mathbf{X} \subset [D] \times \mathbb{R}$ is the set of inputs, tagged by the output that they correspond to, and limited for simplicity to the one-dimensional case. Then $\mathbf{y} \subset \mathbb{R}$ is the corresponding set of outputs. AdaDeltaUpdate is the update function for AdaDelta—we abstract away auxiliary AdaDelta variables.

1: **procedure** LLGP($\alpha, \mathbf{X}, \mathbf{y}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$ ▷ Initialization; Section 4.1
3: $\quad g_{\max} = -\infty$
4: $\quad$**repeat**
5: $\quad\quad$Construct a linear operator $\tilde{K}_{|\boldsymbol{\theta}}$ from $\mathbf{X}, \mathbf{y}, \boldsymbol{\theta}$. ▷ MVM operator
6: $\quad\quad g \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\tilde{K}_{|\boldsymbol{\theta}}}$ ▷ Gradients from an operator, Algorithm 2
7: $\quad\quad \boldsymbol{\theta} \leftarrow \text{AdaDeltaUpdate}(\boldsymbol{\theta}, g)$
8: $\quad\quad g_{\max} = \max(\|g\|, \|g_{\max}\|)$
9: $\quad$**until** $\|g\| \leq \alpha g_{\max}$ ▷ Cutoff; Section 3.4
10: $\quad$**return** $\boldsymbol{\theta}$
11: **end procedure**

For every given $\boldsymbol{\theta}$, we construct an operator $\tilde{K}_{|\boldsymbol{\theta}}$ in Line 5 of Algorithm 1. This operator is built so that it approximates MVM with the true kernel; i.e., $K_{|\boldsymbol{\theta}}\mathbf{z} \approx \tilde{K}_{|\boldsymbol{\theta}}\mathbf{z}$. However, we make an approximation such that the multiplication is fast. This is discussed in Section 3.3.

Since $\mathcal{L}$ is continuously differentiable, its gradient under an approximation of the kernel is approximately the gradient of the parameters with respect to the exact kernel. Using only the MVM operation, we compute the gradient of the log-likelihood, discussed in Section 3.2.

3.2 Computing the Gradient

Given an operator $K_{|\boldsymbol{\theta}}$ enabling matrix-vector multiplication with a positive definite kernel, we find $\nabla\mathcal{L}$ of Equation 1.2, replicated below. Recall that θ_j is the j-th term

of the kernel parameters $\boldsymbol{\theta}$, $\boldsymbol{\alpha} = K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$, and $\mathbf{y}$ are the observed outputs.

$$\partial_{\theta_j}\mathcal{L} = \frac{1}{2}\boldsymbol{\alpha}^\top \partial_{\theta_j} K_{|\boldsymbol{\theta}}\boldsymbol{\alpha} - \frac{1}{2}\operatorname{tr}\left(K_{|\boldsymbol{\theta}}^{-1}\partial_{\theta_j} K_{|\boldsymbol{\theta}}\right).$$

To compute any inverse $K_{|\boldsymbol{\theta}}^{-1}\mathbf{z}$, we depart from Gibbs and MacKay in selecting MINRES instead of LCG as the Krylov-subspace inversion method used to compute inverses from MVMs. MINRES handles numerically semidefinite matrices with more grace [10]. This is essential in GP optimization, where the diagonal noise matrix $\boldsymbol{\epsilon}$, iid for each output, shrinks over the course of learning, making inversion-based methods require additional iterations (Figure 3.1).

To compute the trace term in Equation 1.2, we rely on Gibbs and MacKay's stochastic approximation by introducing the random variable $\mathbf{r}$ with $\operatorname{cov}\mathbf{r} = I$:

$$\operatorname{tr}\left(K_{|\boldsymbol{\theta}}^{-1}\partial_{\theta_j} K_{|\boldsymbol{\theta}}\right) = \mathbb{E}\left[(K_{|\boldsymbol{\theta}}^{-1}\mathbf{r})^\top \partial_{\theta_j} K_{|\boldsymbol{\theta}}\mathbf{r}\right] \text{[12]}. \tag{3.1}$$

For this approximation, the number of samples need not be large, and the estimate improves as the size of $K_{|\boldsymbol{\theta}}$ increases. As in other work [8], we let $\mathbf{r} \sim \text{Unif}\{\pm 1\}$.

If we sample N_t instances of $\mathbf{r}$, every iteration needs to perform $N_t + 1$ inversions including $\mathbf{y}$, if we reuse the same samples $K_{|\boldsymbol{\theta}}^{-1}\mathbf{r}$ between derivatives ∂_{θ_j}. These inversions are done in parallel. Then, since $\partial_{\theta_j} K_{|\boldsymbol{\theta}}$ is both structured and smaller than $K_{|\boldsymbol{\theta}}$, we can perform MVMs with the matrix operator $\partial_{\theta_j} K_{|\boldsymbol{\theta}}$ efficiently. Altogether, Line 11 of Algorithm 2 requires $\tilde{O}(n)$ time to compute; this goes for the entire inner loop of between lines 8 and 14 for finding $\partial_{\theta_j}\mathcal{L}$. This procedure is summarized in Algorithm 2.

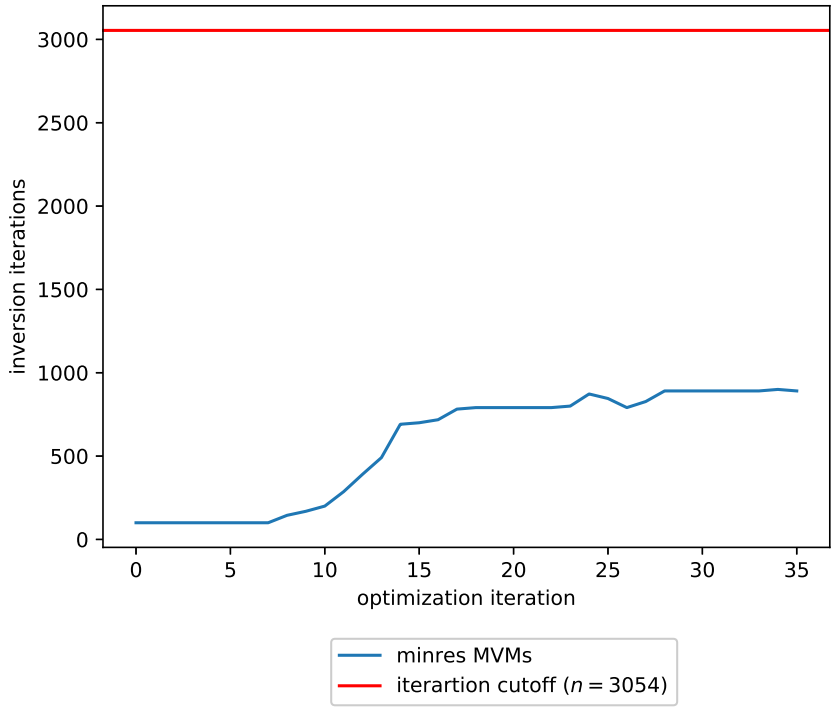


Figure 3.1: Number of MVMs that MINRES must perform at each optimization iteration for a GP applied to the dataset in Section 4.3. The iteration cutoff is the number of training points n in the dataset.

Overall, the work is bottlenecked in the inversions $K_{|\boldsymbol{\theta}}^{-1}\mathbf{r}, K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$, which in turn rely through MINRES on the MVM operation, discussed in Section 3.3. Since $K_{|\boldsymbol{\theta}}$ only enters computation as an MVM operator, the amount of memory consumed is dictated by its representation, which need not be dense.

Algorithm 2 Given a linear MVM operator for a kernel, we compute the gradient of its data log likelihood. N_t is a fixed parameter for our stochastic trace approximation. The output of this procedure is $\nabla\mathcal{L}$. MINRES$(K, \mathbf{z})$ computes $K^{-1}\mathbf{z}$.

procedure GRADIENT(K, $\mathbf{y}$)
  $R \leftarrow \{\mathbf{r}_i\}_{i=1}^{N_t}$, sampling $\mathbf{r} \sim \text{Unif}\{\pm 1\}$.
  for $\mathbf{z}$ in $\{\mathbf{y}\} \cup R$, in parallel **do**
    Store the result of MINRES$(K, \mathbf{z})$ as $K^{-1}\mathbf{z}$.
  end for
  Let $\boldsymbol{\alpha} = K^{-1}\mathbf{y}$; computed before.
  $g \leftarrow \mathbf{0}$
  for θ_j in $\boldsymbol{\theta}$ **do** ▷ Compute $\partial_{\theta_j}\mathcal{L}$
    Create the operator $L = \partial_{\theta_j} K_{|\boldsymbol{\theta}}$
    $\{\mathbf{s}_i\}_{i=1}^{N_t} \leftarrow \{L(\mathbf{r}_i)\}_{i=1}^{N_t}$
    $t \leftarrow \frac{1}{N_t}\sum_{i=1}^{N_t}\left(K_{|\boldsymbol{\theta}}^{-1}\mathbf{r}_i\right)\cdot\mathbf{s}_i$ ▷ t approximates the trace term of Equation 3.1
    $r \leftarrow \boldsymbol{\alpha}\cdot L(\boldsymbol{\alpha})$
    $g_j \leftarrow \frac{1}{2}r - \frac{1}{2}t$ ▷ Equation 1.2
  end for
  return g
end procedure

3.3 Fast MVMs and Parsimonious Kernels

When LMC kernels are evaluated on a grid of points for each output, so $X_d = U$, the simultaneous covariance matrix equation without noise Equation 3.2 over $\mathbf{U}$ holds for Toeplitz matrices K_q formed by the stationary kernels k_q evaluated at pairs of $U \times U$.

$$K_{\mathbf{U},\mathbf{U}} = \sum_q (A_q A_q^\top + \operatorname{diag}\boldsymbol{\kappa}_q) \otimes K_q \tag{3.2}$$

The SLFM and COGP models correspond to all $\boldsymbol{\kappa}_q$ set to 0 and $A_q = \mathbf{a}_q \in \mathbb{R}^{D\times 1}$. Moreover, we include D additional kernels for the independent GP components,

which can be incorporated using kernels K_d where $A_d = 0$ and $\boldsymbol{\kappa}_d$ as the dth standard basis vector of $\mathbb{R}^D$ for $d \in [D]$. This shows a reduction from SLFM to LMC.

Recall Structured Kernel Interpolation (SKI) from Section 2.3.2 as a method of recovery of structure on unstructured inputs. In order to adapt SKI to our context of multiple outputs, we build grid $\mathbf{U} \subset \mathcal{X}'$ out of a common subgrid $U \subset \mathcal{X}$ that extends to all outputs with $\mathbf{U} = [D] \times U$. Since the LMC kernel evaluated between two sets of outputs K_{X_i,X_j} is differentiable, as long as U contains (in the sense of range) each $\{X_d\}_{d\in[D]}$, the corresponding SKI approximation $K_{\mathbf{X},\mathbf{X}} \approx W K_{\mathbf{U},\mathbf{U}} W^\top$ holds with the same asymptotic convergence cubic in $1/m$.

We build a corresponding approximation for the differentiable part of our kernel:

$$K_{\mathbf{X},\mathbf{X}} \approx W K_{\mathbf{U},\mathbf{U}} W^\top + \boldsymbol{\epsilon}. \tag{3.3}$$

We cannot fold $\boldsymbol{\epsilon}$ into the interpolated term $K_{\mathbf{U},\mathbf{U}}$ since it does not correspond to a differentiable kernel, so the SKI approximation fails. But this fact does not prevent efficient representation or multiplication since the matrix is diagonal. In particular, the MVM operation $K_{\mathbf{X},\mathbf{X}}\mathbf{x}$ can be approximated by $W K_{\mathbf{U},\mathbf{U}} W^\top \mathbf{x} + \boldsymbol{\epsilon}\mathbf{x}$, where matrix multiplications by the sparse matrices $\boldsymbol{\epsilon}, W, W^\top$ require $O(n)$ space and time.

We consider different representations of $K_{\mathbf{U},\mathbf{U}}$ from Equation 3.3 to reduce the memory and runtime overhead for performing the multiplication $K_{\mathbf{U},\mathbf{U}}\mathbf{z}$ (where we have computed $\mathbf{z} = W^\top \mathbf{x}$).

3.3.1 SUM: Sum Representation

In SUM, we represent $K_{\mathbf{U},\mathbf{U}}$ with a Q-length list. At each index q, B_q is a dense matrix of order D and K_q is a Toeplitz matrix of order m, where only its top row needs to be represented to perform MVMs.

In turn, multiplication $K_{\mathbf{U},\mathbf{U}}\mathbf{z}$ is performed by multiplying each matrix in the list with $\mathbf{z}$ and summing the results:

$$K_{\mathbf{X},\mathbf{X}}\mathbf{x} \approx W K_{\mathbf{U},\mathbf{u}} W^\top \mathbf{x} + \boldsymbol{\epsilon}\mathbf{x} = W \sum_q \left((A_q A_q^\top + \operatorname{diag} \boldsymbol{\kappa}_q) \otimes K_q \right) \mathbf{z} + \boldsymbol{\epsilon}\mathbf{x}$$

As described before, the Kronecker MVM $(B_q \otimes K_q)\mathbf{z}$ may be expressed as D fast Toeplitz MVMs with K_q and m dense MVMs with B_q. In turn, the runtime for each of the Q terms is $O(Dm \log m)$.

3.3.2 BT: Block-Toeplitz Representation

In BT, we notice that $K_{\mathbf{U},\mathbf{U}}$ is a block matrix with blocks T_{ij}:

$$\sum_q B_q \otimes K_q = \Big(T_{ij} \Big)_{i,j \in [D]^2}, \quad T_{ij} = \sum_q b_{ij}^{(q)} K_q.$$

On a one-dimensional grid U, these matrices are Toeplitz since they are linear combinations of Toeplitz matrices. BT requires D^2 m-sized rows to represent each T_{ij}. Then, using usual block matrix multiplication, an MVM $K_{\mathbf{U},\mathbf{U}}\mathbf{z}$ takes $O(D^2 m \log m)$ time.

3.3.3 SLFM: SLFM Representation

SLFM uses a rank-based representation. Let $R \triangleq \sum_q R_q/Q$ be the average added rank, $R \leq D$.

We first rewrite the grid kernel:

$$K_{U,U} = \sum_q \sum_{r=1}^{R_q} \mathbf{a}_q^{(r)} \mathbf{a}_q^{(r)\top} \otimes K_q + \sum_q \operatorname{diag} \boldsymbol{\kappa}_q \otimes K_q.$$

Note $\mathbf{a}_q^{(r)} \mathbf{a}_q^{(r)\top}$ is rank-1. Under some re-indexing $q' \in [RQ]$ which flattens the double sum such that each q' corresponds to a unique (r, q), the term $\sum_q \sum_{r=1}^{R_q} \mathbf{a}_q^{(r)} \mathbf{a}_q^{(r)\top} \otimes K_q$ may be rewritten as

$$\sum_{q'} \mathbf{a}_{q'} \mathbf{a}_{q'}^\top \otimes K_{q'} = \mathbf{A} \operatorname{blockdiag}_{q'} \left(K_{q'} \right) \mathbf{A}^\top;$$

where the second simplification has $\mathbf{A} = \left(\mathbf{a}_{q'} \right)_{q'} \otimes I_m$ with $\left(\mathbf{a}_{q'} \right)_{q'}$ a matrix of horizontally stacked column vectors [27]. Next, we rearrange the remaining term $\sum_q \operatorname{diag} \boldsymbol{\kappa}_q \otimes K_q$ as $\operatorname{blockdiag}_d(T_d)$, where $T_d = \sum_q \kappa_{qd} K_q$ is Toeplitz.

Thus, the SLFM representation writes $K_{\mathbf{U},\mathbf{U}}$ as the sum of two block diagonal matrices of block order QR and D, where each block is a Toeplitz order m matrix, so MVMs take $O((QR + D)m \log m)$ time.

3.3.4 Asymptotic Performance

Because the runtimes of BT and SLFM are complimentary in the sense that one performs better than the other when $D^2 > QR$ and vice-versa, an algorithm that

uses the aforementioned condition between to decide between which representation to use can minimize runtime. We also found that SUM is efficient in practice for $Q = 1$.

Using the switching condition, MVMs with the original matrix $K_{\mathbf{X},\mathbf{X}}$ altogether have a space and time upper bound of $\tilde{O}(\min(QR, D^2)m+n)$, where the min is earned thanks to the choice between different representations. Every AdaDelta iteration then takes $\tilde{O}(\sqrt{\kappa_2})$ such matrix-vector products for machine-precision answers, with κ_2 the spectral condition number [26].

On a grid of inputs with $\mathbf{X} = \mathbf{U}$, the SKI interpolation vanishes with $W = I$. In this case, using BT alone leads to a faster algorithm—applying the Chan block-Toeplitz preconditioner in a Krylov-subspace based routine has experimentally shown convergence using fewer MVMs [5].

3.4 Stopping Conditions

For a gradient-only stopping heuristic, we maintain the running maximum gradient ∞-norm. This is described explicitly in Algorithm 1. If gradient norms drop below a preset proportion of the running maximum norm more than a pre-set tolerance number of times, we terminate. For example, when applied to the foreign exchange rate prediction dataset in Section 4.3, the heuristic eventually notices that we have slowed down increases in $\mathcal{L}$ because the gradients occasionally drop below the threshold at that point, while not displacing the solution $\boldsymbol{\theta}$ significantly since we must be near a local minimum (Figure 3.2).

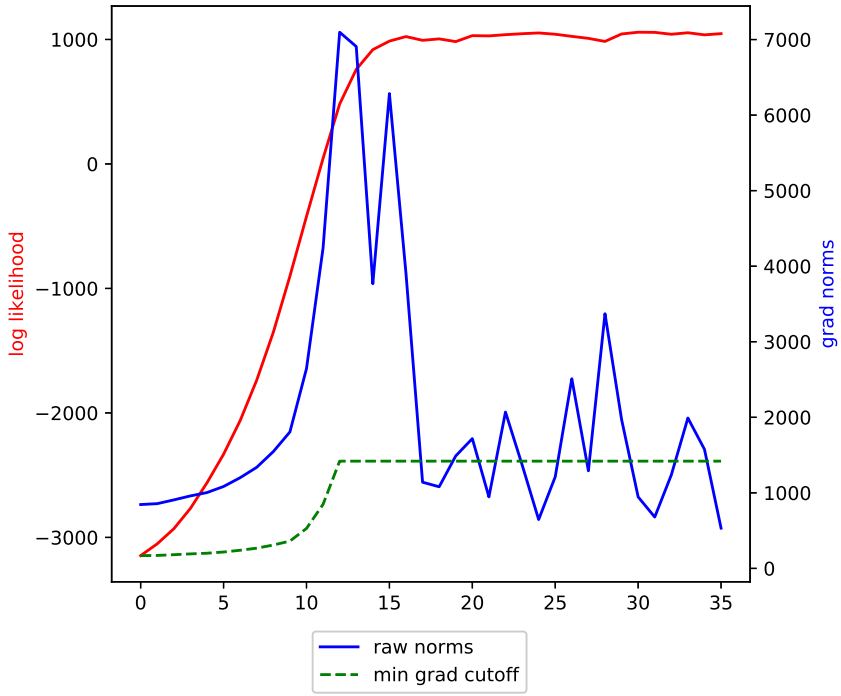


Figure 3.2: In green, we have 20% of the rolling maximum norm. In red and blue are $\mathcal{L}$ (computed exactly and therefore unavailable during benchmarks) and $\|\nabla\mathcal{L}\|_\infty$, respectively.

Chapter 4

Results

We evaluate the methods on held out data by using standardized mean square error (SMSE) of the test points with the predicted mean, and the negative log predictive density (NLPD) of the Gaussian likelihood of the inferred model. Notably, NLPD takes confidence into account, while SMSE only evaluates the mean prediction. In both cases, lower values represent better performance.

4.1 Implementation Details

4.1.1 Hyperparameter Settings

AdaDelta parameters were set to the following on all tests: rate = 1, decay = 0.9, momentum = 0.5, offset = 0.0001. The stopping criterion parameters permit the gradient ∞-norm to drop below a threshold of its maximum value so far a small, fixed number of times, 5. The maximum number of iterations was 100.

For learning, we initialize entries A_q according to a unit normal and all $\boldsymbol{\kappa}_q$ to 1. Note that COGP initializes the coregionalization matrix to 1 uniformly. Like COGP, we initialize noise to 0.1 for all outputs.

LLGP was implemented in Python 3 from the Anaconda, which offered an Intel MKL-linked scipy [18]. The code made heavy use of other packages, namely climin [4], GPy [14], and paramz [34]. Code and benchmarks are available at `https://github.com/vlad17/runlmc`.

Application of our approach to all replication studies was carried out on a large server in a multi-programming environment: CentOS 6.5 with 80 Intel(R) Xeon(R) CPU E7-4870 @ 2.40GHz. The representation evaluation benchmarks were done at once on a cluster of machines running CentOS 5.2-5.9 with Intel(R) Xeon(R) CPU E5430 @ 2.66GHz, where these jobs ran on a single thread per CPU.

4.2 Representation Evaluation

We evaluated the performance of the different kernel representations over various rank and parameterization settings. In particular, we have the following parameters: n total sample size across all outputs, D number of outputs, Q number of kernels k_q, R average added rank, ϵ mean noise, and `ktype` kernel type. Kernel type is one of `mat, periodic, rbf, mix` corresponding to Matérn-3/2, standard periodic[1], and radial basis functions. `mix` refers to a mix of all three kernels.

Each kernel's inverse length scales and periods were selected by sampling uniformly in log space from 1 to 10 with Q samples. Next, we construct a random

[1]We define the periodic kernel as $k(r) = \exp\left(\frac{-\gamma}{2}\sin^2\frac{\pi r}{T}\right)$.

LMC kernel by sampling entries of each A_q from a standard normal distribution, $\boldsymbol{\kappa}_q$ from an inverse gamma with unit shape and scale, and independent noise $\boldsymbol{\epsilon}$ for every output from an inverse gamma with unit scale and mean ϵ. Inputs and outputs were independently generated from Unif$[0, 1]$ for benchmarking.

As expected from their asymptotic runtime, SUM, BT, and SLFM representations are complimentary in matrix-vector multiplication (MVM) speed for different configurations of D, R, Q—this results in sparse inversion computation that consistently outperforms Cholesky decomposition in runtime (Table 4.1). For inverting systems, all computations were carried out until the residual ℓ_2 norm was at most 10^{-4}.

Table 4.1: The runtime in seconds for solving $K\mathbf{x} = \mathbf{y}$ for a random kernel K constructed as in Section 4.2 using MINRES for each of the kernel representations. For comparison, the CHOL representation is wallclock time to compute the Cholesky decomposition of the matrix, which must be constructed, and use this decomposition to invert the system. We averaged over five runs. In every run, we use $n = 5000$ simulated data points, `mix` kernels, and $\epsilon = 0.1$.

D	R	Q	CHOLESKY	SUM	BT	SLFM
2	2	10	40.45	40.04	**8.28**	45.07
10	1	10	37.60	34.51	21.93	**9.86**
10	10	1	9.59	**0.42**	2.42	0.90

We next evaluated the accuracy of the gradients for $N_t = 10$ trace samples. Fixing $R = 3, n = 5000, D = 10$, we quantified the accuracy and speed of computing $\nabla\mathcal{L}$. Since, for each partial derivative, LLGP requires only N_t n-sized vector dot products (Equation 3.1), it generally runs faster than the exact approach (Figure 4.2), which must compute a matrix-matrix dot product (Equation 1.2). The gradients between the two, however, are virtually indistinguishable for smooth kernels that

induce diagonally dominant covariance matrices (Figure 4.1). Kernels such as the single Matérn or periodic kernel with noise on the order of 10^{-4} lead to less accurate gradients, owing to poor MINRES convergence in the inversions (Figure 4.3). We will show that the stochastic gradients suffice for optimization in practical examples.

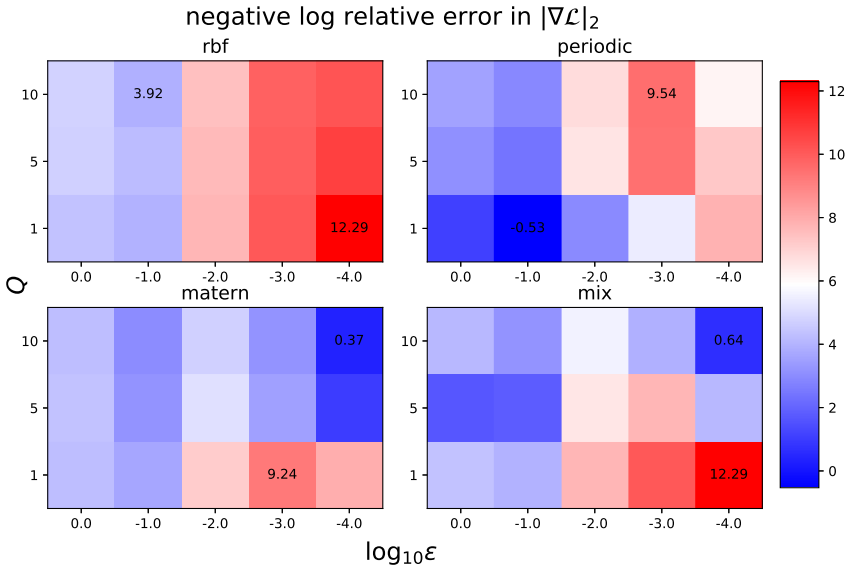


Figure 4.1: Negative logarithm of the relative error in ℓ_2 norm between exact and LLGP gradients. Higher is better, and extremal values are noted. For each data point, the average was taken over five runs.

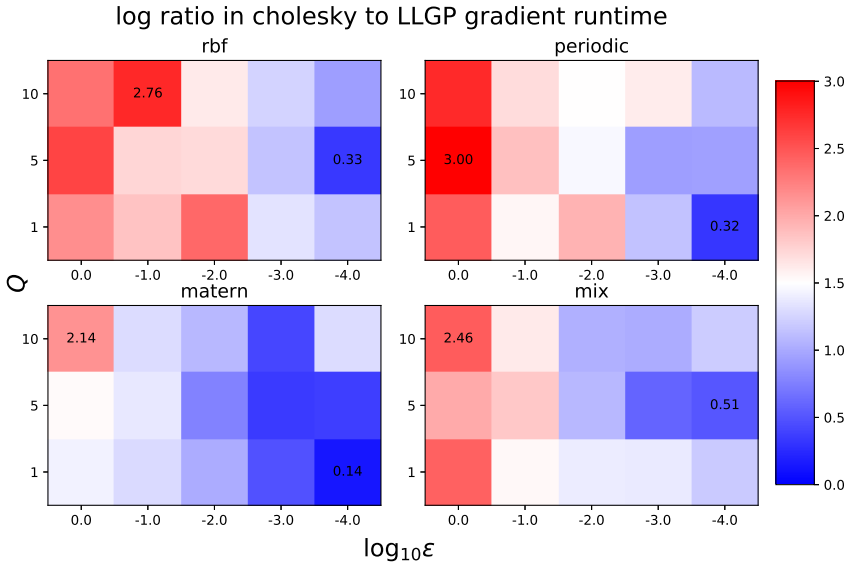


Figure 4.2: Logarithm of the ratio of time required by the exact approach to that of LLGP for computing the gradient over all parameters from Equation 1.2. Higher is better, and extremal values are noted. For each data point, the average was taken over five runs.

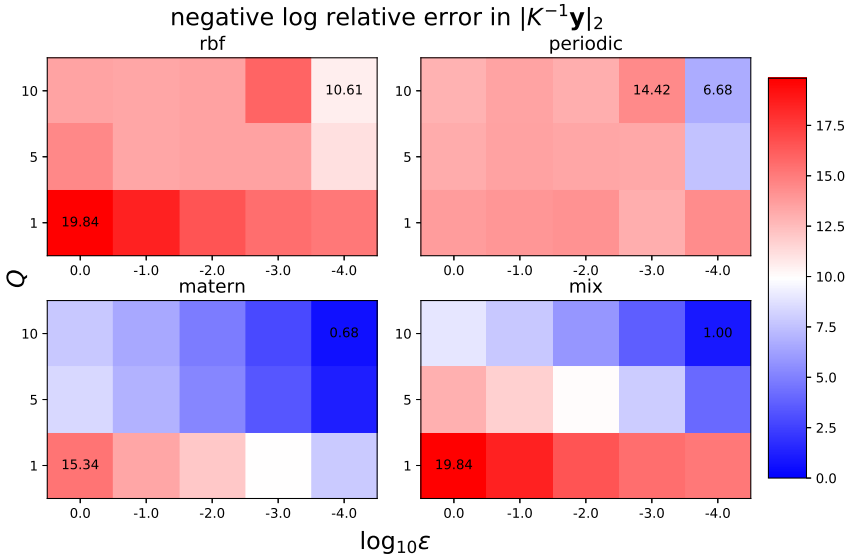


Figure 4.3: Negative logarithm of the relative error in ℓ_2 norm between exact and MINRES solutions to $K\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$. For each data point, the average was taken over five runs.

4.3 Foreign Exchange Rate Prediction

We replicate the medium-sized dataset from COGP as an application to evaluate LLGP performance. The dataset consists of ten foreign currency exchange rates—CAD, EUR, JPY, GBP, CHF, AUD, HKD, NZD, KRW, and MXN—and three precious metals—XAU, XAG, and XPT—implying that $D = 13$.[2] As in COGP, we retrieved the asset to USD rate, then used its reciprocal in all the results discussed below. The LLGP setting has $Q = 1, R = 2$, as recommended in [1] for LMC models on this dataset; let this be the LMC model on LLGP. COGP roughly corresponds to the the SLFM model, which has a total of 94 hyperparameters, compared to 53 for LLGP. All kernels are RBF.

The data used in this example are from 2007, and include $n = 3054$ training points and 150 test points. The test points include 50 contiguous points extracted from each of the CAD, JPY, and AUD exchanges.

For this application, LLGP uses $m = n/D = 238$ interpolating points. We use the COGP settings from the paper.[3] LLGP, for both LMC, outperforms COGP in terms of predictive mean and variance estimation as well as runtime (Table 4.2).

[2]Data are from `http://fx.sauder.ubc.ca/data.html`

[3]COGP hyperparameters for FX2007 were 100 inducing points, 500 iterations, 200 mini-batch size.

Table 4.2: Average predictive performance and training time over 10 runs for LLGP and COGP on the FX2007 dataset. Parenthesized values are standard error. LLGP was run with LMC set to $Q = 1$, $R = 2$, and 238 interpolating points. COGP used a $Q = 2$ kernel with 100 inducing points.

METRIC	LLGP	COGP
SECONDS	**64** (**8**)	296 (2)
SMSE	**0.21** (**0.01**)	0.26 (0.03)
NLPD	**-3.62** (**0.07**)	14.52 (3.10)

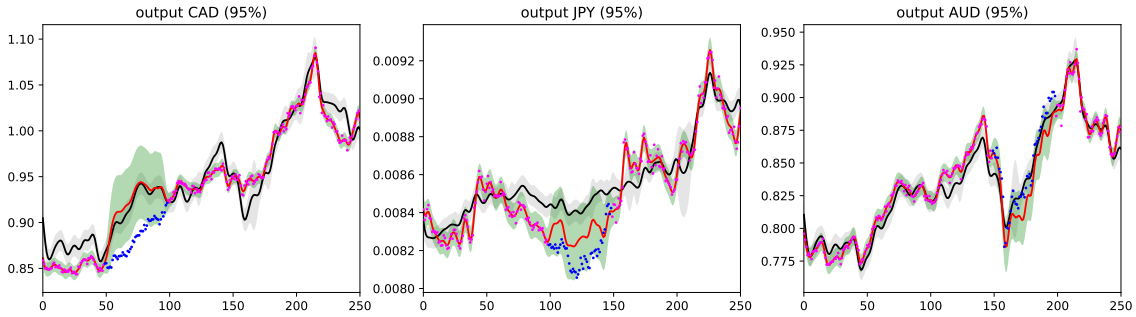


Figure 4.4: Test outputs for the FX2007 dataset. COGP mean is black, with 95% confidence intervals shaded in grey. LLGP mean is a solid red curve, with light green 95% confidence intervals. Magenta points are in the training set, while blue ones are in the test set. Notice LLGP variance corresponds to an appropriate level of uncertainty on the test set and certainty on the training set, as opposed to the uniform variance from COGP.

4.4 Weather Dataset

Next, we replicate results from a weather dataset, a large time series used to validate COGP. In this dataset, $D = 4$ weather sensors Bramblemet, Sotonmet, Cambermet, and Chimet record air temperature over five days in five minute intervals, with some dropped records due to equipment failure. Parts of Cambernet and Chimet are

dropped for imputation, yielding $n = 15789$ training measurements and 374 test measurements.

We use the COGP parameters that were set by default in the code provided by the authors.[4] LLGP was run with the same parameters as in FX2007, simulating the SLFM model. We tested LLGP models on different numbers of interpolating points m.

Table 4.3: Average predictive performance and training time over 10 runs for LLGP and COGP on the weather dataset. Parenthesized values are standard error. Both LLGP and COGP trained the SLFM model. We show LLGP with 500 and 1000 interpolating points and COGP with 200 inducing points.

Metric	LLGP $m = 500$	LLGP $m = 1000$	COGP
seconds	**60** (**14**)	259 (62)	1380 (12)
SMSE	0.09 (0.01)	0.09 (0.01)	**0.08** (**0.00**)
NLPD	2.14 (0.58)	**1.54** (**0.03**)	98.48 (1.30)

LLGP performed slightly worse than COGP in SMSE, but both NLPD and runtime indicate significant improvements (Table 4.3). Varying the number of interpolating points m from 500 to 1000 constructs a tradeoff frontier between increases in m and NLPD decrease at the cost of additional runtime (Figure 4.5). While NLPD improvement diminishes as m increases, LLGP is still an improvement compared to COGP for a range of m by an order of magnitude in runtime and almost two orders of magnitude for NLPD.

[4] `https://github.com/trungngv/cogp`, commit `3b07f621ff11838e89700cfb58d26ca39b119a35`. The weather dataset was run on 1500 iterations, mini-batch size 1000.

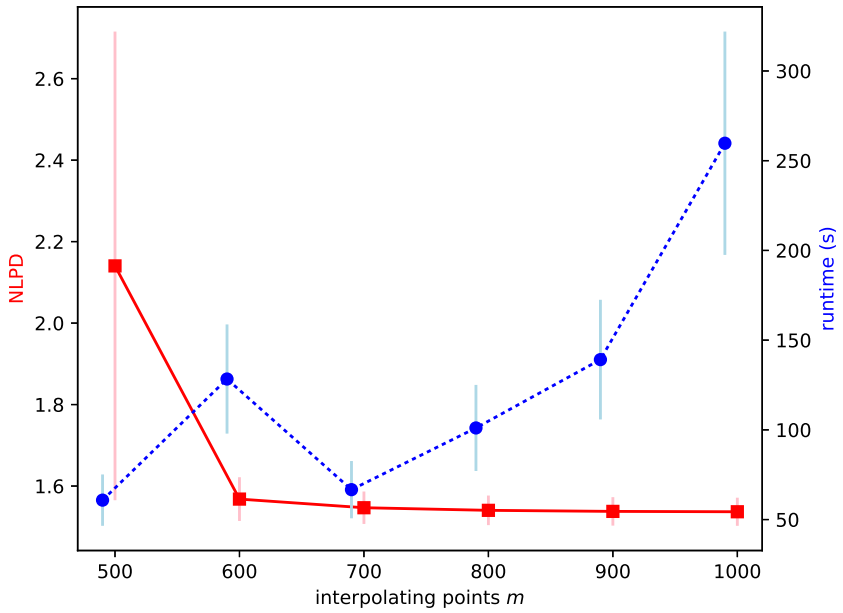


Figure 4.5: Average and standard error of NLPD and runtime of the SLFM model on LLGP across over varying interpolating points. Every setting was run 10 times.

Chapter 5

Conclusion

5.1 Discussion

LLGP recovers speedups from Structured Kernel Interpolation for the problem of multi-output GP regression by recognizing structure unique to linearly coregionalized kernels, and otherwise not necessarily recoverable in general multi-output kernels. This structure further enables a parsimonious representation that allows even large GPs to be learned without explicit construction of the covariance matrix.

LLGP provides a means to approximate the log-likelihood function gradients through interpolation. We have shown on several datasets that this can be done in a way that is faster and leads to more accurate results than variational approximations.

5.2 Future Work

We considered several stochastic approximations for finding $\log \left| K_{|\boldsymbol{\theta}} \right|$ [9, 16], but found these too slow and inaccurate for use in optimization; however, further investigation may prove fruitful, and would enable a more precise stopping condition.

Other future work would extend the inputs to accept multiple dimensions. This can be done without losing internal structure in the kernel [30]: Toeplitz covariance matrices become block-Toeplitz with Toeplitz-blocks (BTTB). The cubic interpolation requires and exponential number of terms, so projection into lower dimensions learned in a supervised manner would be essential. Another useful line for investigation would be more informed stopping heuristics. Finally, an extension to non-Gaussian noise is also feasible in a matrix-free manner by following prior work [8].

Bibliography

[1] Mauricio Alvarez, David Luengo, Michalis Titsias, and Neil D Lawrence. Efficient multioutput Gaussian processes through variational inducing kernels. In *AISTATS*, volume 9, pages 25–32, 2010.

[2] Mauricio Alvarez, Lorenzo Rosasco, Neil Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012.

[3] Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W Hogg, and Michael O'Neil. Fast direct methods for gaussian processes. *arXiv preprint arXiv:1403.6015*, 2014.

[4] J. Bayer, C. Osendorfer, S. Diot-Girard, T. RÃijckstiess, and Sebastian Urban. climin - a pythonic framework for gradient-based function optimization. `http://github.com/BRML/climin`, 2016.

[5] Tony Chan and Julia Olkin. Circulant preconditioners for toeplitz-block matrices. *Numerical Algorithms*, 6(1):89–101, 1994.

[6] Li-Fang Cheng, Gregory Darnell, Corey Chivers, Michael Draugelis, Kai Li, and Barbara Engelhardt. Sparse multi-output gaussian processes for medical time series prediction. *arXiv preprint arXiv:1703.09112*, 2017.

[7] John Cunningham, Krishna Shenoy, and Maneesh Sahani. Fast Gaussian process methods for point process intensity estimation. In *25th international conference on Machine learning*, pages 192–199. ACM, 2008.

[8] Kurt Cutajar, Michael Osborne, John Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *ICML*, pages 2529–2538, 2016.

[9] Sebastian Dorn and Torsten Enßlin. Stochastic determination of matrix determinants. *Physical Review E*, 92(1):013302, 2015.

[10] David Fong and Michael Saunders. CG versus MINRES: an empirical comparison. *SQU Journal for Science*, 17(1):44–62, 2012.

[11] Marc Genton and William Kleiber. Cross-covariance functions for multivariate geostatistics. *Statistical Science*, 30(2):147–163, 2015.

[12] Mark Gibbs and David MacKay. Efficient implementation of Gaussian processes, 1996.

[13] Elad Gilboa, Yunus Saatçi, and John Cunningham. Scaling multidimensional inference for structured Gaussian processes. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):424–436, 2015.

[14] GPy. GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

[15] Alexander Gray and Andrew Moore. Nonparametric density estimation: Toward computational tractability. In *2003 SIAM International Conference on Data Mining*, pages 203–211. SIAM, 2003.

[16] Insu Han, Dmitry Malioutov, and Jinwoo Shin. Large-scale log-determinant computation through stochastic Chebyshev expansions. In *ICML*, pages 908–917, 2015.

[17] James Hensman, Nicolò Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 282–290. AUAI Press, 2013.

[18] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017-02-06].

[19] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[20] Iain Murray. Gaussian processes and fast matrix-vector multiplies. In *Numerical Mathematics in Machine Learning Workshop-International Conference on Machine Learning ICML 2009*, 2009.

[21] Trung Nguyen, Edwin Bonilla, et al. Collaborative multi-output Gaussian processes. In *UAI*, pages 643–652, 2014.

[22] Michael Osborne, Stephen Roberts, Alex Rogers, Sarvapali Ramchurn, and Nicholas Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output Gaussian processes. In *7th international conference on Information processing in sensor networks*, pages 109–120. IEEE Computer Society, 2008.

[23] Joaquin Quiñonero-Candela and Carl Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[24] Joaquin Quiñonero-Candela, Carl Rasmussen, AnÃbal Figueiras-Vidal, et al. Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881, 2010.

[25] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[26] Vikas Raykar and Ramani Duraiswami. Fast large scale Gaussian process regression using approximate matrix-vector products. In *Learning workshop*, 2007.

[27] Matthias Seeger, Yee-Whye Teh, and Michael Jordan. Semiparametric latent factor models. In *Eighth Conference on Artificial Intelligence and Statistics*, 2005.

[28] Yirong Shen, Andrew Ng, and Matthias Seeger. Fast gaussian process regression using kd-trees. *Advances in neural information processing systems*, 18:1225, 2006.

[29] Christopher Williams and Carl Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, pages 514–520, 1996.

[30] Andrew Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on massively scalable Gaussian processes. *arXiv preprint arXiv:1511.01870*, 2015.

[31] Andrew Wilson, Elad Gilboa, John Cunningham, and Arye Nehorai. Fast kernel learning for multidimensional pattern extrapolation. In *Advances in Neural Information Processing Systems*, pages 3626–3634, 2014.

[32] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (kiss-gp). In *32nd International Conference on Machine Learning*, pages 1775–1784, 2015.

[33] Matthew Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[34] Max Zwiessele. paramz. `https://github.com/sods/paramz`, 2017.