# SCALABLE KERNEL METHODS AND THEIR USE IN BLACK-BOX OPTIMIZATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

David Mikael Eriksson

December 2018

SCALABLE KERNEL METHODS AND THEIR USE IN BLACK-BOX OPTIMIZATION

David Mikael Eriksson, Ph.D.

Cornell University 2018

This dissertation uses structured linear algebra to scale kernel regression methods based on Gaussian processes (GPs) and radial basis function (RBF) interpolation to large, high-dimensional datasets. While kernel methods provide a general, principled framework for approximating functions from scattered data, they are often seen as impractical for large data sets as the standard approach to model fitting scales cubically with the number of data points. We introduce RBFs in §1.3 and GPs in §1.4.

Chapter 2 develops novel $O(n)$ approaches for GP regression with $n$ points using fast approximate matrix vector multiplications (MVMs). Kernel learning with GPs require solving linear systems and computing the log determinant of an $n \times n$ kernel matrix. We use iterative methods relying on the fast MVMs to solve the linear systems and leverage stochastic approximations based on Chebyshev and Lanczos to approximate the log determinant. We find that Lanczos is generally highly efficient and accurate and superior to Chebyshev for kernel learning. We consider a large variety of experiments to demonstrate the generality of this approach.

Chapter 3 extends the ideas from Chapter 2 to fitting a GP to both function values and derivatives. This requires linear solves and log determinants with an $n(d+1) \times n(d+1)$ kernel matrix in $d$ dimensions, leading to $O(n^3d^3)$ computations for standard methods. We extend the previous methods and introduce a

pivoted Cholesky preconditioner that cuts the iterations to convergence by several orders of magnitude. Our approaches, together with dimensionality reduction, lets us scale Bayesian optimization with derivatives to high-dimensional problems and large evaluation budgets.

We introduce surrogate optimization in §1.5. Surrogate optimization is a key application of GPs and RBFs, where they are used to model a computationally-expensive black-box function based on previous evaluations. Chapter 4 introduces a global optimization algorithm for computationally expensive black-box function based on RBFs. Given an upper bound on the semi-norm of the objective function in a reproducing kernel Hilbert space associated with the RBF, we prove that our algorithm is globally convergent even though it may not sample densely. We discuss expected convergence rates and illustrate the performance of the method via experiments on a set of test problems.

Chapter 5 describes Plumbing for Optimization with Asynchronous Parallelism (`POAP`) and the Python Surrogate Optimization Toolbox (`pySOT`). `POAP` is an event-driven framework for building and combining asynchronous optimization strategies, designed for global optimization of computationally expensive black-box functions where concurrent function evaluations are appealing. `pySOT` is a collection of synchronous and asynchronous surrogate optimization strategies, implemented in the `POAP` framework. The `pySOT` framework includes a variety of surrogate models, experimental designs, optimization strategies, test problems, and serves as a useful platform to compare methods. We use `pySOT`, to make an extensive comparison between synchronous and asynchronous parallel surrogate optimization methods, and find that asynchrony is never worse than synchrony on several challenging multimodal test problems.

## BIOGRAPHICAL SKETCH

David Eriksson was born in Hammarö, Sweden on September 24, 1989 to Mikael Eriksson and Malin Eriksson. David has always been interested in mathematics and programming and he competed in several mathematics competitions during high-school where he was the first reserve for the Swedish IMO team. He attended Chalmers University of Technology where he obtained a Bachelor of Science degree in Mathematics and was a member of the team that finished second in the Nordic Mathematics Contest. David wanted to gain more research experience and spent a year at the NASA Goddard Space Flight Center where he worked on many interesting modeling problems and published several articles. David returned to Chalmers University of Technology to pursue a Master of Science in Engineering Mathematics and Computational Science. He worked part-time at the Fraunhofer-Chalmers Centre where he wrote his thesis on path-planning in massive point clouds. David started his Ph.D. at Cornell in August 2014 and immediately started working with his advisor David Bindel and his committee member Christine Shoemaker. In January 2019, he will start working as a Research Scientist at Uber AI Labs.

This thesis is dedicated to my family.

## ACKNOWLEDGEMENTS

I owe my deepest gratitude to my advisor David Bindel, who gave me the freedom and flexibility to explore many exciting topics, while also helping me mature as a mathematician and as a programmer. Thank you to Christine Shoemaker for introducing me to surrogate optimization, and to my committee members Charles Van Loan and Alex Townsend for your continuous support. I am grateful to my collaborators Kun Dong, Eric Lee, Hannes Nickisch, and Andrew Wilson, who were all involved in the work on scalable Gaussian processes, which led to two NIPS publications.

I want to thank the entire Center for Applied Mathematics (CAM) and everyone who made CAM a great Ph.D. program. Most importantly, I want to thank my friends Marc Gilles, Kun Dong, Eric Lee, Bram Wallace, Evan Randles, Kyle Perline, Andrew Loeb, Kevin O'Keeffe, my house mates, and many more for making my time at Cornell University as enjoyable as possible. I am also greatly indebted to my Mom, Dad, Brother, and Grandparents, for their unlimited love and support.

I would also like to thank everyone I worked with during my internships at MathWorks in the summer of 2017 and Google in the summer of 2018. These internships were very influential to the work in this thesis and for convincing me to pursue an industry career.

This work was supported by the Center for Applied Mathematics, Thanks to Scandinavia, The Sweden-America Foundation, and teaching assistantships in the Computer Science department at Cornell University.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# LIST OF SYMBOLS

CHAPTER 1

# INTRODUCTION

## 1.1 Summary of introduction

We start by motivating a general framework for scattered data interpolation with kernel methods in §1.2. We describe radial basis function (RBF) interpolation in §1.3 as a special case of interpolation with kernel methods. The error analysis for RBFs takes place in the native space, which is motivated in §1.3.1. In §1.3.2, we summarize some of the error analysis for RBFs; we show in Chapter 4 that this leads to a very natural optimization algorithm. We end the RBF section by briefly summarizing some stability bounds and how to efficiently add a few new points to the RBF interpolant in §1.3.3 and §1.3.4 respectively.

We proceed to introduce Gaussian process (GP) regression in §1.4 and explain its numerous connections to RBFs. We explain how derivative information can be incorporated in the GP model, and similarly in an RBF model. Scalability issues for GPs are addressed in Chapter 2 and Chapter 3. In §1.5, we introduce surrogate optimization and give an overview of surrogate optimization techniques in §1.5.1. We explore surrogate optimization with asynchrony in Chapter 5.

## 1.2 Kernel methods

Before introducing GPs and RBFs, we introduce a more general framework for scattered data interpolation. Given a set of pairwise distinct data points $X =$

$\{x_i\}_{i=1}^n \subset \Omega$ and corresponding function values $f(x_1), \ldots, f(x_n)$ we look for a continuous function $s_{f,X}(x)$ such that $s_{f,X}(x_i) = f(x_i)$, $i = 1, \ldots, n$. A common choice is to look for a function $s_{f,X}(x)$ that is a linear combination of continuous basis functions:

$$s_{f,X}(x) = \sum_{i=1}^{n} \lambda_i b_i(x)$$

and solve the linear system $A_X \lambda = f_X$, where $(A_X)_{ij} = b_j(x_i)$ and $(f_X)_i = f(x_i)$, to determine the coefficients $\lambda_1, \ldots, \lambda_n$. This problem is well-posed if there is a unique solution to this linear system, i.e., $A_X$ is non-singular. For $d = 1$, polynomial interpolation with the monomial basis functions $b_i(x) = x^{i-1}$ yields a well-posed interpolation problem for any pairwise distinct data points. If $\Omega \subset \mathbb{R}^d$, $d \geq 2$ contains an interior point, the famous Mairhuber-Curtis theorem states that in order for $\det A_X \neq 0$ for all pairwise distinct data points in $\Omega$, the basis functions must depend on the data points $X$ [46, 11]. We will therefore restrict our attention to basis functions that depend on $X$.

Characterizing all data dependent basis functions that lead to a well-posed interpolation problem is challenging, so a simple restriction is to require $A_X$ to be symmetric and positive definite for any pairwise distinct data points [17]. This is achieved by considering symmetric positive definite (SPD) kernels:

**Definition 1.** *A (continuous) symmetric function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a positive semi-definite kernel if*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0 \tag{1.1}$$

*for any pairwise disjoint $x_1, \ldots, x_n \in \mathbb{R}^d$ and $c_1, \ldots, c_n \in \mathbb{R}$. The kernel is called symmetric positive definite (SPD) if equality in (1.1) implies $c_1 = \cdots = c_n = 0$.*

For consistency with the naming convention in the numerical linear algebra

literature, we deviate from the naming convention in the literature on kernel methods in which positive definite and strictly positive definite are used in place of positive semi-definite and positive definite.

RBF interpolation restricts the kernel to be radial, that is, $k(x, y)$ depends only on $r := \|x - y\|$ and we often use $\varphi(r)$ to denote a radial kernel. It is common to consider conditionally positive definite (CPD) kernels in RBF interpolation, defined as follows:

**Definition 2.** *A (continuous) function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called conditionally semi-positive definite kernel of order $\nu$ if*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0 \tag{1.2}$$

*for any pairwise disjoint $x_1, \ldots, x_n \in \mathbb{R}^d$ and $c_1, \ldots c_n \in \mathbb{R}$ that satisfy*

$$\sum_{i=1}^{n} c_i q(x_i) = 0 \tag{1.3}$$

*for any real-valued polynomial $q$ of degree at most $\nu - 1$. The kernel is called conditionally positive definite (CPD) of order $\nu$ if equality in (1.2) implies $c_1 = \cdots = c_n = 0$.*

To end up with a well-posed interpolation problem, we need to modify the form of the interpolant $s_{f,X}$ by adding a low-degree polynomial of degree $\nu - 1$, as well as imposing a unisolvency condition of the points $X$. We will describe this in the next section where we introduce RBFs. Note that an SPD kernel is automatically CPD of order 0.

While RBFs are deterministic models, GPs are infinite dimensional stochastic processes that are described by their mean and covariance functions. The

covariance function is modeled using a positive semi-definite kernel which describes interactions between points. We will give several examples of kernels in §1.4 when we introduce GPs.

## 1.3 Radial basis functions

This section summarizes radial basis function interpolation and the associated convergence theory in the setting of a native space. Radial basis function (RBF) interpolation is one of the most popular approaches to approximating scattered data in an arbitrary number of dimensions [5, 17, 63, 72]. The set of interpolation points $X$ are assumed to be pairwise distinct and $\varphi$ is a radial kernel of order $\nu$. The RBF model takes the form

$$s_{f,X}(x) = \sum_{i=1}^{n} \lambda_i \varphi(\|x - x_i\|) + p(x) \tag{1.4}$$

where $p \in \Pi^d_{\nu-1}$, the space of polynomials with $d$ variables of degree no more than $\nu - 1$. There are many possible choices for $\varphi$ and we summarize the most popular choices and some of their properties in Table 1.1 on page 10.

Some kernels in Table 1.1 have a shape parameter $\epsilon$ and a good choice is critical to achieve a good function approximation [18, 62]. The coefficients $\lambda_i$ are determined by imposing the interpolation conditions $s_{f,X}(x_i) = f(x_i)$ for $i = 1, \ldots, n$ and the discrete orthogonality condition

$$\sum_{i=1}^{n} \lambda_i q(x_i) = 0, \qquad \forall q \in \Pi^d_{\nu-1}. \tag{1.5}$$

Note that this condition is in agreement with Definition 2. If we let $\{\pi_i\}_{i=1}^m$ be a basis for the $m = \binom{\nu-1+d}{\nu-1}$-dimensional linear space $\Pi^d_{\nu-1}$ we can write $p(x) =$

$\sum_{i=1}^{m} c_i \pi_i(x)$ and the interpolation conditions lead to the following linear system of equations:

$$\begin{bmatrix} \Phi_{XX} & P_X \\ P_X^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ c \end{bmatrix} = \begin{bmatrix} f_X \\ 0 \end{bmatrix}, \tag{1.6}$$

where $(\Phi_{XX})_{ij} = \phi(\|x_i - x_j\|)$, $(P_X)_{ij} = \pi_j(x_i)$, and $f_X = [f(x_1), \ldots, f(x_n)]^T$. The solution to the linear system of equations is unique as long as rank$(P_X) = m$, which is often referred to as a unisolvency condition on the points $X$; the coefficients of $c$ are uniquely determined from the values at $X$. We next show that the system in (1.6) is non-singular by showing that the only solution to the homogeneous linear system ($f_X = 0$) is $\lambda = c = 0$. Multiplying the first set of equations by $\lambda^T$ yields $\lambda^T \Phi_{XX} \lambda + \lambda^T P_X c = 0$, but since $\lambda^T P_X c = c^T (P_X^T \lambda) = 0$ we have $\lambda^T \Phi_{XX} \lambda = 0$. It follows that $\lambda = 0$ since $\varphi$ is CPD of order $\nu$. The first set of equations are now $P_X c = 0$, and we conclude that $c = 0$ since $P_X$ has full rank.

### 1.3.1 The native space

We study the convergence of interpolation with RBFs in an associated *native space*. Before we introduce the native space we need to define the RBF space $\mathcal{A}_\varphi$, which is the space of functions $s_{f,X}$ of the form (1.4) that satisfy (1.5) for $n(s)$ pairwise distinct points $X = \{x_1, \ldots, x_{n(s)}\} \subseteq \Omega$. The space $\mathcal{A}_\varphi$ can be equipped with the semi-inner product

$$\langle s, u \rangle = \sum_{i=1}^{n(s)} \lambda_i u(x_i)$$

for $s, u \in \mathcal{A}_\varphi$ defined through

$$s_{f,X}(x) = \sum_{i=1}^{n(s)} \lambda_i \varphi(\|x - x_i\|) + p(x) \quad \text{and} \quad u_{f,Y}(x) = \sum_{j=1}^{n(u)} \mu_j \varphi(\|x - y_j\|) + q(x)$$

where both $\{x_i\}_{i=1}^{n(s)}$ and $\{y_j\}_{j=1}^{n(u)}$ are pairwise distinct, $p, q \in \Pi_{\nu-1}^d$, and both $\lambda$ and $\mu$ satisfy the discrete orthogonality condition (1.5). It is easy to verify that this is indeed a semi-inner product; see e.g., [30]. We can now define a semi-norm on $\mathcal{A}_\varphi$ via

$$\begin{aligned} |s_{f,X}|^2 &:= \langle s_{f,X}, s_{f,X} \rangle \\ &= \sum_{i=1}^{n} \lambda_i s_{f,X}(x_i) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j \varphi(\|x_i - x_j\|) \\ &= \lambda^T \Phi_{XX} \lambda. \end{aligned}$$

For a given kernel we can find a Hilbert space $\mathcal{N}_\varphi(\Omega) \subseteq C(\Omega)$, which is usually referred to as the native space. We define the native space as the functions $f \in C(\Omega)$ that have uniformly bounded interpolant norms for any unisolvent set of points $X \subset \Omega$ and the native space semi-norm is given by

$$|f|_{\mathcal{N}_\varphi} := \sup_{X \subset \Omega,\, |X| < \infty} |s_{f,X}|.$$

The native space is hard to characterize for infinitely smooth kernels, but in the special case of "polyharmonic" kernels

$$\varphi_{\text{ph}}(r) := \begin{cases} \frac{\Gamma(d/2-\ell)}{2^{2\ell}\pi^{d/2}(\ell-1)!} r^{2\ell-d} & d \text{ odd} \\ \frac{(-1)^{\ell+(d-2)/2}}{2^{2\ell-1}\pi^{d/2}(\ell-1)!(\ell-d/2)!} r^{2\ell-d} \log(r) & d \text{ even} \end{cases}$$

the native space turns out to be instances of Beppo-Levi spaces:

$$\text{BL}_\ell(\mathbb{R}^d) = \{f \in C(\mathbb{R}^d) : D^\alpha f \in L^2(\mathbb{R}^d), \ \ \forall |\alpha| = \ell, \, \alpha \in \mathbb{N}^d\},$$

for $\ell > d/2$. The following proposition is taken from [30] and is a useful characterization of the native space for the most popular piecewise smooth kernels.

**Proposition 1.** *Let $\varphi(r) = r$, $\varphi(r) = r^2 \log(r)$ or $\varphi(r) = r^3$. Further let $\kappa = 1$ in the linear case, $\kappa = 2$ in the thin plate spline case and $\kappa = 3$ in the cubic case, and choose the integer $m$ such that $0 \leq m \leq d$ in the linear case, $1 \leq m \leq d + 1$ in the thin plate spline case, and $1 \leq m \leq d + 2$ in the cubic case. Define $\nu := (d + \kappa)/2$ if $d + \kappa$ is even, and $\nu := (d + \kappa + 1)/2$ otherwise. If $f \in C^\nu(\mathbb{R}^d)$ has bounded support, then $f \in \mathcal{N}_{\varphi,k}(\mathbb{R}^d)$.*

### 1.3.2 Error estimates

In this section, we summarize the standard convergence theory for RBFs. We need to introduce the cardinal basis functions, which are the solutions to the linear system of equations

$$\begin{bmatrix} \Phi_{XX} & P_X \\ P_X^T & 0 \end{bmatrix} \begin{bmatrix} u^*(x) \\ v^*(x) \end{bmatrix} = \begin{bmatrix} \Phi_{Xx} \\ P_x \end{bmatrix},$$

where $x \in \Omega$. An immediate consequence is that we can write the RBF interpolant in the following form:

$$s_{f,X}(x) = \lambda^T \Phi_{Xx} + c^T P_x^T = [\lambda^T \Phi_{XX} + c^T P_X^T] u^*(x) + \lambda^T P_X v^*(x) = f_X^T u^*(x).$$

The next important ingredient is the power function $P_{X,\varphi}(x)$, defined as

$$[P_{X,\varphi}(x)]^2 = \varphi(0) - 2u^*(x)^T \Phi_{Xx} + u^*(x)^T \Phi_{XX} u^*(x). \tag{1.7}$$

Using the definition of the cardinal functions we can rewrite the power function in the following way

$$\begin{aligned}
[P_{X,\varphi}(x)]^2 &= \varphi(0) - 2u^*(x)^T\Phi_{Xx} + u^*(x)[\Phi_{Xx} - P_X v^*(x)] \\
&= \varphi(0) - u^*(x)^T[\Phi_{Xx} + P_X v^*(x)] \\
&= \varphi(0) - u^*(x)^T\Phi_{Xx} - v^*(x)^T P_x \\
&= \varphi(0) - b(x)^T A^{-1} b(x)
\end{aligned}$$

where

$$b(x) = \begin{bmatrix} \Phi_{Xx} \\ P_x \end{bmatrix} \text{ and } A = \begin{bmatrix} \Phi_{XX} & P_X \\ P_X^T & 0 \end{bmatrix}.$$

It is possible to derive the following pointwise inequality based on the power function

$$|f(x) - s_{f,X}(x)| \leq P_{X,\varphi}(x)\sqrt{|f|^2_{\mathcal{N}_\varphi} - |s_{f,X}|^2_{\mathcal{N}_\varphi}} \leq P_{X,\varphi}(x)|f|_{\mathcal{N}_\varphi}, \tag{1.8}$$

which holds for all $x \in \Omega$. This bound is often interpreted as a worst case bound; it tells us the smallest and largest possible function values at $x$.

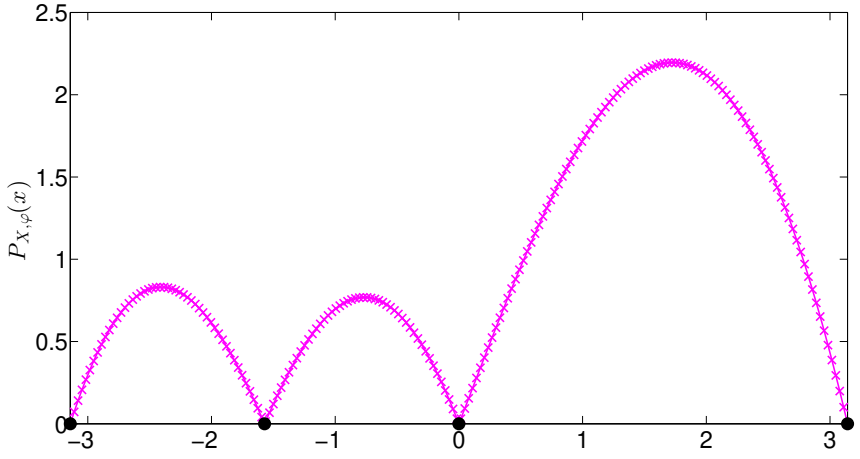



Figure 1.1: An example of the power function for the points $\{-\pi, \pi/2, 0, \pi\}$. The power function is zero at evaluated points and large when the distance to the closest evaluated point is large.

Error estimates for the RBF interpolation problem are based on the fill-distance

$$h_{X,\Omega} := \sup_{x\in\Omega} \min_{x_j\in X} \|x - x_j\|,$$

and convergence is studied as $h_{X,\Omega} \to 0$. The standard approach to proving a pointwise convergence rate is to start from (1.8) and bound the power function via a Taylor expansion. The bounds $P^2_{X,\varphi}(x) \leq F_\varphi(h_{X,\Omega})$, which hold for $h_{X,\Omega}$ small enough, are summarized in Table 1.1 on page 10 for the most common RBF kernels [72]. Note that these results are only applicable in cases where the fill-distance decreases with the number of points, which is not necessarily the case in e.g., global optimization, which will be introduced in §1.5.1.

### 1.3.3 Stability

We saw in the previous section and in in Table 1.1 that we can achieve exponential convergence for the Gaussian kernel as the fill-distance decreases, but there is a conflict between this theoretically achievable accuracy and numerical stability. The goal of this section is to study what happens to the condition number of $\Phi_{XX}$ when the separation-distance

$$q_X = \frac{1}{2} \min_{i\neq j} \|x_i - x_j\|$$

decreases. We will focus on symmetric positive definite kernels and refer to Wendland [72] for a rigorous treatment of conditionally positive definite kernels. In this case, $\Phi_{XX}$ will be symmetric and positive definite and saying something about the condition number of $\Phi_{XX}$ requires an upper bound on the largest eigenvalue and a lower bound on the smallest eigenvalue since

$$\kappa(\Phi_{XX}) := \|\Phi_{XX}\| \cdot \|\Phi^{-1}_{XX}\| = \frac{\lambda_{\max}(\Phi_{XX})}{\lambda_{\min}(\Phi_{XX})}.$$

Finding an upper bound on $\lambda_{\max}(\Phi_{XX})$ ends up being a straightforward application of Gershgorin's theorem which says that

$$|\lambda_{\max}(\Phi_{XX}) - (\Phi_{XX})_{ii}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |(\Phi_{XX})_{ij}|,$$

for some $i \in \{1, \ldots, n\}$. This allows us to conclude that

$$\lambda_{\max}(\Phi_{XX}) \leq n \max_{i,j=1,\ldots,n} |(\Phi_{XX})_{ij}| = n \max_{x_i, x_j \in X} |\varphi(x_i - x_j)| \leq n\varphi(0).$$

After establishing an upper bound on $\lambda_{\max}(\Phi_{XX})$ we turn our attention to finding a lower bound $G_\varphi(q_X) \leq \lambda_{\min}(\Phi_{XX})$. Deriving these lower bounds is rather technical and we refer to Wendland [72] for more details, but lower bounds in terms of $q_X$ have been summarized in Table 1.1.

| Name | $\varphi(r)$ | Order | $F_\varphi(h_{X,\Omega})$ | $G_\varphi(q_X)$ |
|---|---|---|---|---|
| Gaussian | $e^{-\epsilon^2 r^2}$ | $\nu = 0$ | $e^{-c\lvert\log(h)\rvert/h}$ | $c(\sqrt{2}\epsilon)^{-d} e^{-40.71d^2/(q\epsilon^2)^2} q^{-d}$ |
| Inverse multiquadric | $\left(1 + \epsilon^2 r^2\right)^{\beta}, \beta < 0$ | $\nu = 0$ | $e^{-\tilde{c}/h}$ | $cq^{\beta - \frac{d}{2} + \frac{1}{2}} e^{-c/(q\epsilon)}$ |
| Multiquadric | $(-1)^{\lceil\beta\rceil}\left(1 + \epsilon^2 r^2\right)^{\beta}, 0 < \beta \notin \mathbb{N},$ | $\nu = \lceil\beta\rceil$ | $e^{-\tilde{c}/h}$ | $cq^{\beta - \frac{d}{2} + \frac{1}{2}} e^{-c/(q\epsilon)}$ |
| Radial powers | $(-1)^{\lceil\beta/2\rceil} r^{\beta}, 0 < \beta \notin 2\mathbb{N}$ | $\nu = \lceil\beta/2\rceil$ | $h^{\beta}$ | $cq^{\beta}$ |
| Thin-plate spline | $(-1)^{\beta+1} r^{2\beta} \log(r), \beta \in \mathbb{N}$ | $\nu = \beta + 1$ | $h^{2\beta}$ | $cq^{2\beta}$ |

Table 1.1: Some popular choices of kernels and their order and convergence properties. The functions $F_\varphi(h_{X,\Omega})$ are asymptotic upper bounds on the squared power function in terms of the fill-distance, while $G_\varphi(q_X)$ are lower bounds on the smallest eigenvalue of $\Phi_{XX}$ based on the separation distance. The shape parameter $\epsilon$ has to satisfy $\epsilon > 0$.

It has been shown that

$$G_\varphi(q_X) \leq F_\varphi(h_{X,\Omega})$$

and if the data is well-distributed we have $q_X \approx h_{X,\Omega}$ so a small value of $F_\varphi(h_{X,\Omega})$ implies a small value of $G_\varphi(q_X)$. We have arrived at something that is more commonly known as the trade-off principle, which tells us that a fast decrease in

$F_\varphi(h_{X,\Omega})$ implies a fast increase in the condition number of $\Phi_{XX}$ [17]. The theoretical guarantee of exponential convergence for the Gaussian and (inverse) multiquadric kernels are not achievable in practice as the condition number of $\Phi_{XX}$ grows exponentially, leading to numerical instabilities. This is one of the reasons why the radial powers and thin-plate splines are often favored in practice, since they offer a better balance between convergence and stability.

Recent work has shown how to overcome these stability issues for the Gaussian and (inverse) multiquadric kernels in the limit $\epsilon \to 0$, relying on a complex Contour-Padé integration algorithm [14, 20, 21, 53]. However, this approach is limited to very small datasets, and thus leads to another trade-off principle stating that high accuracy is only achievable for very small datasets. Another way to deal with the numerical instability caused by a small value of $q_X$ is to add regularization to the kernel, $\tilde{\varphi}(\|x_i - x_j\|) = \varphi(\|x_i - x_j\|) + \eta\,\delta_{ij}$, for some regularization parameter $\eta \geq 0$, where $\eta$ is often chosen using cross-validation.

### 1.3.4 Incremental updates

A direct solver of the RBF system (1.6) requires computing a dense LU (or Cholesky for SPD kernels) factorization at a cost of $O((m+n)^3) = O(n^3)$ flops. There are several contexts, including global optimization in §1.5, where we have computed an RBF interpolant and want to add $q \ll n$ new points to the model. This section describes how to update the RBF interpolant in quadratic time. Given an initial set of $n$ points with $\text{rank}(P_X) = m$ we compute an initial LU

factorization with pivoting

$$\tilde{A} := \begin{bmatrix} 0 & P_X^T \\ P_X & \Phi_{XX} \end{bmatrix} = P^T L_{11} U_{11}.$$

Note that the blocks have been rearranged compared to (1.6) to make it more natural to add new points. After adding the $q$ new points $\hat{X} = \{\hat{x}_i\}_{i=1}^q$ we want to find an LU factorization of the extended system

$$\hat{A} = \left[\begin{array}{cc|c} 0 & P_X^T & P_{\hat{X}}^T \\ P_X & \Phi_{XX} & \Phi_{X\hat{X}} \\ \hline P_{\hat{X}} & \Phi_{\hat{X}X} & \Phi_{\hat{X}\hat{X}} \end{array}\right] := \begin{bmatrix} \tilde{A} & B \\ B^T & C \end{bmatrix}.$$

Assuming the kernel $\varphi$ is of order $\nu$ and $p(x)$ is of order at least $\nu - 1$, a unisolvent set of initial points $X$ implies that the trailing Schur complement is positive semi-definite, so we can look for a factorization of the form

$$\begin{bmatrix} \hat{A} & B \\ B^T & C \end{bmatrix} = \begin{bmatrix} P^T & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & L_{22}^T \end{bmatrix} = \begin{bmatrix} P^T L_{11} U_{11} & P^T L_{11} U_{12} \\ L_{21} U_{11} & L_{21} U_{12} + L_{22} L_{22}^T \end{bmatrix}.$$

We need to solve the two triangular systems $B = P^T L_{11} U_{12}$ and $B^T = L_{21} U_{11}$ followed by computing a Cholesky factorization of $C - L_{21} U_{12}$. This implies that we can update the factorization in $O(qn^2 + q^3)$ flops, which is better than computing a new LU factorization in $O(n^3)$ flops. This incremental update idea is very useful in the context of surrogate optimization, which we introduce in §1.5.1 and consider in more detail in Chapter 5.

## 1.4 Gaussian processes

A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution [56]. They share many similarities with RBFs as both are based on kernel interpolation. More generally, a GP can be used to define a distribution over functions $f(x) \sim \mathcal{GP}(\mu(x), k(x, x'))$, where each function value is a random variable indexed by $x \in \mathbb{R}^d$, and $\mu : \mathbb{R}^d \to \mathbb{R}$ and $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are the mean and covariance functions of the process. The mean function is often a low-degree polynomial while the covariance function fills the same role as the kernel in RBF interpolation, with the additional restriction that it must be positive semi-definite. However, previous work by Wahba [69] connects regression with smoothing splines to the use of improper priors. We will show later in this section that we can also modify the spline kernels used in RBF interpolation to make them positive definite on any compact domain.

The covariance function is often chosen to be the squared exponential (or Gaussian) kernel

$$k_{\text{SE}}(x, y) = s^2 \exp\left(-\frac{\|x - y\|^2}{2\ell^2}\right)$$

or Matérn kernel

$$k_{\text{Matern}}(x, y) = s^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{\|x - y\|}{\ell}\right)^{\nu} K_\nu \left(\sqrt{2\nu}\frac{\|x - y\|}{\ell}\right)$$

where $\Gamma$ is the gamma function, $K_\nu$ is the modified Bessel function of the second kind, $s^2$ is the signal variance, $\ell > 0$ is the lengthscale, and $\nu > 0$. The Matérn kernel is $\lceil \nu \rceil - 1$ times differentiable and $1/2$, $3/2$, and $5/2$ are common choices for $\nu$ to model heavy-tailed correlations. The length scale $\ell$ is used instead of the $\epsilon$ parameter in RBF interpolation, with the interpretation that a larger value of the lengthscale indicates that more points are correlated. The parameter $s$ is added

to model the signal variance of the function. The spectral behavior of both these kernels has been well-studied for years, and we recommend Wathen and Zhu [71] for recent results. Particularly relevant is a theorem due to Weyl, which says that if a symmetric kernel has $\nu$ continuous derivatives, then the eigenvalues decay like $|\lambda_n| = O(n^{-\nu-1/2})$. Hence, the eigenvalues decay much more slowly for the Matérn kernel, which has two derivatives at zero for $\nu = 5/2$, one derivative at zero for $\nu = 3/2$, and no derivatives at zero for $\nu = 1/2$.

The thin-plate spline and radial powers cannot be used in GP regression as they are not positive definite, but we can modify them to make them valid GP kernels on any compact domain $\Omega$ [74]. This leads to the family of spline kernels

$$\varphi_{\text{spline}}(r) = k_{\text{spline}}(x, y) = \begin{cases} s^2(\|x-y\|^3 + a\|x-y\|^2 + b) & d \text{ odd} \\ s^2(\|x-y\|^2 \log\|x-y\| + a\|x-y\|^2 + b) & d \text{ even} \end{cases}$$

where $a, b$ are chosen to make the spline kernels SPD on the given domain. Letting $R = \max_{x,y\in\Omega} \|x-y\|$, we can impose the boundary conditions $\varphi(R) = 0$ and $\frac{\partial\varphi(r)}{\partial r}\Big|_{r=R} = 0$. This leads to the kernels

$$k_{\text{spline}}(x, y) = \begin{cases} s^2(\|x-y\|^3 - \frac{3}{2}R\|x-y\|^2 + \frac{1}{2}R^3) & d \text{ odd} \\ s^2(\|x-y\|^2 \log\|x-y\| - \left(\frac{1}{2} + \log R\right)\|x-y\|^2 + \frac{1}{2}R^2) & d \text{ even} \end{cases}$$

which are SPD on any compact domain $\Omega$ and can therefore be used for GP regression. We use the spline kernel for implicit surface reconstruction in Chapter 3.

For an arbitrary kernel, we denote any kernel hyperparameters by the vector $\theta$. To be concise we will generally not explicitly denote the dependence of $k$ and associated matrices on $\theta$. For any locations $X$, $f_X \sim \mathcal{N}(\mu_X, K_{XX})$, where $\mu_X$ represent the vectors of function values $\mu$ evaluated at each of the $x_i \in X$, and

$K_{XX}$ is the covariance matrix. Suppose we have a vector of corresponding function values $y_X = [y_1, \ldots, y_n]^T \in \mathbb{R}^n$, where each entry is potentially contaminated by independent Gaussian noise with variance $\sigma^2$. The noise variance is often unknown, so we learn it from the data. In the RBF setting we may use cross-validation to learn $\sigma$, but under a Gaussian process prior we learn $\sigma$ and $\theta$ by optimizing the log marginal likelihood

$$\mathcal{L}(y_X \mid \theta) = -\frac{1}{2}\left[(y_X - \mu_X)^T \tilde{K}_{XX}^{-1}(y_X - \mu_X) + \log|\tilde{K}_{XX}| + n\log 2\pi\right] \tag{1.9}$$

where $\tilde{K}_{XX} = K_{XX} + \sigma^2 I$. Optimization of (1.9) is expensive, since the cheapest way of evaluating $\log|\tilde{K}_{XX}|$ and its derivatives without taking advantage of the structure of $\tilde{K}_{XX}$ involves computing the $O(n^3)$ Cholesky factorization of $\tilde{K}_{XX}$. $O(n^3)$ computations are too expensive for inference and learning beyond even just a few thousand points and we address these scalability issues in Chapter 2.

The posterior mean and posterior variance at a point $x$ are given by

$$\mathbb{E}[f(x)] = K_{Xx}^T(K_{XX} + \sigma^2 I)^{-1}(y_X - \mu_X),$$
$$\mathbb{V}[f(x)] = K_{xx} - K_{Xx}^T(K_{XX} + \sigma^2 I)^{-1}K_{Xx}.$$

The careful reader may recognize the similarities between not only the posterior mean and the RBF prediction, but the predictive variance and the power function. The predictive variance for GPs gives us an idea of the average case error, while the RBF error bound in (1.8) is often interpreted as a worst case bound.

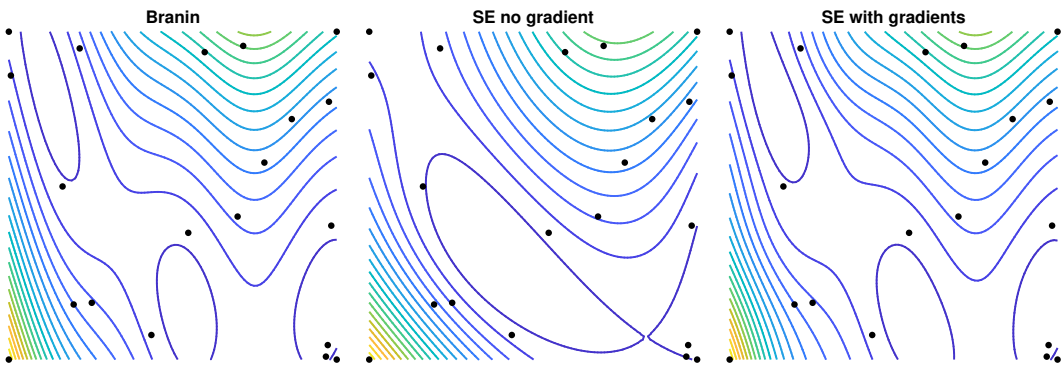



Figure 1.2: An example where gradient information pays off; the true function is on the left. Compare the regular GP without derivatives (middle) to the GP with derivatives (right). Unlike the former, the latter is able to accurately capture critical points of the function.

### 1.4.1 Derivative information

The GP model can easily be extended to incorporate derivative information, which is especially valuable in higher dimensions, but comes at a cost: the kernel matrix $K_{XX}^{\nabla}$ is of size $n(d+1)$-by-$n(d+1)$. This makes scalability an even larger issue as training and prediction become $O(n^3d^3)$ and $O(nd)$ respectively [57]. We address these scalability issues in Chapter 3. Figure 1.2 illustrates the value of derivative information; fitting with derivatives is evidently much more accurate than fitting function values alone.

We define a multi-output GP that allows us to both predict derivatives and make inference based on derivative information. The multi-output GP model takes the form

$$\mu^{\nabla}(x) = \begin{bmatrix} \mu(x) \\ \\ \partial_x \mu(x) \end{bmatrix}, \qquad k^{\nabla}(x, x') = \begin{bmatrix} k(x, x') & (\partial_{x'} k(x, x'))^T \\ \\ \partial_x k(x, x') & \partial^2 k(x, x') \end{bmatrix},$$

where $\partial_x k(x, x')$ and $\partial^2 k(x, x')$ represent the column vector of (scaled) partial

derivatives in $x$ and the matrix of (scaled) second partials in $x$ and $x'$, respectively. As in the scalar GP case, we model measurements of the function as contaminated by independent Gaussian noise, but we can use different noise variances for the function values and each partial derivative.

## 1.5 Surrogate optimization

Surrogate optimization techniques try to solve the global optimization problem

$$\text{Find } x^* \in \Omega \text{ such that } f(x^*) \leq f(x), \quad \forall x \in \Omega \tag{1.10}$$

where $f : \Omega \to \mathbb{R}$ is continuous and $\Omega \subset \mathbb{R}^d$ is compact. We are primarily interested in problems where evaluating $f$ requires running a time consuming simulation model or that evaluating $f$ has a high monetary cost. In addition, we assume that $f$ lacks special structure like linearity and convexity and assume that there can be many local minima. We typically only observe the value of $f(x)$, but the case where we also observe derivatives will be considered in Chapter 3. Several heuristic algorithms such as genetic algorithms, particle swarms algorithms, differential evolution, and simulated annealing are popular choices for global optimization Horst and Pardalos [38], but these algorithms generally require many evaluations to find a good solution to (1.10).

A class of methods that has been shown to perform well with a small number of evaluations is surrogate optimization methods; these methods use a surrogate model that approximates the objective function to choose where to evaluate next. The most popular approach is Bayesian optimization (BO) where we place a GP prior on $f$ [23]. It is also possible to approximate $f$ using an RBF model [29, 58, 60, 73]. Other possible surrogate models are polynomial regres-

sion models and multivariate adaptive regression splines Friedman [24], but this thesis will focus on RBFs and GPs.

### 1.5.1 Surrogate optimization overview

Most surrogate optimization methods follow the same main steps. The first step consists of generating an experimental design with $n_0$ points that are evaluated and used to fit an initial surrogate model. We proceed to the adaptive phase where we solve an auxiliary problem at each step to find new point(s) to evaluate. This auxiliary problem quantifies the expected value from choosing $x$ as the next evaluation. We evaluate the new point(s), update the surrogate model, and repeat this procedure until a stopping criterion has been met. This is illustrated in Algorithm 1.

---

Algorithm 1: Synchronous surrogate optimization algorithm

---

1: Model $f$ using an RBF or place a GP prior on $f$
2: Generate an experimental design
3: Evaluate the points in the experimental design
4: **while** Stopping criterion not met **do**
5: Update the RBF model or the posterior distribution on $f$
6: Optimize auxiliary function $\alpha(x)$ for new point(s) to evaluate
7: Evaluate the new point(s)

---

The simplest experimental design is choosing the $2^d$ corners of the hypercube $\Omega$, often referred to as the 2-factorial design, but this is infeasible when $d$ is large and the evaluations are expensive. Two common alternatives are the Latin

hypercube design (LHD) and the symmetric Latin hypercube design (SLHD), which allows an arbitrary number of design points.

### 1.5.2 Surrogate optimization methods

Evaluating $f$ is expensive, so we optimize an auxiliary function $\alpha(x)$ involving the surrogate model and previously evaluated points to find the next point(s) to evaluate. This auxiliary problem must balance exploration and exploitation, where exploration emphasizes evaluating points far from previous evaluations to improve the surrogate model and escape local minima, while exploitation aims to improve promising solutions to make sure we make progress. We start by introducing two popular auxiliary functions in BO, where they are often referred to as acquisition functions. We refer to Frazier [23] for a description of several additional acquisition functions.

#### Expected improvement

Expected improvement (EI) is likely the most widely used auxiliary function in BO, where the main idea is choosing the point that gives us the largest expected improvement. EI is one-step-optimal, it assumes we have evaluated $f$ (without noise) at $n$ points and have exactly one evaluation left. If $f_n^* = \min_{i=1,\dots,n} f(x_i)$ and we spend our last evaluation at a point $x_{n+1}$, we will return $\min(f_n^*, f(x_{n+1}))$, in which case the improvement from the last function evaluation is

$$I_n(x) = \max\{0, f_n^* - f_{n+1}(x)\}.$$

The expected value can be evaluated analytically under a Gaussian process posterior

$$\mathrm{EI}_n(x) := \mathbb{E}[I_n(x)] = \begin{cases} (f_n^* - \mathbb{E}[f(x)])\Phi(Z) + \sqrt{\mathbb{V}[f(x)]}\phi(Z) & \text{if } \mathbb{V}[f(x)] > 0 \\ 0 & \text{if } \mathbb{V}[f(x)] = 0 \end{cases}$$

where $Z = (f_n^* - \mathbb{E}[f(x)])/\sqrt{\mathbb{V}[f(x)]}$, $\Phi$ is the standard normal cumulative distribution function, and $\phi$ is the standard normal probability density function [40]. EI is implemented in our surrogate optimization software `pySOT`, which we describe in Chapter 5.

**Lower confidence bound**

The main idea of the lower confidence bound (LCB) is to choose the point that minimizes the lower bound of the confidence interval based on the predicted value and predicted variance. Formally, the LCB acquisition is defined as

$$\mathrm{LCB}(x) = \mathbb{E}[f(x)] - \kappa\sqrt{\mathbb{V}[f(x)]},$$

where $\kappa$ is left to the user. We explore a similar idea for RBFs in Chapter 4.

**Stochastic RBF**

The weighted-distance merit function introduced in [58] was proposed in combination with using an RBF model for $f$. The main idea is to generate a set of candidate points $\Lambda$ and pick the candidate point that provides the best balance between a small predicted value and a large distance to previously evaluated points. Exploration is achieved by giving preference to candidate points far from previous evaluations. More specifically, for each $x \in \Lambda$ we let $\Delta(x)$ be the

distance from $x$ to the point closest to $x$ that is currently being or has been evaluated. By defining $\Delta^{\max} = \max\{\Delta(x) : x \in \Lambda\}$ and $\Delta^{\min} = \min\{\Delta(x) : x \in \Lambda\}$ a good measure of exploration is a small value of $V^D(x) = \frac{\Delta^{\max} - \Delta(x)}{\Delta^{\max} - \Delta^{\min}}$, where $0 \leq V^D(x) \leq 1$ for all $x \in \Lambda$. Exploitation is achieved through the surrogate model $s_{f,X}(x)$, where a small value of the quantity $V^S(x) = \frac{s_{f,X}(x) - s^{\min}}{s^{\max} - s^{\min}}$ provides a measure of exploitation, where $s^{\max} = \max\{s_{f,X}(x) : x \in \Lambda\}$ and $s^{\min} = \min\{s_{f,X}(x) : x \in \Lambda\}$.

The best candidate point is the minimizer of $wV^S(x) + (1 - w)V^D(x)$, for a given $w \in [0, 1]$, which shows that $w$ serves as a balance between exploitation and exploration. A weight close to 0 emphasizes exploration while a weight close to 1 emphasizes exploitation. Algorithm 2 shows how to select the most promising candidate point.

---

Algorithm 2: Candidate point selection

---

Compute $s^{\max} \leftarrow \max_{x \in \Lambda} \; s_{f,X}(x)$ and $s^{\min} \leftarrow \min_{x \in \Lambda} \; s_{f,X}(x)$

**for** each $x \in \Lambda$ **do**

$$V^S(x) \leftarrow \begin{cases} \frac{s_{f,X}(x) - s^{\min}}{s^{\max} - s^{\min}} & \text{if } s^{\max} > s^{\min} \\ 1 & \text{otherwise} \end{cases}$$

**for** each $x \in \Lambda$ **do**

$\Delta(x) \leftarrow \min_{y \in \Omega} d(x, y)$

Compute $\Delta^{\max} \leftarrow \max_{x \in \Lambda} \; \Delta(x)$ and $\Delta^{\min} \leftarrow \min_{x \in \Lambda} \; \Delta(x)$

**for** each $x \in \Lambda$ **do**

$$V^D(x) \leftarrow \begin{cases} \frac{\Delta^{\max} - \Delta(x)}{\Delta^{\max} - \Delta^{\min}} & \text{if } \Delta^{\max} > \Delta^{\min} \\ 1 & \text{otherwise} \end{cases}$$

**return** $\operatorname{argmin}_{x \in \Lambda} \; wV^S(x) + (1 - w)V^D(x)$

---

The SRBF method works well for low-dimensional optimization problems. Given a sampling radius $\gamma$, the candidate points are generated as $\mathcal{N}(0, \gamma^2)$ perturbations along each coordinate direction from the best solution [58]. Large values of the sampling radius will generate candidate points far away from the best solution while smaller values of the sampling radius will generate candidate points that are close to the best solution. The sampling radius is initialized to a large value and is updated depending on progress; we increase the sampling radius after a series of successful improvements and decrease if we fail to make improvement. We refer to Regis and Shoemaker [58] for exact details on how the sampling radius is updated.

#### DYCORS

The DYCORS method was developed for high-dimensional problems and the idea is to start by perturbing all coordinates and perturb only a few towards the end of the optimization run [60]. This is achieved by assigning a probability to perturb each dimension. If $n_0$ points are used in the experimental design and the evaluation budget is given by $n_{\max}$, each coordinate is perturbed with probability $p_n$ for $n_0 \leq n \leq n_{\max}$. One possible probability function is $p_n = \min\left(\frac{20}{d}, 1\right) \times \left[1 - \frac{\log(n-n_0)}{\log(n_{\max}-n_0)}\right]$.

## 1.6 Outline

Chapter 2 shows how to resolve the scalability issues in GP regression using structured linear algebra based on work published in NIPS 2017 [13]. This work is generalized to include derivative information in Chapter 3 where one of our

numerical experiments shows the connection between scalable GPs with derivatives and Bayesian optimization. This work is accepted for publication in NIPS 2018 [16]. We introduce a novel surrogate optimization algorithm based on RBFs in Chapter 4, and prove that it converges to the global minimum without dense sampling given a bound on the native space semi-norm. We also show how to compute an upper bound on the semi-norm given a Lipschitz constant. Chapter 5 describes our open source software `pySOT`, which is designed for asynchronous surrogate optimization, and we make an extensive between asynchronous parallel and batch synchronous parallel. We conclude in Chapter 6.

CHAPTER 2

# SCALABLE GAUSSIAN PROCESSES

This chapter will describe recent work on scaling Gaussian process (GP) regression to large datasets. We build on previous work utilizing fast matrix vector multiplication (MVM) with the kernel matrix and show how to accurately estimate the log determinant and derivatives of the kernel matrix using Chebyshev and Lanczos. We find that Lanczos is generally superior to Chebyshev and demonstrate the methods on a large set of experiments. Most of the content in this chapter is based on [13].

## 2.1 Background

There have been many attempts to overcome the scalability issues with Gaussian processes and many current approaches to scalable Gaussian processes [e.g., 54, 43, 35] focus on inference assuming a fixed kernel, or use approximations that do not allow for very flexible kernel learning [77], due to poor scaling with number of basis functions or inducing points. A popular approach to GP scalability is to replace the exact kernel $k(x, z)$ by an approximate kernel that admits fast computations [54]. Several methods approximate $k(x, z)$ via *inducing points* $U = \{u_j\}_{j=1}^{q} \subset \mathbb{R}^d$. An example is the subset of regressor (SoR) kernel:

$$k^{SoR}(x, z) = K_{xU} K_{UU}^{-1} K_{Uz}$$

which is a low-rank approximation [64]. The SoR matrix $K_{XX}^{\text{SoR}} \in \mathbb{R}^{n \times n}$ has rank at most $q$, allowing us to solve linear systems involving $\tilde{K}_{XX}^{\text{SoR}} = K_{XX}^{\text{SoR}} + \sigma^2 I$ and to compute $\log |\tilde{K}_{XX}^{\text{SoR}}|$ in $O(q^2 n + q^3)$ time. Note that we need $\sigma > 0$ to guarantee that

$\tilde{K}_{XX}^{\text{SoR}}$ is invertible. Another popular kernel approximation is the fully independent training conditional (FITC), which is a diagonal correction of SoR so that the diagonal is the same as for the original kernel [65]. Thus kernel matrices from FITC have low-rank plus diagonal structure. This modification has had exceptional practical significance, leading to improved point predictions and much more realistic predictive uncertainty [54, 55], making FITC arguably the most popular approach for scalable GPs.

### 2.1.1 SKI

Wilson and Nickisch [75] provides a mechanism for fast MVMs through proposing the structured kernel interpolation (SKI) approximation,

$$K_{XX} \approx W K_{UU} W^T \tag{2.1}$$

where $W$ is an $n$-by-$q$ matrix of interpolation weights; the authors of [75] use local cubic interpolation so that $W$ is sparse. The sparsity in $W$ makes it possible to naturally exploit algebraic structure (such as Kronecker or Toeplitz structure) in $K_{UU}$ when the inducing points $U$ are on a grid, for extremely fast matrix vector multiplications with the approximate $K_{XX}$ even if the data inputs $X$ are arbitrarily located. For instance, if $K_{UU}$ is Toeplitz, then each MVM with the approximate $K_{XX}$ costs only $O(n + q \log q)$. By contrast, placing the inducing points $U$ on a grid for classical inducing point methods, such as SoR or FITC, does not result in substantial performance gains, due to the costly cross-covariance matrices $K_{xU}$ and $K_{Uz}$.

Approximating the log determinant is challenging with SKI and the scaled eigenvalue method was introduced in [78] to estimate $\log |K_{XX} + \sigma^2 I|$. The eigen-

values $\{\lambda_i\}_{i=1}^n$ of $K_{XX}$ can be approximated using the $n$ largest eigenvalues of a covariance matrix $\tilde{K}_{YY}$ on a full grid with $q$ points such that $X \subset Y$. Specifically,

$$\log|K_{XX} + \sigma^2 I| = \sum_{i=1}^{n} \log(\lambda_i + \sigma^2) \approx \sum_{i=1}^{n} \log\left(\frac{n}{q}\tilde{\lambda}_i + \sigma^2\right)$$

The induced kernel $K_{UU}$ plays the role of $\tilde{K}_{YY}$ when the scaled eigenvalue method is applied to SKI and the eigenvalues of $K_{UU}$ can be efficiently computed. Assuming that the eigenvalues can be computed efficiently is a strong assumption.

### 2.1.2 Diagonal correction to SKI

The SKI approximation may provide a poor estimate of the diagonal entries of the original kernel matrix for kernels with limited smoothness, such as the Matérn kernel. We thus modify the SKI approximation to add a diagonal matrix $D$,

$$K_{XX} \approx W K_{UU} W^T + D, \tag{2.2}$$

such that the diagonal of the approximated $K_{XX}$ is exact. In other words, $D$ substracts the diagonal of $WK_{UU}W^T$ and adds the true diagonal of $K_{XX}$. This modification is not possible for the scaled eigenvalue method for approximating log determinants in [75], since adding a diagonal matrix makes it impossible to approximate the eigenvalues of $K_{XX}$ from the eigenvalues of $K_{UU}$. However, (2.2) still admits fast MVMs. Computing $D$ with SKI costs only $O(n)$ flops since $W$ is sparse for local cubic interpolation. We can therefore compute $(W^T e_i)^T K_{UU}(W^T e_i)$ in $O(1)$ flops.

## 2.2 Approximating the log determinant

Our goal is to estimate, for a symmetric positive definite matrix $\tilde{K}$,

$$\log|\tilde{K}| = \operatorname{tr}(\log(\tilde{K})) \quad \text{and} \quad \frac{\partial}{\partial\theta_i}\left[\log|\tilde{K}|\right] = \operatorname{tr}\left(\tilde{K}^{-1}\left(\frac{\partial\tilde{K}}{\partial\theta_i}\right)\right),$$

where log is the matrix logarithm [36]. We compute the traces involved in both the log determinant and its derivative via *stochastic trace estimators* [39], which approximate the trace of a matrix using only matrix vector products.

The key idea is that for a given matrix $A$ and a random probe vector $z$ with independent entries with mean zero and variance one, then $\operatorname{tr}(A) = \mathbb{E}[z^T A z]$; a common choice is to let the entries of the probe vectors be Rademacher random variables. In practice, we estimate the trace by the sample mean over $n_z$ independent probe vectors. Often surprisingly few probe vectors suffice.

To estimate $\operatorname{tr}(\log(\tilde{K}))$ using Monte Carlo we need to multiply $\log(\tilde{K})$ by the probe vectors $z_i$. We consider two ways to estimate $\log(\tilde{K})z$: by a polynomial approximation of log or by using the connection between the Gaussian quadrature rule and the Lanczos method [32, 68]. In both cases, we show how to re-use the same probe vectors for an inexpensive coupled estimator of the derivatives. In addition, we may use standard radial basis function interpolation of the log determinant evaluated at a few systematically chosen points in the hyperparameter space as an inexpensive surrogate for the log determinant.

### 2.2.1 Chebyshev techniques

Chebyshev polynomials are defined by the recursion

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{j+1}(x) = 2xT_j(x) - T_{j-1}(x) \text{ for } j \geq 1,$$

where $x \in [-1, 1]$. For $f : [-1, 1] \to \mathbb{R}$ the Chebyshev interpolant of degree $t$ is

$$f(x) \approx p_t(x) := \sum_{j=0}^{t} c_j T_j(x), \quad \text{where } c_j = \frac{2 - \delta_{j0}}{t+1} \sum_{k=0}^{t} f(x_k) T_j(x_k)$$

where $\delta_{j0}$ is the Kronecker delta and $x_i = \cos(\pi(i + 1/2)/(t + 1))$ for $k = 0, 1, 2, \ldots, t$; see [26]. Using the Chebyshev interpolant of $\log(1 + \alpha x)$, we approximate $\log|\tilde{K}|$ by

$$\log|\tilde{K}| - n\log\beta = \log|I + \alpha B| \approx \sum_{j=0}^{t} c_j \operatorname{tr}(T_j(B))$$

when $B = (\tilde{K}/\beta - 1)/\alpha$ has eigenvalues $\lambda_i \in (-1, 1)$.

For stochastic estimation of $\operatorname{tr}(T_j(B))$, we only need to compute $z^T T_j(B) z$ for each given probe vector $z$. We compute vectors $w_j = T_j(B)z$ and $\partial w_j/\partial\theta_i$ via the coupled recurrences

$$\begin{aligned} w_0 = z, \quad w_1 = Bz, \quad w_{j+1} = 2Bw_j - w_{j-1} \text{ for } j \geq 1, \\ \frac{\partial w_0}{\partial\theta_i} = 0, \quad \frac{\partial w_1}{\partial\theta_i} = \frac{\partial B}{\partial\theta_i} z, \quad \frac{\partial w_{j+1}}{\partial\theta_i} = 2\left(\frac{\partial B}{\partial\theta_i} w_j + B\frac{\partial w_j}{\partial\theta_i}\right) - \frac{\partial w_{j-1}}{\partial\theta_i} \text{ for } j \geq 1. \end{aligned}$$

This gives the estimators

$$\log|\tilde{K}| \approx \mathbb{E}\left[\sum_{j=0}^{t} c_j z^T w_j\right] \quad \text{and} \quad \frac{\partial}{\partial\theta_i}\log|\tilde{K}| \approx \mathbb{E}\left[\sum_{j=0}^{t} c_j z^T \frac{\partial w_j}{\partial\theta_i}\right].$$

Thus, each derivative of the approximation costs two extra MVMs per term.

### 2.2.2 Lanczos decomposition

We can also approximate $z^T \log(\tilde{K})z$ via a Lanczos decomposition; see [28] for discussion of a Lanczos-based computation of $z^T f(\tilde{K})z$ and [68, 2] for stochastic Lanczos estimation of log determinants. We run $t$ steps of the Lanczos algorithm, which computes the decomposition

$$\tilde{K}Q_t = Q_t T + \beta_t q_{t+1} e_t^T$$

where $Q_t = \begin{bmatrix} q_1 & q_2 & \dots q_t \end{bmatrix} \in \mathbb{R}^{n \times t}$ is a matrix with orthonormal columns such that $q_1 = z/\|z\|$, $T \in \mathbb{R}^{t \times t}$ is tridiagonal, $\beta_t$ is the residual, and $e_t$ is the $t$th Cartesian unit vector. We estimate

$$z^T f(\tilde{K})z \approx e_1^T f(\|z\|^2 T)e_1 \tag{2.3}$$

where $e_1$ is the first column of the identity. The Lanczos algorithm is numerically unstable. Several practical implementations resolve this issue [10, 61]. The approximation (2.3) corresponds to a Gauss quadrature rule for the Riemann-Stieltjes integral of the measure associated with the eigenvalue distribution of $\tilde{K}$. It is exact when $f$ is a polynomial of degree up to $2t - 1$. This approximation is also exact when $\tilde{K}$ has at most $t$ distinct eigenvalues, which is particularly relevant to Gaussian process regression, since frequently the kernel matrices only have a small number of eigenvalues that are not close to zero.

The Lanczos decomposition also allows us to estimate derivatives of the log determinant at minimal cost. Via the Lanczos decomposition, we have

$$\hat{g} = Q_t(T^{-1}e_1\|z\|) \approx \tilde{K}^{-1}z.$$

This approximation requires no additional matrix vector multiplications beyond those used to compute the Lanczos decomposition, which we already

used to estimate $\log(\tilde{K})z$; in exact arithmetic, this is equivalent to $t$ steps of CG. Computing $\hat{g}$ in this way takes $O(tn)$ additional time; subsequently, we only need one matrix-vector multiply by $\partial\tilde{K}/\partial\theta_i$ for each probe vector to estimate $\mathrm{tr}(\tilde{K}^{-1}(\partial\tilde{K}/\partial\theta_i)) = \mathbb{E}[(\tilde{K}^{-1}z)^T(\partial\tilde{K}/\partial\theta_i)z]$.

### 2.2.3 Estimating higher derivatives

We have already described how to use stochastic estimators to compute the log marginal likelihood and its first derivatives. The same approach applies to computing higher-order derivatives for a Newton-like iteration, to understand the sensitivity of the maximum likelihood parameters, or for similar tasks. The first derivatives of the full log marginal likelihood are

$$\frac{\partial\mathcal{L}}{\partial\theta_i} = -\frac{1}{2}\left[\mathrm{tr}\left(\tilde{K}^{-1}\frac{\partial\tilde{K}}{\partial\theta_i}\right) - \alpha^T\frac{\partial\tilde{K}}{\partial\theta_i}\alpha\right]$$

and the second derivatives of the two terms are

$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\left[\log|\tilde{K}|\right] = \mathrm{tr}\left(\tilde{K}^{-1}\frac{\partial^2\tilde{K}}{\partial\theta_i\partial\theta_j} - \tilde{K}^{-1}\frac{\partial\tilde{K}}{\partial\theta_i}\tilde{K}^{-1}\frac{\partial\tilde{K}}{\partial\theta_j}\right),$$
$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\left[(y-\mu_X)^T\alpha\right] = 2\alpha^T\frac{\partial\tilde{K}}{\partial\theta_i}\tilde{K}^{-1}\frac{\partial\tilde{K}}{\partial\theta_j}\alpha - \alpha^T\frac{\partial^2\tilde{K}}{\partial\theta_i\partial\theta_j}\alpha.$$

Superficially, evaluating the second derivatives would appear to require several additional solves above and beyond those used to estimate the first derivatives of the log determinant. In fact, we can get an unbiased estimator for the second derivatives with no additional solves, but only fast products with the derivatives of the kernel matrices. Let $z$ and $w$ be independent probe vectors, and define $g = \tilde{K}^{-1}z$, $h = \tilde{K}^{-1}w$, and $\alpha = \tilde{K}^{-1}(y_X - \mu_X)$. Then

$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\left[\log|\tilde{K}|\right] = \mathbb{E}\left[g^T\frac{\partial^2\tilde{K}}{\partial\theta_i\partial\theta_j}z - \left(g^T\frac{\partial\tilde{K}}{\partial\theta_i}w\right)\left(h^T\frac{\partial\tilde{K}}{\partial\theta_j}z\right)\right],$$
$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\left[(y-\mu_X)^T\alpha\right] = 2\mathbb{E}\left[\left(z^T\frac{\partial\tilde{K}}{\partial\theta_i}\alpha\right)\left(g^T\frac{\partial\tilde{K}}{\partial\theta_j}\alpha\right)\right] - \alpha^T\frac{\partial^2\tilde{K}}{\partial\theta_i\partial\theta_j}\alpha.$$

Hence, if we use the stochastic Lanczos method to compute the log determinant and its derivatives, the additional work required to obtain a second derivative estimate is one MVM by each second partial of the kernel for each probe vector and for $\alpha$, one MVM of each first partial of the kernel with $\alpha$, and a few dot products.

## 2.3 Error properties

In addition to the usual errors from sources such as solver termination criteria and floating point arithmetic, our approach to kernel learning involves several additional sources of error: we approximate the true kernel with one that enables fast MVMs, we approximate traces using stochastic estimation, and we approximate the actions of $\log(\tilde{K})$ and $\tilde{K}^{-1}$ on probe vectors.

We can compute first-order estimates of the sensitivity of the log likelihood to perturbations in the kernel using the same stochastic estimators we use for the derivatives with respect to hyperparameters. For example, if $\mathcal{L}^{\text{ref}}$ is the likelihood for a reference kernel $\tilde{K}^{\text{ref}} = \tilde{K} + E$, then

$$\mathcal{L}^{\text{ref}}(\theta|y) = \mathcal{L}(\theta|y) - \frac{1}{2}\left(\mathbb{E}\left[g^T E z\right] - \alpha^T E \alpha\right) + O(\|E\|^2),$$

and we can bound the change in likelihood at first order by $\|E\|\left(\|g\|\|z\| + \|\alpha\|^2\right)$. Given bounds on the norms of $\partial E/\partial\theta_i$, we can similarly estimate changes in the gradient of the likelihood, allowing us to bound how the marginal likelihood hyperparameter estimates depend on kernel approximations.

If $\tilde{K} = U\Lambda U^T + \sigma^2 I$, the Hutchinson trace estimator has known variance [1]

$$\text{Var}[z^T \log(\tilde{K}) z] = \sum_{i \neq j} [\log(\tilde{K})]_{ij}^2 \leq \sum_{i=1}^{n} \log(1 + \lambda_j/\sigma^2)^2.$$

If the eigenvalues of the kernel matrix without noise decay rapidly enough compared to $\sigma$, the variance will be small compared to the magnitude of $\mathrm{tr}(\log \tilde{K}) = 2n \log \sigma + \sum_{i=1}^{n} \log(1 + \lambda_j/\sigma^2)$. Hence, we need fewer probe vectors to obtain reasonable accuracy than one would expect from bounds that are blind to the matrix structure. In our experiments, we typically only use 5–10 probes — and we use the sample variance across these probes to estimate *a posteriori* the stochastic component of the error in the log likelihood computation. If we are willing to estimate the Hessian of the log likelihood, we can increase rates of convergence for finding kernel hyperparameters.

The Chebyshev approximation scheme requires $O(\sqrt{\kappa} \log(\kappa/\epsilon))$ steps to obtain an $O(\epsilon)$ approximation error in computing $z^T \log(\tilde{K})z$, where $\kappa = \lambda_{\max}/\lambda_{\min}$ is the condition number of $\tilde{K}$ [32]. This behavior is independent of the distribution of eigenvalues within the interval $[\lambda_{\min}, \lambda_{\max}]$, and is close to optimal when eigenvalues are spread quasi-uniformly across the interval. Nonetheless, when the condition number is large, convergence may be quite slow. The Lanczos approach converges at least twice as fast as Chebyshev in general [68, Remark 1], and converges much more rapidly when the eigenvalues are *not* uniform within the interval, as is the case with log determinants of many kernel matrices. Hence, we recommend the Lanczos approach over the Chebyshev approach in general. In all of our experiments, the error associated with approximating $z^T \log(\tilde{K})z$ by Lanczos was dominated by other sources of error.

### 2.3.1 Comparison to a reference kernel

Suppose more generally that $\tilde{K} = K + \sigma^2 I$ is an approximation to a reference kernel matrix $\tilde{K}^{\text{ref}} = K^{\text{ref}} + \sigma^2 I$, and let $E = K^{\text{ref}} - K$. Let $\mathcal{L}(\theta|y)$ and $\mathcal{L}^{\text{ref}}(\theta|y)$ be the log likelihood functions for the two kernels; then

$$\mathcal{L}^{\text{ref}}(\theta|y) = \mathcal{L}(\theta|y) - \frac{1}{2}\left[\text{tr}(\tilde{K}^{-1}E) - \alpha^T E\alpha\right] + O(\|E\|^2)$$
$$\frac{\partial}{\partial\theta_i}\mathcal{L}^{\text{ref}}(\theta|y) = \frac{\partial}{\partial\theta_i}\mathcal{L}(\theta|y) - \frac{1}{2}\left[\text{tr}\left(\tilde{K}^{-1}\frac{\partial E}{\partial\theta_i} - \tilde{K}^{-1}\frac{\partial\tilde{K}}{\partial\theta_i}\tilde{K}^{-1}E\right) - \alpha^T\frac{\partial E}{\partial\theta_i}\alpha\right] + O(\|E\|^2).$$

If we are willing to pay the price of a few MVMs with $E$, we can use these expressions to improve our maximum likelihood estimate. Let $z$ and $w$ be independent probe vectors with $g = \tilde{K}^{-1}z$ and $\hat{g} = \tilde{K}^{-1}w$. To estimate the trace in the derivative computation, we use the standard stochastic trace estimation approach together with the observation that $\mathbb{E}[ww^T] = I$:

$$\text{tr}\left(\tilde{K}^{-1}\frac{\partial E}{\partial\theta_i} - \tilde{K}^{-1}\frac{\partial\tilde{K}}{\partial\theta_i}\tilde{K}^{-1}E\right) = \mathbb{E}\left[g^T\frac{\partial E}{\partial\theta_i}z - g^T\frac{\partial K}{\partial\theta_i}w\hat{g}^T Ez\right]$$

This linearization may be used directly (with a stochastic estimator); alternately, if we have an estimates for $\|E\|$ and $\|\partial E/\partial\theta_i\|$, we can substitute these in order to get estimated bounds on the magnitude of the derivatives. Coupled with a similar estimator for the Hessian of the likelihood function (described in the supplementary materials), we can use this method to compute the maximum likelihood parameters for the fast kernel, then compute a correction $-H^{-1}\nabla_\theta\mathcal{L}^{\text{ref}}$ to estimate the maximum likelihood parameters of the reference kernel.

## 2.4 Experiments

We test our stochastic trace estimator with both Chebyshev and Lanczos approximation schemes on a large set of diverse numerical experiments. Through-

out we use the SKI method [75] of Eq. (2.1) for fast MVMs. We find that the Lanczos is able to do kernel recovery and inference significantly faster and more accurately than competing methods.

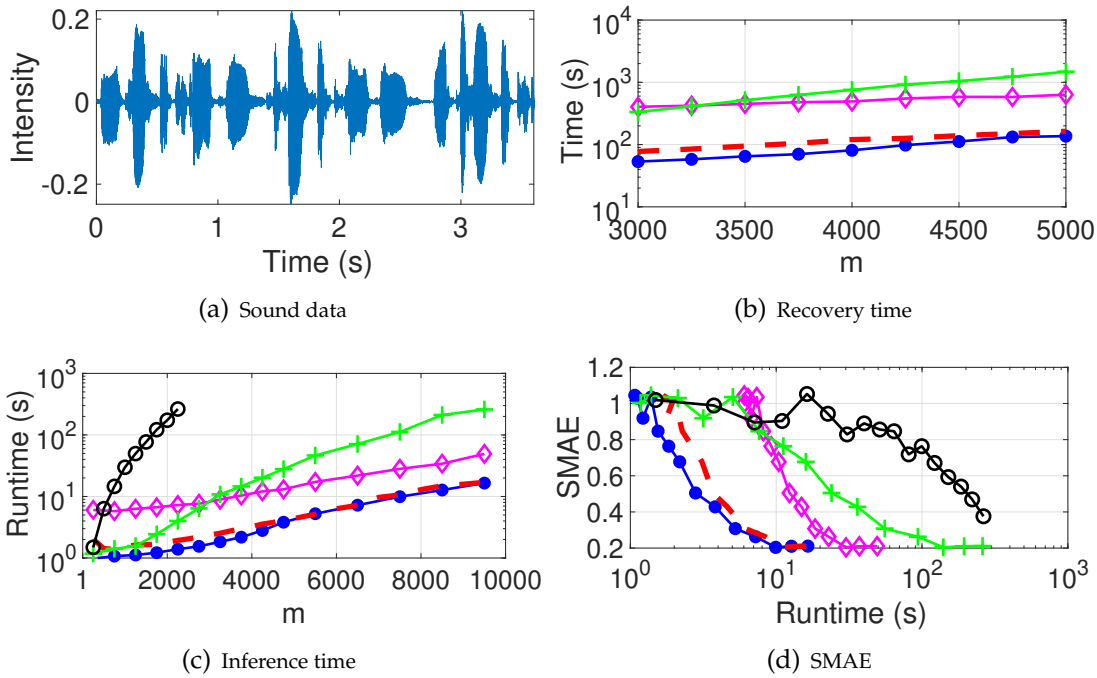
### 2.4.1 Natural sound modeling

Here we consider the natural sound benchmark in [75], shown in Figure 2.1(a). Our goal is to recover contiguous missing regions in a waveform with $n = 59,306$ training points. We exploit Toeplitz structure in the $K_{UU}$ matrix of our SKI approximate kernel for accelerated MVMs.

The experiment in [75] only considered scalable inference and prediction, but not hyperparameter learning, since the scaled eigenvalue approach requires all the eigenvalues for an $q \times q$ Toeplitz matrix, which can be computationally prohibitive with cost $O(q^2)$. However, evaluating the marginal likelihood on this training set is not an obstacle for Lanczos and Chebyshev since we can use fast MVMs with the SKI approximation at a cost of $O(n + q \log q)$.

In Figure 2.1(b), we show how Lanczos, Chebyshev, and an RBF (surrogate) based on precomputed logdet values to scale with the number of inducing points $q$ compared to the scaled eigenvalue method and FITC. We use 5 probe vectors and 25 iterations for Lanczos, both when building the surrogate and for hyperparameter learning with Lanczos. We also use 5 probe vectors for Chebyshev and 100 moments. Figure 2.1(b) shows the runtime of the hyperparameter learning phase for different numbers of inducing points $q$, where Lanczos and the surrogate are clearly more efficient than scaled eigenvalues and Chebyshev. For hyperparameter learning, FITC took several hours to run, compared

to minutes for the alternatives; we therefore exclude FITC from Figure 2.1(b). Figure 2.1(c) shows the time to do inference on the 691 test points, while 2.1(d) shows the standardized mean absolute error (SMAE) on the same test points. As expected, Lanczos and surrogate make accurate predictions much faster than Chebyshev, scaled eigenvalues, and FITC. In short, Lanczos and the surrogate approach are much faster than alternatives for hyperparameter learning with a large number of inducing points and training points.




(a) Sound data (b) Recovery time

(c) Inference time (d) SMAE

Figure 2.1: Sound modeling using 59,306 training points and 691 test points. The intensity of the time series can be seen in (a). Train time for SE kernel hyperparameters is in (b) and the time for inference is in (c). The standardized mean absolute error (SMAE) as a function of time for an evaluation of the marginal likelihood and all derivatives is shown in (d). Surrogate is (——), Lanczos is (- - -), Chebyshev is (— ◇ —), scaled eigenvalues is (— + —), and FITC is (— o —).

### 2.4.2 Daily precipitation prediction

This experiment involves precipitation data from the year of 2010 collected from around 5500 weather stations in the US[1]. The hourly precipitation data is pre-processed into daily data if full information of the day is available. The dataset has $628,474$ entries in terms of precipitation per day given the date, longitude and latitude. We randomly select $100,000$ data points as test points and use the remaining points for training. We then perform hyperparameter learning and prediction with the SE kernel, using Lanczos, scaled eigenvalues, and exact.

For Lanczos and scaled eigenvalues, we optimize the hyperparameters on the subset of data for January 2010, with an induced grid of 100 points per spatial dimension and 300 in the temporal dimension. Due to memory constraints we only use a subset of $12,000$ entries for training with the exact method. While scaled eigenvalues can perform well when fast eigendecompositions are possible, as in this experiment, Lanczos nonetheless still runs faster and with slightly lower MSE.

| Method | $n$ | $q$ | MSE | Time [min] |
|---|---|---|---|---|
| Lanczos | 528k | 3M | 0.613 | 14.3 |
| Scaled eigenvalues | 528k | 3M | 0.621 | 15.9 |
| Exact | 12k | - | 0.903 | 11.8 |

Table 2.1: Prediction comparison for the daily precipitation data showing the number of training points $n$, number of induced grid points $q$, the mean squared error, and the inference time.

Incidentally, we are able to use 3 *million* inducing points in Lanczos and scaled eigenvalues, which is enabled by the SKI representation [75] of covari-

[1] `https://catalog.data.gov/dataset/u-s-hourly-precipitation-data`

ance matrices, for a a very accurate approximation. This number of inducing points $q$ is unprecedented for typical alternatives which scale as $O(q^3)$.

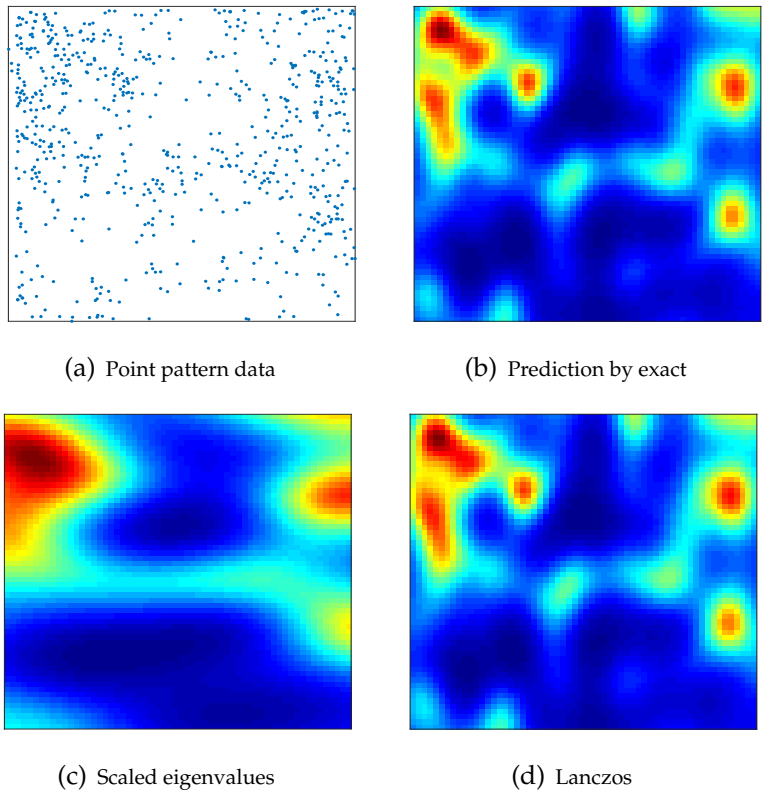
### 2.4.3 Hickory data

In this experiment, we apply Lanczos to the log-Gaussian Cox process model with a Laplace approximation for the posterior distribution. We use the SE kernel and the Poisson likelihood in our model. The scaled eigenvalue method does not apply directly to non-Gaussian likelihoods; we thus applied the scaled eigenvalue method in [75] in conjunction with the Fiedler bound in [19] for the scaled eigenvalue comparison. Indeed, a key advantage of the Lanczos approach is that it can be applied whenever fast MVMs are available, which means no additional approximations such as the Fiedler bound are required for non-Gaussian likelihoods.

This dataset, which comes from the R package `spatstat`, is a point pattern of 703 hickory trees in a forest in Michigan. We discretize the area into a $60 \times 60$ grid and fit our model with exact, scaled eigenvalues, and Lanczos. We see in Table 2.2 that Lanczos recovers hyperparameters that are much closer to the exact values than the scaled eigenvalue approach. Figure 2.2 shows that the predictions by Lanczos are also indistinguishable from the exact computation.

| Method | $s$ | $\ell_1$ | $\ell_2$ | $-\log p(y\|\theta)$ | Time [s] |
|---|---|---|---|---|---|
| Exact | 0.696 | 0.063 | 0.085 | 1827.56 | 465.9 |
| Lanczos | 0.693 | 0.066 | 0.096 | 1828.07 | 21.4 |
| Scaled eigenvalues | 0.543 | 0.237 | 0.112 | 1851.69 | 2.5 |

Table 2.2: Hyperparameters recovered on the Hickory dataset.



(a) Point pattern data (b) Prediction by exact

(c) Scaled eigenvalues (d) Lanczos

Figure 2.2: Predictions by exact, scaled eigenvalues, and Lanczos on the Hickory dataset.

### 2.4.4 Crime prediction

In this experiment, we apply Lanczos with the spectral mixture kernel to the crime forecasting problem considered in [19]. This dataset consists of $233,088$ incidents of assault in Chicago from January 1, 2004 to December 31, 2013. We use the first 8 years for training and attempt to predict the crime rate for the last 2 years. For the spatial dimensions, we use the log-Gaussian Cox process model, with the Matérn-5/2 kernel, the negative binomial likelihood, and the Laplace approximation for the posterior. We use a spectral mixture kernel with 20 components and an extra constant component for the temporal dimension. We discretize the data into a $17 \times 26$ spatial grid corresponding to 1-by-1 mile

grid cells. In the temporal dimension we sum our data by weeks for a total of 522 weeks. After removing the cells that are outside Chicago, we have a total of $157,644$ observations.

The results for Lanczos and scaled eigenvalues (in conjunction with the Fiedler bound due to the non-Gaussian likelihood) can be seen in Table 2.3. The Lanczos method used 5 Hutchinson probe vectors and 30 Lanczos steps. For both methods we allow 100 iterations of LBFGS to recover hyperparameters and we often observe early convergence. While the RMSE for Lanczos and scaled eigenvalues happen to be close on this example, the recovered hyperparameters using scaled eigenvalues are very different than for Lanczos. For example, the scaled eigenvalue method learns a much larger $\sigma^2$ than Lanczos, indicating model misspecification. In general, as the data become increasingly non-Gaussian the Fiedler bound (used for fast scaled eigenvalues on non-Gaussian likelihoods) will become increasingly misspecified, while Lanczos will be unaffected.

| Method | $\ell_1$ | $\ell_2$ | $\sigma^2$ | $T_{\text{recovery}}$[s] | $T_{\text{prediction}}$[s] | $\text{RMSE}_{\text{train}}$ | $\text{RMSE}_{\text{test}}$ |
|---|---|---|---|---|---|---|---|
| Lanczos | 0.65 | 0.67 | 69.72 | 264 | 10.30 | 1.17 | 1.33 |
| Scaled eigenvalues | 0.32 | 0.10 | 191.17 | 67 | 3.75 | 1.19 | 1.36 |

Table 2.3: Hyperparameters recovered, recovery time and RMSE for Lanczos and scaled eigenvalues on the Chicago assault data. Here $\ell_1$ and $\ell_2$ are the length scales in spatial dimensions and $\sigma^2$ is the noise level. $T_{\text{recovery}}$ is the time for recovering hyperparameters. $T_{\text{prediction}}$ is the time for prediction at all $157,644$ observations (including training and testing).

### 2.4.5 Deep kernel learning

To handle high-dimensional datasets, we bring our methods into the deep kernel learning framework [79] by replacing the final layer of a pre-trained deep neural network (DNN) with a GP. This experiment uses the gas sensor dataset from the UCI machine learning repository. It has 2565 instances with 128 dimensions. We pre-train a DNN, then attach a GP with SE kernels to the two-dimensional output of the second-to-last layer. We then further train all parameters of the resulting kernel, *including* the weights of the DNN, through the GP marginal likelihood. In this example, Lanczos and the scaled eigenvalue approach perform similarly well. Nonetheless, we see that Lanczos can effectively be used with SKI on a high dimensional problem to train hundreds of thousands of kernel parameters.

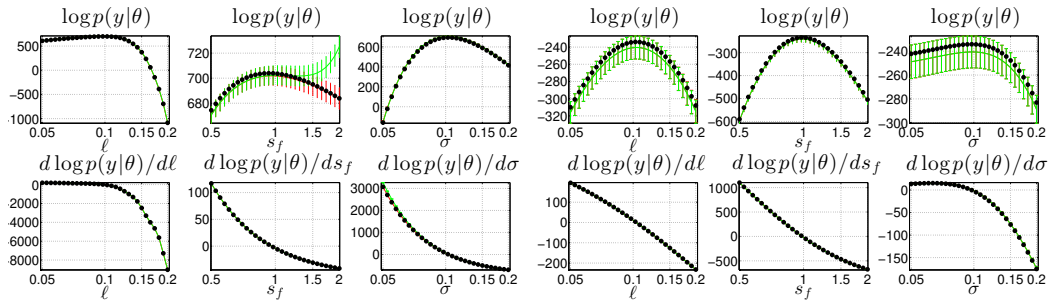
| Method | DNN | Lanczos | Scaled eigenvalues |
|---|---|---|---|
| RMSE | $0.1366 \pm 0.0387$ | $0.1053 \pm 0.0248$ | $0.1045 \pm 0.0228$ |
| Time [s] | 0.4438 | 2.0680 | 1.6320 |

Table 2.4: Prediction RMSE and per training iteration runtime.

### 2.4.6 1D cross-section plots

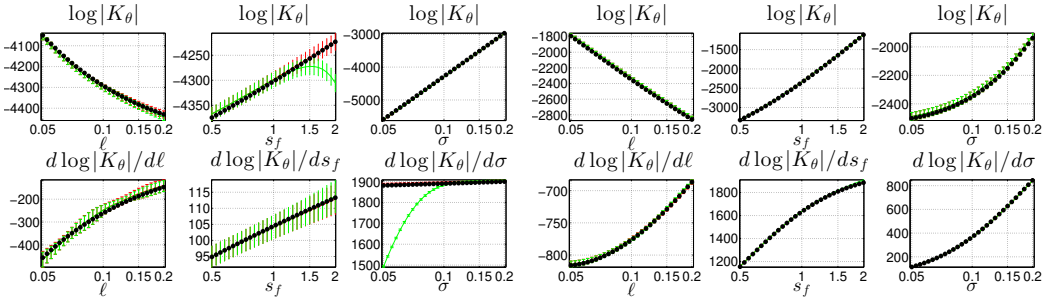
In this experiment we compare the accuracy of Lanczos and Chebyshev for 1-dimensional perturbations of a set of true hyper-parameters, and demonstrate how critical it is to use diagonal replacement for some approximate kernels. We choose the true hyper-parameters to be $(\ell, s, \sigma) = (0.1, 1, 0.1)$ and consider two different types of datasets. The first dataset consists of 1000 equally spaced points in the interval $[0, 4]$ in which case the kernel matrix of a stationary kernel

is Toeplitz and we can make use of fast matrix-vector multiplication. The second dataset consists of 1000 data points drawn independently from a $U(0, 4)$ distribution. We use SKI with cubic interpolation to construct an approximate kernel based on 1000 equally spaced points. The function values are drawn from a GP with the true hyper-parameters, for both the true and approximate kernel. We use 250 iterations for Lanczos and 250 Chebyshev moments in order to assure convergence of both methods. The results for the first dataset with the SE and Matérn kernels can be seen in Figure 2.3(a)-2.3(d). The results for the second dataset with the SKI kernel can be seen in Figure 2.4(a)-2.4(d).




(a) log marginal likelihood for the SE kernel

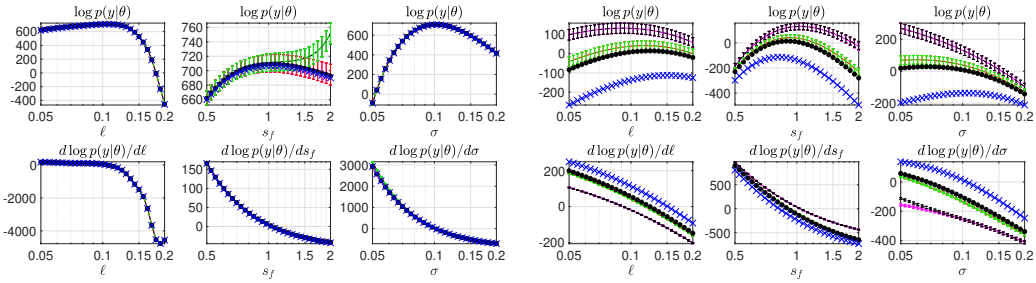
(b) log marginal likelihood for the Matérn kernel




(c) log determinant for the RBF kernel

(d) log determinant for the Matérn kernel

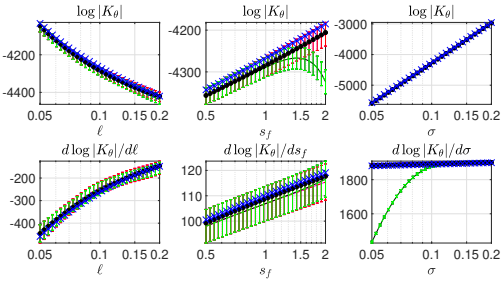
Figure 2.3: 1-dimensional perturbations for the exact SE and Matérn 1/2 kernel where the data is 1000 equally spaced points in the interval $[0, 4]$. The exact values are (•), Lanczos is (——), Chebyshev is (——). The error bars of Lanczos and Chebyshev are 1 standard deviation and were computed from 10 runs with different probe vectors



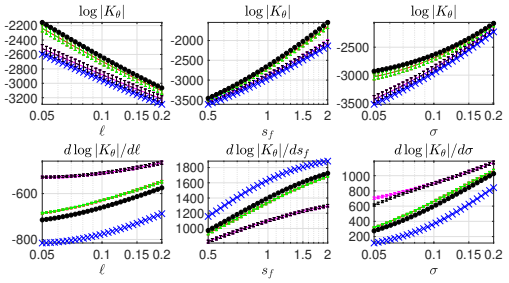
(a) log marginal likelihood for the SE kernel



(b) log marginal likelihood for the Matérn kernel



(c) log determinant for the RBF kernel



(d) log determinant for the Matérn kernel

Figure 2.4: 1-dimensional perturbations of the SKI (cubic) approximations of the RBF and Matérn 1/2 kernel where the data is 1000 points drawn from $\mathcal{N}(0, 2)$. The exact values are (•), Lanczos with diagonal replacement is (——), Chebyshev with diagonal replacement is (——), Lanczos without diagonal replacement is (——), Chebyshev without diagonal replacement is (——), and scaled eigenvalues is (×). Diagonal replacement makes no perceptual difference for the SE kernel so the lines are overlapping in this case. The error bars of Lanczos and Chebyshev are 1 standard deviation and were computed from 10 runs with different probe vectors

Lanczos yields an excellent approximation to the log determinant and its derivatives for both the exact and the approximate kernels, while Chebyshev struggles with large values of $s$ and small values of $\sigma$ on the exact and approximate SE kernel. This is expected since Chebyshev has issues with the singularity at zero while Lanczos has large quadrature weights close to zero to compensate for this singularity. The scaled eigenvalue method has issues with the approximate Matérn 1/2 kernel.

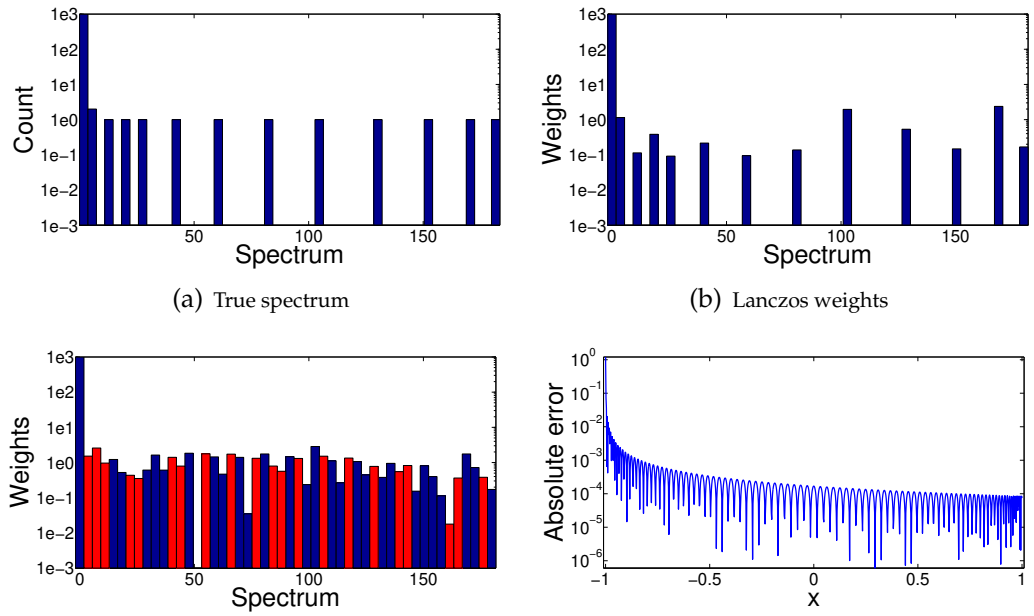
### 2.4.7 Why Lanczos is better than Chebyshev

In this experiment, we study the performance advantage of Lanczos over Chebyshev. Figure 2.5 shows that the Ritz values of Lanczos quickly converge to the spectrum of the SE kernel thanks to the absence of interior eigenvalues. The Chebyshev approximation shows the expected equioscillation behavior. More importantly, the Chebyshev approximation for logarithms has its greatest error near zero where the majority of the eigenvalues are, and those also have the heaviest weight in the log determinant.

Another advantage of Lanczos is that it requires minimal knowledge of the spectrum, while Chebyshev needs the extremal eigenvalues for rescaling. In addition, with Lanczos we can get the derivatives with only one MVM per hyperparameter, while Chebyshev requires an MVM at each iteration, leading to extra computation and memory usage.

### 2.4.8 The importance of diagonal correction

This experiment shows that diagonal correction of the approximate kernel can be very important. Diagonal correction cannot be used efficiently for some methods, such as the scaled eigenvalue method, and this may hurt its predictive performance. Our experiment is similar to [54]. We generate 1000 uniformly distributed points in the interval $[-10, 10]$, and we choose a small number of inducing points in such a way that there is a large chunk of the interval where there is no inducing point. We are interested in the behavior of the predictive uncertainties on this subinterval. The function values are given by $f(x) = 1 + x/2 + \sin(x)$ and normally distributed noise with standard deviation




(a) True spectrum

(b) Lanczos weights

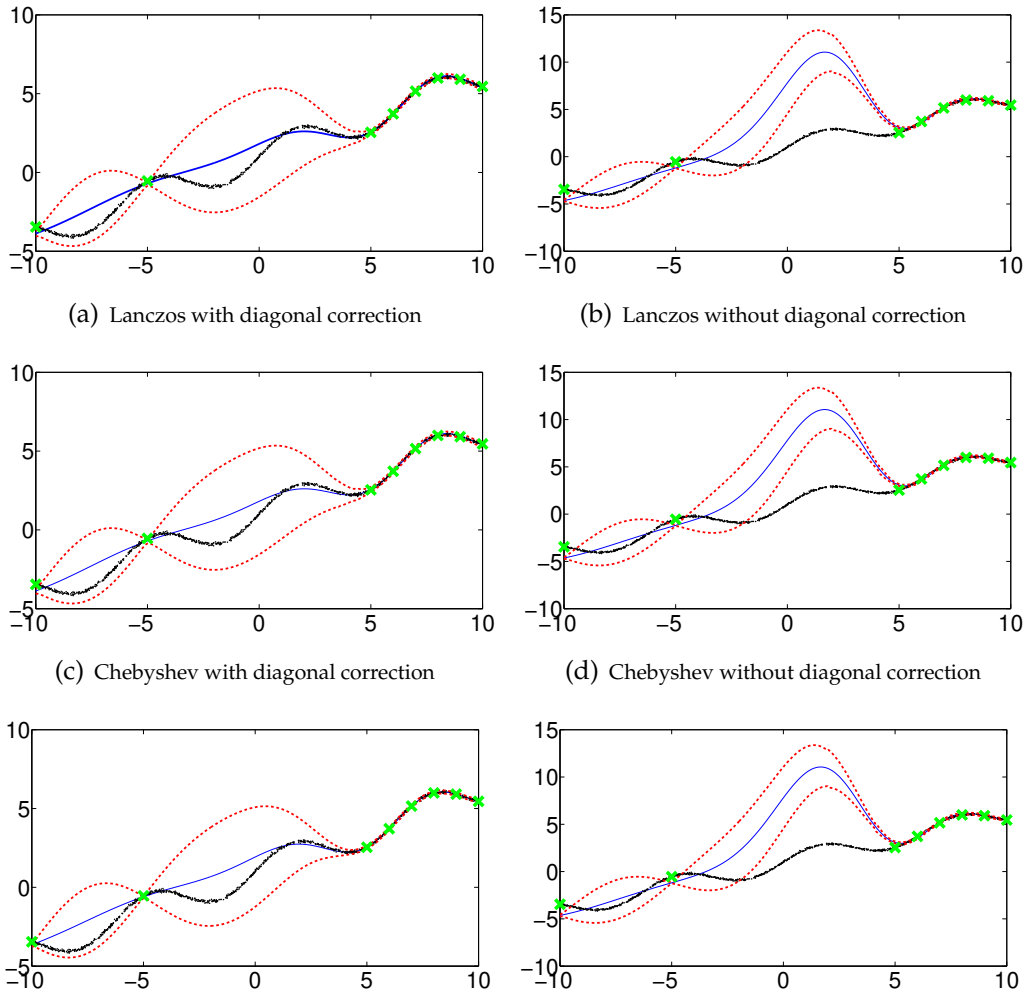
(c) Chebyshev weights

(d) Chebyshev absolute error

Figure 2.5: A comparison between the true spectrum, the Lanczos weights ($t = 50$), and the Chebyshev weights ($t = 100$) for the SE kernel with $\ell = 0.3$, $s = 1$, and $\sigma = 0.1$. All weights and counts are on a log-scale so that they are easier to compare. Blue bars correspond to positive weights while red bars correspond to negative weights.

0.05 is added to the function values. We find the optimal hyper-parameters of the Matérn 3/2 using the exact method and use these hyper-parameters to make predictions with Lanczos, Chebyshev, FITC, and the scaled eigenvalue method. We consider Lanczos both with and without diagonal correction in order to see how this affects the predictions. The results can be seen in Figure 2.6.

It is clear that Lanczos and Chebyshev are too confident in the predictive mean when diagonal correction is not used, while the predictive uncertainties agree well with FITC when diagonal correction is used. The scaled eigenvalue method cannot be used efficiently with diagonal correction and we see that this leads to predictions similar to Lanczos and Chebyshev without diagonal correc-



(a) Lanczos with diagonal correction

(b) Lanczos without diagonal correction

(c) Chebyshev with diagonal correction

(d) Chebyshev without diagonal correction

(e) FITC

(f) Scaled eigenvalue method

Figure 2.6: Example that shows how important diagonal correction can be for some kernels. The Matérn 3/2 kernel was used to fit the data given by the black dots. This data was generated from the function $f(x) = 1 + x/2 + \sin(x)$ to which we added normally distributed noise with standard deviation 0.05. We used the exact method to find the optimal hyper-parameters and used these hyper-parameters to study the different behavior of the predictive uncertainties when the inducing points are given by the green crosses. The solid blue line is the predictive mean and the dotted red lines shows a confidence interval of two standard deviations.

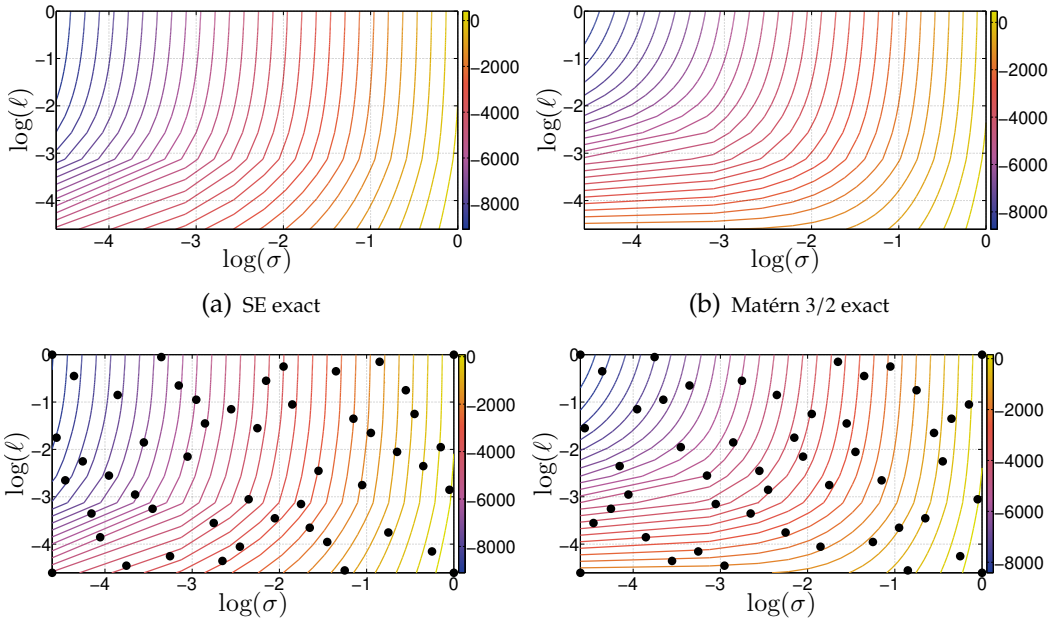
tion. The flexibility of being able to use diagonal correction with Lanczos and Chebyshev makes these approaches very appealing.

### 2.4.9 Surrogate log determinant approximation

The point of this experiment is to illustrate how accurate the level-curves of the surrogate model are compared to the level-curves of the true log determinant. We consider the SE and the Matérn 3/2 kernels and the same datasets that we considered in 2.4.6. We fix $s = 1$ and study how the level curves compare when we vary $\ell$ and $\sigma$. Building the surrogate with all three hyper-parameters produces similar results, but requires more design points. We use 50 design points to construct a cubic RBF with a linear tail. The values of the log determinant and its derivatives are computed with Lanczos. It is clear from Figure 2.7 that the surrogate model does a good job approximating the log determinant for both kernels.

### 2.4.10 Kernel hyper-parameter recovery

This experiments tests how well we can recover hyper-parameters from data generated from a GP. We compare Chebyshev, Lanczos, the surrogate, the scaled eigenvalue method, and FITC. We consider a dataset of 5000 points generated from a $\mathcal{N}(0, 2)$ distribution. We use SKI with cubic interpolation and a total of 2000 inducing points for Lanczos, Chebyshev, and then scaled eigenvalue method. FITC was used with 750 equally spaced points because it has a longer runtime as a function of the number of inducing points. We consider the SE kernel and the Matérn 3/2 kernel and sample from a GP with ground truth parameters $(\ell, s, \sigma) = (0.01, 0.5, 0.05)$. The GPs for which we try to recover the hyper-parameters were generated from the original kernel. It is important to emphasize that there are two sources of errors present: the error from the kernel




(a) SE exact (b) Matérn 3/2 exact

(c) SE surrogate (d) Matérn 3/2 surrogate

Figure 2.7: Level curves of the exact and surrogate approximation of the log determinant as a function of $\ell$ and $\sigma$ for the SE and Matérn 3/2 kernels. We used $s = 1$ and the dataset consisted of 1000 equally spaced points in the interval $[0, 4]$. The surrogate model was constructed from the points shown with (•) and the log determinant values were computed using stochastic Lanczos.

approximation errors and the stochastic error from Lanczos and Chebyshev. We saw in Figure 2.3 and 2.4 that the stochastic error for Lanczos is relatively small, so this follow-up experiment helps us understand how Lanczos is influenced by the error incurred from an approximate kernel. We show the true log marginal likelihood, the recovered hyper-parameters, and the run-time in Table 2.5.

It is clear from Table 2.5 that most methods are able to recover parameters close to the ground truth for the SE kernel. The results are more interesting for the Matérn 3/2 kernel where FITC struggles and the parameters recovered by FITC have a value of the log marginal likelihood that is much worse than the other methods.

| | | RBF | Matérn 3/2 |
|---|---|---|---|
| True | $-\log p(y\|\theta)$ | −6.22e3 | −4.91e3 |
| | Hypers | (0.01, 0.5, 0.05) | (0.01, 0.5, 0.05) |
| Exact | $-\log p(y\|\theta)$ | −6.23e3 | −4.91e3 |
| | Hypers | (1.01e−2, 4.81e−1, 5.03e−2) | (9.63e−3, 4.87e−1, 4.96e−2) |
| | Time (s) | 368.9 | 466.7 |
| Lanczos | $-\log p(y\|\theta)$ | −6.22e3 | −4.86e3 |
| | Hypers | (1.00e−2, 4.77e−1, 5.03e−2) | (1.04e−2, 4.87e−1, 4.67e−2) |
| | Time (s) | 66.2 | 133.4 |
| Chebyshev | $-\log p(y\|\theta)$ | −6.23e3 | −4.81e3 |
| | Hypers | (9.84e−3, 4.85e−1, 5.12e−2) | (1.11e−2, 4.66e−1, 5.78e−2) |
| | Time (s) | 110.3 | 173.3 |
| Surrogate | $-\log p(y\|\theta)$ | −6.22e3 | −4.86e3 |
| | Hypers | (1.01e−2, 4.88e−1, 4.85e−2) | (1.02e−2, 4.80e−1, 4.66e−2) |
| | Time (s) | 48.2 | 44.3 |
| Scaled eigenvalues | $-\log p(y\|\theta)$ | −6.22e3 | −4.71e3 |
| | Hypers | (1.04e−2, 4.52e−1, 5.14e−2) | (1.13e−2, 4.53e−1, 6.37e−2) |
| | Time (s) | 90.2 | 127.3 |
| FITC | $-\log p(y\|\theta)$ | −6.22e3 | −4.11e3 |
| | Hypers | (1.03e−2, 4.90e−1, 5.07e−2) | (1.34e−2, 5.22e−1, 8.91e−2) |
| | Time (s) | 86.6 | 136.9 |

Table 2.5: Hyper-parameter recovery for the SE and Matérn 3/2 kernels. The data was generated from 5000 normally distributed points. Lanczos, surrogate, and scaled eigenvalues all used 2000 inducing points while FITC used 750. These numbers where chosen to make their run times close to equal. Diagonal correction was applied to the Matérn 3/2 approximate kernel. The value of the log marginal likelihood was was computed from the exact kernel and shows the value of the hyper-parameters recovered by each method. We ran Lanczos 5 times and averaged the values.

## 2.5 Conclusion

The methods presented in this chapter are general and rely only on fast matrix vector multiplications (MVMs) with the kernel matrix. These MVMs can be used to efficiently solve linear systems with the kernel matrix using the conjugate gradient (CG) method. The biggest computational challenge is the estimation of the log determinant and its derivatives, and we have illustrated the promise of combining stochastic trace estimation with the Lanczos process. We have shown the scalability and flexibility of our approach through experiments with kernel learning for several real-world data sets using both Gaussian and non-Gaussian likelihoods, and highly parametrized deep kernels. The next chapter will show how to extend the work in this chapter to incorporate gradient information in the GP model. We discuss several additional extensions in Chapter 6.

CHAPTER 3

# SCALABLE GAUSSIAN PROCESSES WITH DERIVATIVES

In this chapter we generalize the work from the previous chapter to incorporate derivative information into the GP model. For many simulation models, derivatives may be computed at little extra cost via finite differences, complex step approximations, adjoint methods, or algorithmic differentiation [22]. Exact kernel learning is ill-suited for GPs with derivatives, which requires $O(n^3d^3)$ computation and $O(n^2d^2)$ storage. We demonstrate our work on applications in Bayesian Optimization (BO) [81], implicit surface reconstruction [45], and terrain reconstruction. Most of the content in this chapter is based on [16].

## 3.1 Background

While many scalable approximation methods for GP regression have been proposed, scalable methods incorporating derivatives have received little attention. In this chapter, we propose scalable methods for GPs with derivative information built on the *structured kernel interpolation* (SKI) framework [75], which uses local interpolation to map scattered data onto a large grid of inducing points, enabling fast MVMs using FFTs. As the uniform grids in SKI scale poorly to high-dimensional spaces, we also extend the structured kernel interpolation for products (SKIP) method, which approximates a high-dimensional product kernel as a Hadamard product of low rank Lanczos decompositions [25]. Both SKI and SKIP provide fast approximate kernel MVMs, which are a building block to solve linear systems with the kernel matrix and to approximate log determinants [13].

The eigenspectrum of $K_{XX}^{\nabla}$, the kernel matrix with derivative information, may exhibit slow decay, despite $K_{XX}$ itself possessing fast spectral decay. Fast forward to Figure 3.1 for two examples. The unfavorable spectrum of $K_{XX}^{\nabla}$ implies low-rank approximation such as SoR and FITC are infeasible. As a result, preconditioning is necessary for convergence of any iterative method.

## 3.2 Scalable GPs with derivatives

One standard approach to scaling GPs substitutes the exact kernel with an approximate kernel. When the GP fits values and gradients, one may attempt to separately approximate the kernel and the kernel derivatives. Unfortunately, this may lead to indefiniteness, as the resulting approximation is no longer a valid kernel. Instead, we differentiate the approximate kernel, which preserves positive definiteness. We do this for the SKI and SKIP kernels below, but our general approach applies to any differentiable approximate MVM.

### 3.2.1 D-SKI

D-SKI (SKI with derivatives) is the standard kernel matrix for Gaussian processes with derivatives, but applied to the SKI kernel. Equivalently, we differentiate the interpolation scheme:

$$k(x, x') \approx \sum_i w_i(x) k(x_i, x') \rightarrow \nabla k(x, x') \approx \sum_i \nabla w_i(x) k(x_i, x').$$

One can use cubic convolutional interpolation [41], which we did in Chapter 2, but higher order methods lead to greater accuracy, and we therefore use quintic

interpolation [49]. The resulting D-SKI kernel matrix has the form

$$\begin{bmatrix} K_{XX} & (\partial K_{XX})^T \\ \partial K_{XX} & \partial^2 K_{XX} \end{bmatrix} \approx \begin{bmatrix} W \\ \partial W \end{bmatrix} K_{UU} \begin{bmatrix} W \\ \partial W \end{bmatrix}^T = \begin{bmatrix} WK_{UU}W^T & WK_{UU}(\partial W)^T \\ (\partial W)K_{UU}W^T & (\partial W)K_{UU}(\partial W)^T \end{bmatrix},$$

where the elements of sparse matrices $W$ and $\partial W$ are determined by $w_i(x)$ and $\nabla w_i(x)$ — assuming quintic interpolation, $W$ and $\partial W$ will each have $6^d$ elements per row. As with SKI with $q$ gridpoints, we use FFTs to obtain $O(q \log q)$ MVMs with $K_{UU}$. Because $W$ and $\partial W$ have $O(n6^d)$ and $O(nd6^d)$ nonzero elements, respectively, our MVM complexity is $O(nd6^d + q \log q)$.

### 3.2.2 D-SKIP

Several common kernels are *separable*, i.e. they can be expressed as products of one-dimensional kernels. Assuming a compatible interpolation scheme, this structure is inherited by the SKI approximation for the kernel matrix without derivatives,

$$K_{XX} \approx (W_1 K_1 W_1^T) \odot (W_2 K_2 W_2^T) \odot \ldots \odot (W_d K_d W_d^T),$$

where $A \odot B$ denotes the Hadamard product of matrices $A$ and $B$ with the same dimensions, and $W_j$ and $K_j$ denote the SKI interpolation and inducing point grid matrices in the $j$th coordinate direction. The same Hadamard product structure applies to the kernel matrix with derivatives; for example, for $d = 2$,

$$K_{XX}^{\nabla} \approx \begin{bmatrix} W_1 K_1 W_1^T & W_1 K_1 \partial W_1^T & W_1 K_1 W_1^T \\ \partial W_1 K_1 W_1^T & \partial W_1 K_1 \partial W_1^T & \partial W_1 K_1 W_1^T \\ W_1 K_1 W_1^T & W_1 K_1 \partial W_1^T & W_1 K_1 W_1^T \end{bmatrix} \odot \begin{bmatrix} W_2 K_2 W_2^T & W_2 K_2 W_2^T & W_2 K_2 \partial W_2^T \\ W_2 K_2 W_2^T & W_2 K_2 W_2^T & W_2 K_2 \partial W_2^T \\ \partial W_2 K_2 W_2^T & \partial W_2 K_2 W_2^T & \partial W_2 K_2 \partial W_2^T \end{bmatrix}. \quad (3.1)$$

Equation 3.1 expresses $K_{XX}^{\nabla}$ as a Hadamard product of one dimensional kernel matrices. Following this approximation, we apply the SKIP reduction [25] and use Lanczos to further approximate equation 3.1 as $(Q_1 T_1 Q_1^T) \odot (Q_2 T_2 Q_2^T)$. This can be used for fast MVMs with the kernel matrix, see the appendix for details. Applied to kernel matrices with derivatives, we call this approach D-SKIP. D-SKIP achieves better scaling with $d$ than D-SKI as constructing the D-SKIP kernel costs $O(d^2(n + p \log p + p^3 n \log d))$ flops, and each MVM costs $O(dp^2 n)$ flops where $p$ is the effective rank of the kernel at each step (rank of the Lanczos decomposition). We achieve high accuracy with $p \ll n$.

### 3.2.3 Preconditioning

Recent work has explored several preconditioners for exact kernel matrices without derivatives [12]. We have had success with preconditioners of the form $M = \sigma^2 I + FF^T$ where $K_{XX}^{\nabla} \approx FF^T$ with $F \in \mathbb{R}^{n \times p}$. Solving with the Sherman-Morrison-Woodbury formula (*a.k.a* the matrix inversion lemma) is inaccurate for small $\sigma$; we use the more stable formula $M^{-1}b = \sigma^{-2}(f - Q_1(Q_1^T b))$ where $Q_1$ is computed in $O(p^2 n)$ time by the economy QR factorization

$$\begin{bmatrix} F \\ \\ \sigma I \end{bmatrix} = \begin{bmatrix} Q_1 \\ \\ Q_2 \end{bmatrix} R.$$

In our experiments with solvers for D-SKI and D-SKIP, we have found that a truncated pivoted Cholesky factorization, $K_{XX}^{\nabla} \approx (\Pi L)(\Pi L)^T$ works well for the low-rank factorization. Computing the pivoted Cholesky factorization is cheaper than MVM-based preconditioners such as Lanczos or truncated eigendecompositions as it only requires the diagonal and the ability to form the rows

where pivots are selected. Pivoted Cholesky is a natural choice when inducing point methods are applied as the pivoting can itself be viewed as an inducing point method where the most important information is selected to construct a low-rank preconditioner [34]. The D-SKI diagonal can be formed in $O(nd6^d)$ flops while rows cost $O(nd6^d + q)$ flops; for D-SKIP both the diagonal and the rows can be formed in $O(nd)$ flops.

### 3.2.4 Dimensionality reduction

In many high-dimensional function approximation problems, only a few directions are relevant. That is, if $f : \mathbb{R}^d \to \mathbb{R}$ is a function to be approximated, there is often a matrix $P$ with $\tilde{d} < d$ orthonormal columns spanning an *active subspace* of $\mathbb{R}^d$ such that $f(x) \approx f(PP^T x)$ for all $x$ in some domain $\Omega$ of interest [8]. The optimal subspace is given by the dominant eigenvectors of the covariance matrix $C = \int_\Omega \nabla f(x) \nabla f(x)^T \, dx$, generally estimated by Monte Carlo integration. Once the subspace is determined, the function can be approximated through a Gaussian process on the reduced space, i.e. we replace the original kernel $k(x, x')$ with a new kernel $\check{k}(x, x') = k(P^T x, P^T x')$. Because we assume gradient information, dimensionality reduction based on active subspaces is a natural pre-processing phase before applying D-SKI and D-SKIP.

## 3.3 Experiments

Our experiments use the squared exponential (SE) kernel, which has product structure and can be used with D-SKIP; and the spline kernel, to which D-SKIP

does not directly apply. We use these kernels in tandem with D-SKI and D-SKIP to achieve the fast MVMs derived in §3.2. We write D-SE to denote the exact SE kernel with derivatives.

### 3.3.1 Eigenspectrum approximation

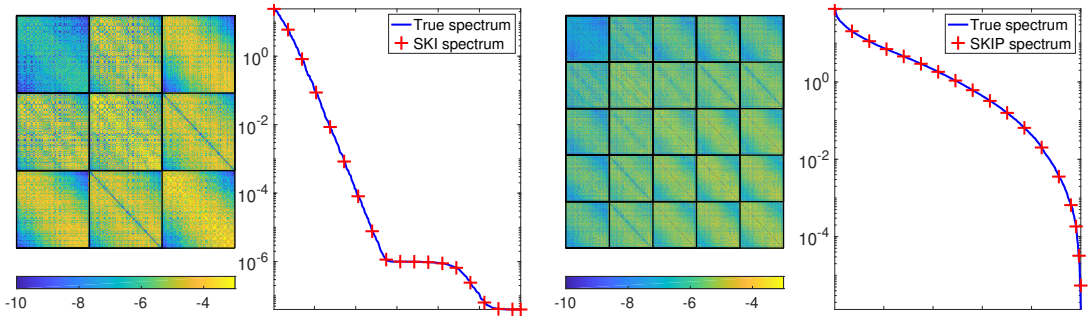



Figure 3.1: (Left two images) $\log_{10}$ error in D-SKI approximation and comparison to the exact spectrum. (Right two images) $\log_{10}$ error in D-SKIP approximation and comparison to the exact spectrum.

D-SKI and D-SKIP with the SE kernel approximate the original kernel well, both in terms of MVM accuracy and spectral profile. Comparing D-SKI and D-SKIP to their exact counterparts in Figure 3.1, we see their matrix entries are very close (leading to MVM accuracy near $10^{-5}$), and their spectral profiles are indistinguishable. The same is true with the spline kernel. Additionally, scaling tests in Figure 3.2 verify the predicted complexity of D-SKI and D-SKIP. We show the relative fitting accuracy of SE, SKI, D-SE, and D-SKI on some standard test functions in Table 3.1.

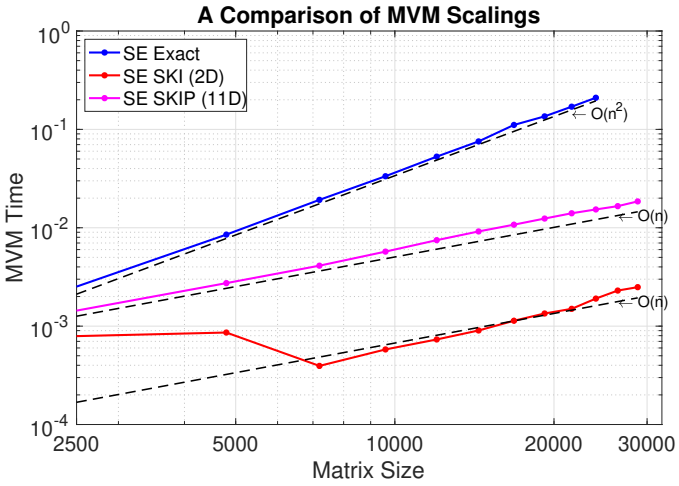



Figure 3.2: Scaling tests for D-SKI in two dimensions and D-SKIP in 11 dimensions. D-SKIP uses fewer data points for identical matrix sizes.

### 3.3.2 Kernel learning on test functions

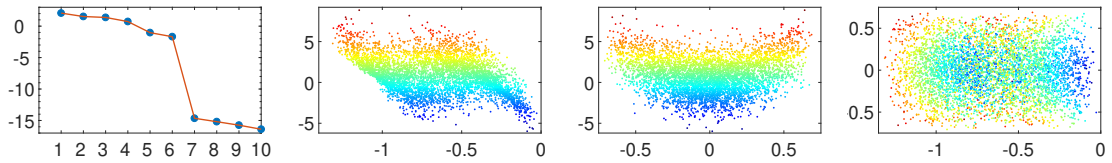
We consider several popular test functions in two and three dimensions to illustrate that derivative information leads to higher accuracy. The results are illustrated in Table 3.1.

| | Branin | Franke | Sine Norm | Sixhump | StyTang | Hartman3 |
|---|---|---|---|---|---|---|
| SE | 6.02e-3 | 8.73e-3 | 8.64e-3 | 6.44e-3 | 4.49e-3 | 1.30e-2 |
| SKI | 3.97e-3 | 5.51e-3 | 5.37e-3 | 5.11e-3 | 2.25e-3 | 8.59e-3 |
| D-SE | 1.83e-3 | 1.59e-3 | 3.33e-3 | 1.05e-3 | 1.00e-3 | 3.17e-3 |
| D-SKI | 1.03e-3 | 4.06e-4 | 1.32e-3 | 5.66e-4 | 5.22e-4 | 1.67e-3 |

Table 3.1: Relative RMSE error on 10000 testing points for test functions from [66], including five 2D functions (Branin, Franke, Sine Norm, Sixhump, and Styblinski-Tang) and the 3D Hartman function. We train the SE kernel on 4000 points, the D-SE kernel on $4000/(d+1)$ points, and SKI and D-SKI with SE kernel on 10000 points to achieve comparable runtimes between methods.

### 3.3.3 Dimensionality reduction

We apply active subspace pre-processing to the 20 dimensional Welsh test function in [4]. The top six eigenvalues of its gradient covariance matrix are well separated from the rest as seen in Figure 3.3(a). However, the function is far from smooth when projected onto the leading 1D or 2D active subspace, as Figure 3.3(b) - 3.3(d) indicates, where the color shows the function value.




(a) Log Directional Variation (b) First Active Direction (c) Second Active Direction (d) Leading 2D Active Subspace

Figure 3.3: 3.3(a) shows the top 10 eigenvalues of the gradient covariance. Welsh is projected onto the first and second active direction in 3.3(b) and 3.3(c). After joining them together, we see in 3.3(d) that points of different color are highly mixed, indicating a very spiky surface.

We therefore apply D-SKI and D-SKIP on the 3D and 6D active subspace, respectively, using 5000 training points, and compare the prediction error against D-SE with 190 training points because of our scaling advantage. Table 3.2 reveals that while the 3D active subspace fails to capture all the variation of the function, the 6D active subspace is able to do so. These properties are demonstrated by the poor prediction of D-SKI in 3D and the excellent prediction of D-SKIP in 6D.

| | D-SE | D-SKI (3D) | D-SKIP (6D) |
|---|---|---|---|
| RMSE | 4.900e-02 | 2.267e-01 | 3.366e-03 |
| SMAE | 4.624e-02 | 2.073e-01 | 2.590e-03 |

Table 3.2: Relative RMSE and SMAE prediction error for Welsh. The D-SE kernel is trained on $4000/(d+1)$ points, with D-SKI and D-SKIP trained on 5000 points. The 6D active subspace is sufficient to capture the variation of the test function.

### 3.3.4 Preconditioning

We discover that preconditioning is crucial for the convergence of iterative solvers using approximation schemes such as D-SKI and D-SKIP. To illustrate the performance of conjugate gradient (CG) method with and without the above-mentioned truncated pivoted Cholesky preconditioner, we test D-SKI on the 2D Franke function with 2000 data points, and D-SKIP on the 5D Friedman function with 1000 data points. In both cases, we compute a pivoted Cholesky decomposition truncated at rank 100 for preconditioning, and the number of steps it takes for CG/PCG to converge are demonstrated in Figure 3.4 below. It is clear that preconditioning universally and significantly reduces the number of steps required for convergence.

### 3.3.5 Rough terrain reconstruction

Rough terrain reconstruction is a key application in robotics [27, 42], autonomous navigation [31], and geostatistics. Through a set of terrrain measurements, the problem is to predict the underlying topography of some region. In the following experiment, we consider roughly 23 million nonuniformly sam-

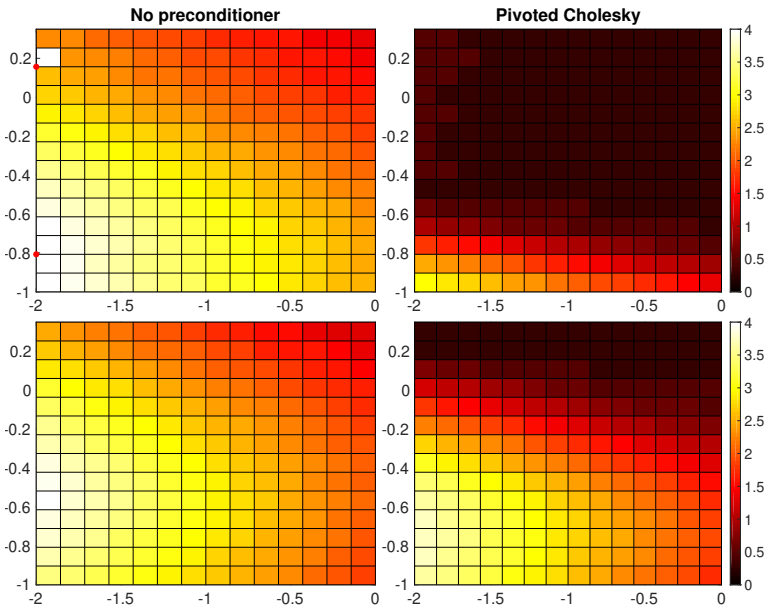



Figure 3.4: The color shows $\log_{10}$ of the number of iterations to reach a tolerance of 1e-4. The first row compares D-SKI with and without a preconditioner. The second row compares D-SKIP with and without a preconditioner. The red dots represent no convergence. The y-axis shows $\log_{10}(\ell)$ and the x-axis $\log_{10}(\sigma)$ and we used a fixed value of $s = 1$.

pled elevation measurements of Mount St. Helens obtained via LiDAR [7]. We bin the measurements into a $970 \times 950$ grid, and downsample to a $120 \times 117$ grid. Derivatives are approximated using a finite difference scheme.

We randomly select 90% of the grid for training and the remainder for testing. We do not include results for D-SE, as its kernel matrix has dimension roughly $4 \cdot 10^4$. We plot contour maps predicted by SKI and D-SKI in Figure 3.5 —the latter looks far closer to the ground truth than the former. This is quantified in the following table:

The Korean Peninsula elevation and bathymetry dataset[48] is sampled at a resolution of 12 cells per degree and has $180 \times 240$ entries on a rectangular grid. We take a smaller subgrid of $17 \times 23$ points as training data. To reduce data

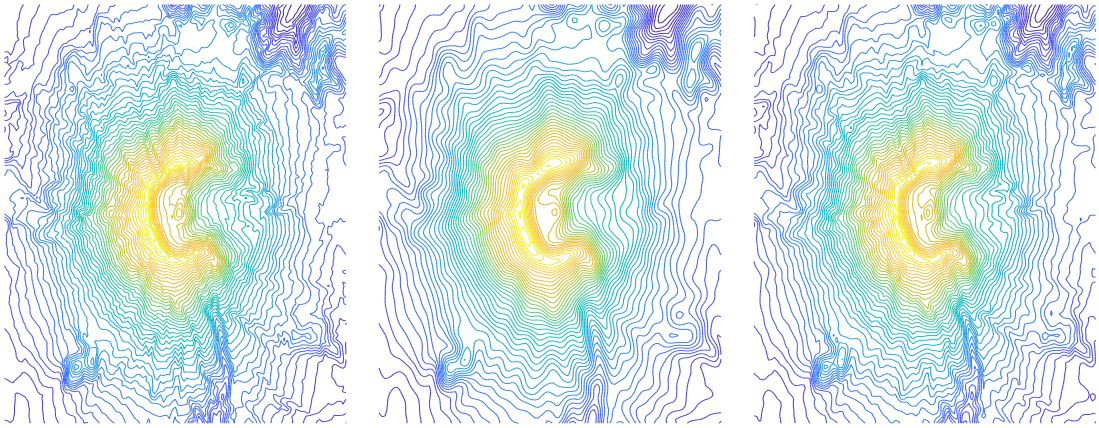


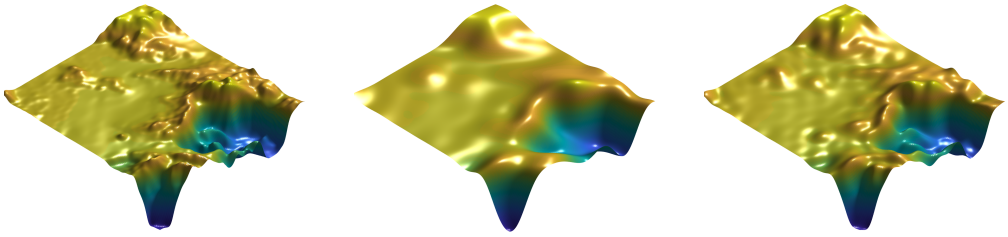
Figure 3.5: On the left is the true elevation map of Mount St. Helens. In the middle is the elevation map calculated with the SKI. On the right is the elevation map calculated with D-SKI.

| | $\ell$ | $s$ | $\sigma$ | $\sigma_2$ | Testing SMAE | Overall SMAE | Time[s] |
|---|---|---|---|---|---|---|---|
| SKI | 35.196 | 207.689 | 12.865 | n.a. | 0.0308 | 0.0357 | 37.67 |
| D-SKI | 12.630 | 317.825 | 6.446 | 2.799 | 0.0165 | 0.0254 | 131.70 |

Table 3.3: The hyperparameters of SKI and D-SKI are listed. Note that there are two different noise parameters $\sigma_1$ and $\sigma_2$ in D-SKI, for the value and gradient respectively.

noise, we apply a Gaussian filter with $\sigma_{\text{filter}} = 2$ as a pre-processing step. We observe that the recovered surfaces with SKI and D-SKI highly resemble their respective counterparts with exact computation and that incorporating gradient information enables us to recover more terrain detail.

| | $\ell$ | $s$ | $\sigma$ | SMAE | Time[s] |
|---|---|---|---|---|---|
| SKI | 16.786 | 855.406 | 184.253 | 0.1521 | 10.094 |
| D-SKI | 9.181 | 719.376 | 29.486 | 0.0746 | 11.643 |



(a) Ground Truth (b) SKI (c) D-SKI

Figure 3.6: D-SKI is clearly able to capture more detail in the map than SKI. Note that inclusion of derivative information in this case leads to a negligible increase in calculation time.

### 3.3.6 Implicit surface reconstruction

Reconstructing surfaces from point cloud data and surface normals is a standard problem in computer vision and graphics. One popular approach is to fit an implicit function that is zero on the surface with gradients equal to the surface normal. Local Hermite RBF interpolation has been considered in prior work [45], but this approach is sensitive to noise. In our experiments, using a GP instead of splining reproduces implicit surfaces with very high accuracy. In this case, a GP with derivative information is required, as the function values are all zero.

In Figure 3.7, we fit the Stanford bunny using 25000 points and associated normals, leading to a $K_{XX}^{\nabla}$ matrix of dimension $10^5$, clearly far too large for exact training. We therefore use SKI with the thin-plate spline kernel, with a total of 30 grid points in each dimension. The left image is a ground truth mesh of the underlying point cloud and normals. The middle image shows the same mesh, but with heavily noised points and normals. Using this noisy data, we fit a GP and reconstruct a surface shown in the right image, which looks very close to the original.

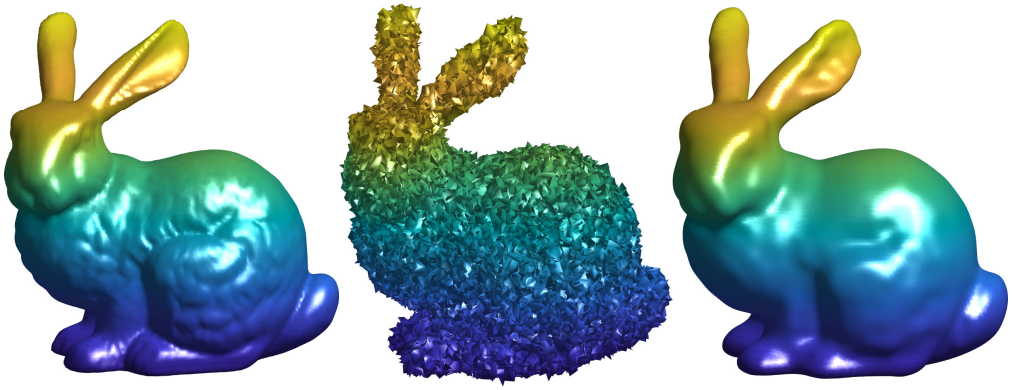


Figure 3.7: (Left) Original surface (Middle) Noisy surface (Right) SKI reconstruction from noisy surface ($s = 0.4, \sigma = 0.12$)

### 3.3.7 Bayesian optimization with derivatives

Prior work examines Bayesian optimization (BO) with derivative information in low-dimensional spaces to optimize model hyperparameters [81]. Wang et al. consider high-dimensional BO (without gradients) with random projections uncovering low-dimensional structure [70]. We propose BO with derivatives and dimensionality reduction via active subspaces, detailed in Algorithm 1.

---

Algorithm 3: BO with derivatives and active subspace learning

---

1: **while** Budget not exhausted **do**

2: Calculate active subspace projection $P \in \mathbb{R}^{D \times d}$ using sampled gradients

3: Optimize acquisition function, $u_{n+1} = \arg\max \mathcal{A}(u)$ with $x_{n+1} = Pu_{n+1}$

4: Sample point $x_{n+1}$, value $f_{n+1}$, and gradient $\nabla f_{n+1}$

5: Update data $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{x_{n+1}, f_{n+1}, \nabla f_{n+1}\}$

6: Update hyperparameters of GP with gradient defined by kernel $k(P^T x, P^T x')$

7: **end**

---

Algorithm 1 estimates the active subspace and fits a GP with derivatives in the reduced space. Kernel learning, fitting, and optimization of the acquisition function all occur in this low-dimensional subspace. In our tests, we use the expected improvement (EI) acquisition function, which involves both the mean and predictive variance. We consider two approaches to rapidly evaluate the predictive variance $v(x) = k(x,x) - K_{xX}\tilde{K}_{XX}^{-1}K_{Xx}$ at a test point $x$. In the first approach, which provides a biased estimate of the predictive variance, we replace $\tilde{K}_{XX}^{-1}$ with the preconditioner solve computed by pivoted Cholesky; using the stable QR-based evaluation algorithm, we have

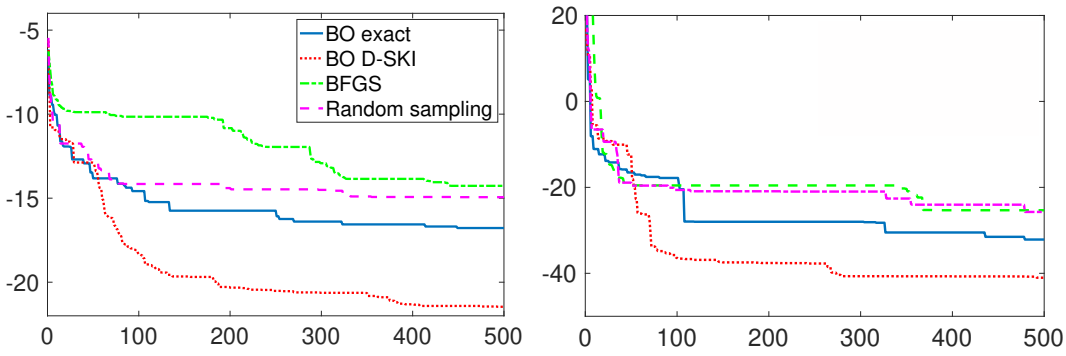
$$v(x) \approx \hat{v}(x) \equiv k(x,x) - \sigma^{-2}(\|K_{Xx}\|^2 - \|Q_1^T K_{Xx}\|^2).$$

In the second approach, we use a randomized estimator as in [3] to compute the predictive variance at many points $X'$ simultaneously, and use the pivoted Cholesky approximation as a control variate to reduce the estimator variance:

$$v_{X'} = \mathrm{diag}(K_{X'X'}) - \mathbb{E}_z\left[z \odot (K_{X'X}\tilde{K}_{XX}^{-1}K_{XX'}z - K_{X'X}M^{-1}K_{XX'}z)\right] - \hat{v}_{X'}.$$

The latter approach is unbiased, but gives very noisy estimates unless many probe vectors $z$ are used. Both the pivoted Cholesky approximation to the predictive variance and the randomized estimator resulted in similar optimizer performance in our experiments.

To test this algorithm, we consider five instances of the 5D Ackley and 5D Rastrigin functions randomly embedded in $[-10, 15]^{50}$ and $[-4, 5]^{50}$, respectively. In Figure 3.8(a) and Figure 3.8(b), we show the performance of our algorithm using the D-SKI kernel and the EI acquisition function. We fix $d = 2$, and at each iteration we pick two directions in the estimated active subspace at random. We also compare to three other methods: BO with EI and no gradients in the original space; multi-start BFGS with full gradients; and random search. In both




(a) BO on Ackley (b) BO on Rastrigin

Figure 3.8: In the following experiments, 5D Ackley and 5D Rastrigin are embedded into 50 a dimensional space. We run Algorithm 1, comparing it with BO exact, multi-start BFGS, and random sampling. D-SKI with active subspace learning clearly outperforms the other methods.

examples, the BO variants perform better than the alternatives, and our method outperforms standard BO.

## 3.4 Conclusion

The work in this chapter extended the work from Chapter 2 to the case where we observe both values and gradients. Gradients are a valuable additional source of information for GP regression, but inclusion of $d$ extra pieces of information per point naturally leads to new scaling issues. We introduced two methods to deal with these scaling issues: D-SKI and D-SKIP. Both are structured interpolation methods, and the latter also uses kernel product structure. We have discussed practical details — preconditioning is necessary to guarantee convergence of iterative methods and active subspace calculation reveals low-dimensional structure when gradients are available. We presented several experiments with ker-

nel learning, dimensionality reduction, terrain reconstruction, implicit surface fitting, and scalable Bayesian optimization with gradients. For simplicity, these examples all possessed full gradient information; however, our methods trivially extend if only partial gradient information is available. Future work and potential extensions are discussed in Chapter 6.

CHAPTER 4

# GLOBAL OPTIMIZATION IN RBF NATIVE SPACES

In this chapter we consider the global optimization problem of minimizing $f : \Omega \to \mathbb{R}$ where $\Omega$ is compact. Given only continuity of $f$, an optimization method will converge to a global minimum if and only if it eventually samples densely in $\Omega$ [67]. Given further information about the function, such as a Lipschitz constant, one can find a global minimum more efficiently. While prior surrogate-based optimization methods achieve global convergence by dense sampling, standard approximation theory results for these surrogates often require more structure than simple continuity. In particular, RBF approximations converge only when $f$ belongs to a certain reproducing kernel Hilbert space (the *native space* for the RBF). In this chapter, we introduce a new global optimization method based on this approximation theory. Given only an upper bound on the native space norm of $f$, we use RBF approximation theory to develop a new algorithm that converges globally without dense sampling.

## 4.1 Background

The inequality in (1.8) provides a lower bound for $f(x)$ anywhere in the domain:

$$f(x) \geq \ell_{f,X}(x) := s_{f,X}(x) - P_{X,\varphi}(x)\sqrt{|f|^2_{\mathcal{N}_\varphi} - |s_X|^2_{\mathcal{N}_\varphi}}, \tag{4.1}$$

and this lower bound only depends on $|f|_{\mathcal{N}_\varphi}$ and the set $X$. In this section, we develop a greedy global optimization algorithm based on minimizing this lower bound at each step. For this to be feasible we assume that $f \in \mathcal{N}_\varphi$ and that we know $|f|_{\mathcal{N}_\varphi}$ or an upper bound $\rho \geq |f|_{\mathcal{N}_\varphi}$. The assumption $f \in \mathcal{N}_\varphi$ is not strong

when we are working with e.g., polyharmonic kernels since the native spaces are instances of Beppo-Levi spaces as was discussed on page 6. In particular, we make the following assumptions:

1. $f \in \mathcal{N}_\varphi$.
2. $|f|_{\mathcal{N}_\varphi}$ or an upper bound $\rho$ is known.
3. $X$ is $(\nu - 1)$-unisolvent (if $\varphi$ is conditionally positive definite of order $\nu$).
4. We can find the global minimum of $\ell_{f,X_n}$ in $\Omega$.

### 4.1.1 The algorithm

The ideas from the previous section lead to an optimization algorithm that selects the point that minimizes the lower bound. Global minimization of the lower bounds may itself be challenging and have several local minima. It is possible to use a heuristic method such as a genetic algorithm or a particle swarm, but since both the gradient and the Hessian of the lower bounds are easy to obtain we can use a multi-start gradient method. The optimization algorithm is given in Algorithm 4.

This algorithm will stop the optimization process when we have found a function value within $\epsilon$ of the global minimum value. This is achieved by using the fact that the global minimum value $f^*$ satisfies

$$\min_{y\in\Omega} \ell_{f,X_n}(y) \leq f^* \leq \min f_{X_n}, \qquad \forall n \geq n_0,$$

from which it is obvious that

$$\left|\min f_{X_n} - f^*\right| \leq \left|\min f_{X_n} - \min_{y\in\Omega} \ell_{f,X_n}(y)\right|.$$

Algorithm 4: Optimization algorithm based on a semi-norm budget

Tolerance $\epsilon$

$X_{n_0} \leftarrow \{x_1, x_2, \ldots, x_{n_0}\}$ a unisolvent set of initial points

$f_{X_{n_0}} \leftarrow \{f(x_1), f(x_2), \ldots, f(x_{n_0})\}$

Build $s_{f,X_{n_0}}$ from $(X_{n_0}, f_{X_{n_0}})$

$n \leftarrow n_0$

**while** $\left| \min f_{X_n} - \min_{x \in \Omega} \ell_{f,X_n}(x) \right| > \epsilon$ **do**

    $y \leftarrow \arg\min_{x \in \Omega} \ell_{f,X_n}(x)$

    $X_{n+1} \leftarrow X_n \cup \{y\}$

    $f_{X_{n+1}} \leftarrow f_{X_n} \cup \{f(y)\}$

    Build $s_{f,X_{n+1}}$ from $(X_{n+1}, f_{X_{n+1}})$

    $n \leftarrow n + 1$

We also have the property that the sequence of lower bounds is non-decreasing; that is, for each iteration $n$ where $s_{f,X_n}(y) \neq f(y)$ we have that $\ell_{f,X_n}(x) < \ell_{f,X_{n+1}}(x)$ for all $x \notin X_n$. This property is what we will use to prove that this algorithm is globally convergent without necessarily sampling densely in $\Omega$ as long as the assumptions are satisfied. In particular, if we are given an upper bound $\rho \geq |f|_{\mathcal{N}_\varphi}$, then Algorithm 4 will do more exploration and less exploitation since the lower bounds are weaker. On the other hand, if $\rho < |f|_{\mathcal{N}_\varphi}$, then global convergence cannot be guaranteed.

## 4.2 Convergence of the method

In this section we prove that Algorithm 4 is globally convergent even though it may not sample densely. We first show the result for SPD kernels in §4.2.1, then explain how to extend the results to CPD kernels in §4.2.2.

### 4.2.1 Global convergence for SPD kernels

Suppose $k(x, y) = \varphi(r)$ is a radial SPD kernel and that there is some continuous monotonically increasing function $\psi : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ with $\psi(0) = 0$ such that $|k(x, y) - k(x, x)| \leq \psi(\delta)$ for any $\|x - y\| \leq \delta$. Note that this is a weak assumption and that it is easy to construct such a function for the kernels introduced in Chapter 1. We start by constructing an upper bound on the power function (1.7) by bounding it by the power function based only on the closest point.

**Lemma 2.** *Let $X$ be any set of interpolation nodes. Then for any $y$,*

$$P_{X,\varphi}(y)^2 \leq 2\psi\left(\min_{x \in X} \|y - x\|\right).$$

*Proof.* Let $x$ be the closest point to $y$ in $X$. Then

$$P_{X,\varphi}(y)^2 \leq P_{x,\varphi}(y)^2 = k(y, y) - \frac{k(x, y)^2}{k(x, x)}$$

because the power function decreases with each new interpolation point. It follows that

$$\begin{aligned} k(y, y) - \frac{k(x, y)^2}{k(x, x)} &= k(y, y) - 2k(x, y) + k(x, x) - \frac{(k(x, x) - k(x, y))^2}{k(x, x)} \\ &\leq |k(x, x) - k(x, y)| + |k(y, y) - k(x, y)| \\ &\leq 2\psi\left(\min_{x \in X} \|y - x\|\right). \end{aligned}$$

□

The following lemma is a trivial consequence of the fact that the sequence of lower bounds is non-decreasing.

**Lemma 3.** *For any $j > 0$,*

$$\ell_{f,X_n}(x_{n+j}) \leq f^*.$$

*Proof.* By construction, $\ell_{f,X_{n+j-1}}(x_{n+j}) \leq f^*$; and the lower bounds increase with each additional point, so $\ell_{f,X_n}(x_{n+j}) \leq \ell_{f,X_{n+j-1}}(x_{n+j})$. □

Next, we combine the previous two lemmas with the inequality in (1.8) to bound how close a function value is from the global minimum value $f^*$.

**Lemma 4.** *Suppose $\|x_n - x_{n+j}\| \leq \delta$. Then*

$$|f(x_{n+j}) - f^*| \leq \sqrt{8\psi(\delta)}\,|f|_{\mathcal{N}_\varphi}.$$

*Proof.* Let $\gamma = \sqrt{|f|^2_{\mathcal{N}_\varphi} - |s_{f,X_n}|^2_{\mathcal{N}_\varphi}} \leq |f|_{\mathcal{N}_\varphi}$. We first observe that

$$\begin{aligned}
|f(x_{n+j}) - \ell_{f,X_n}(x_{n+j})| &= \left|f(x_{n+j}) - s_{f,X_n}(x_{n+j}) + \gamma P_{X_n,\varphi}(x_{n+j})\right| \\
&\leq \left|f(x_{n+j}) - s_{f,X_n}(x_{n+j})\right| + \gamma P_{X_n,\varphi}(x_{n+j}) \\
&\leq 2\gamma P_{X_n,\varphi}(x_{n+j}) \\
&\leq 2|f|_{\mathcal{N}_\varphi} P_{X_n,\varphi}(x_{n+j}).
\end{aligned}$$

By Lemma 2 and Lemma 3,

$$f(x_{n+j}) - f^* \leq 2P_{X_n,\varphi}(x_{n+j})|f|_{\mathcal{N}_\varphi} \leq 2\sqrt{2\psi(\delta)}|f|_{\mathcal{N}_\varphi}.$$

□

We are now ready to prove that Algorithm 4 on page 68 is globally convergent. The main idea of the proof is to use Bolzano-Weierstrass to pick a convergent subsequence and show that the corresponding sequence of function values is arbitrarily close to $f^*$. The proof is valid if there are multiple global minimizers of $f$ since we can always pick one convergent subsequence.

**Theorem 5.** *Let $\Omega \subset \mathbb{R}^d$ be compact, and suppose $\rho \geq |f|_{\mathcal{N}_\varphi}$. Then Algorithm 4 is globally convergent; that is,* $\min f_{X_n} \to f^*$.

*Proof.* Let $(x_n)$ be the sequence selected by Algorithm 4. The Bolzano-Weierstrass theorem guarantees a convergent subsequence $(x_{n_i})$ such that $x_{n_i} \to \hat{x} \in \Omega$, and continuity of $f$ guarantees that $f(x_{n_i}) \to f(\hat{x})$. For any $\epsilon > 0$, continuity of $\psi$ (together with $\psi(0) = 0$) implies $\exists \delta > 0$ such that $\sqrt{8\psi(\delta)}|f|_{\mathcal{N}_\varphi} < \epsilon$. For any convergent subsequence $(x_{n_i})$, there exists an $i_0$ such that for all $j > 0$, $\|x_{n_{i_0+j}} - x_{i_0}\| \leq \delta$; by the previous lemma, this implies

$$0 \leq f(x_{i_0+j}) - f^* \leq \epsilon.$$

We are therefore guaranteed $f(\hat{x}) = f^*$. □

### 4.2.2 Global convergence for CPD kernels

Now consider the case where the kernel $k(x, y) = \varphi(r)$ is a radial CPD kernel relative to some polynomial space with basis functions $\pi_1, \pi_2, \ldots \pi_m$. In this setting, we need a modification to Lemma 2. We assume a function $\psi$ that controls both the kernel (as before) and the polynomial basis functions — that is, we also require $|\pi_j(x) - \pi_j(y)| \leq \psi(\|x - y\|)$, $j = 1, \ldots, m$.

**Lemma 6.** *Let $X$ be any set of interpolation nodes consisting of a fixed subset $X_0$ which is unisolvent with respect to interpolation in $\Pi^d_{\nu-1}$ and a subset $X_1$ containing the remaining nodes. Then there exists a constant $C$ that does not depend on $X$, such that for any $y$,*

$$P_{X,\varphi}(y)^2 \leq 2C\psi\left(\min_{x\in X_1}\|y-x\|\right).$$

*Proof.* For any points $u, v$ not in $X_0$, we can define the modified SPD kernel function

$$\hat{k}(u,v) = k(u,v) - w(u)^T A_0^{-1} w(v)$$

where

$$A_0 = \begin{bmatrix} K_{X_0X_0} & P_{X_0} \\ P_{X_0}^T & 0 \end{bmatrix}, \qquad w(u) = \begin{bmatrix} K_{X_0u} \\ P_u^T \end{bmatrix}.$$

By compactness of $\Omega$ and continuity of the kernel and polynomial basis functions, there exists some $C > 1$ such that $\|A_0^{-1}w(u)\|_1 \leq C - 1$ for all $u \in \Omega$. Note that

$$\begin{aligned} |\hat{k}(u,v) - \hat{k}(u,u)| &\leq |k(u,v) - k(u,u)| + |w(u)^T A_0^{-1}(w(v) - w(u)| \\ &\leq \psi(\|v-u\|) + (C-1)\|w(v) - w(u)\|_\infty \\ &\leq \psi(\|v-u\|) + (C-1)\psi(\|v-u\|) \\ &= C\psi(\|v-u\|), \end{aligned}$$

The modified kernel is SPD on $\Omega \setminus X_0$, and the power function $P_{X,\varphi}(y)$ for the original kernel is the same as the power function $\hat{P}_{X_1,\varphi}(y)$ for the modified kernel. Let $x$ be the point in $X_1$ that is nearest to $y$. Then

$$P_{X,\varphi}(y)^2 \leq P_{X_0\cup\{x\},\varphi}(y)^2 = \hat{P}_{x,\varphi}(y) \leq 2C\psi(\|x-y\|),$$

by applying Lemma 2. □

The proof of Theorem 5 for a CPD kernel parallels what we saw in the SPD case, except that we must work with the scaled function $\hat{\psi}(\delta) = C\psi(\delta)$.

### 4.2.3 Convergence rates

The standard convergence rate theory for RBF interpolation was summarized in §1.3.2. However, these results do not apply in our setting as convergence is measured in terms of the fill distance. Algorithm 4 may not sample densely which will prevent the fill-distance from approaching zero. We are interested in how fast $\min f_{X_n}$ approaches $f(x^*)$, and if we use the fact that

$$\left|f(x^*) - \min f_{X_n}\right| \leq |f(x^*) - \ell_{f,X_n}(x^*)|,$$

it is enough to find a rate for how fast $\ell_{f,X_n}(x^*)$ approaches $f(x^*)$. Mimicking the main steps in Lemma 4,

$$\begin{aligned} |f(x^*) - \ell_{f,X_n}(x^*)| &= \left|f(x^*) - s_{f,X_n}(x^*) + \gamma P_{X_n,\varphi}(x^*)\right| \\ &\leq \left|f(x^*) - s_{f,X_n}(x^*)\right| + \gamma P_{X_n,\varphi}(x^*) \\ &\leq 2\gamma P_{X_n,\varphi}(x^*) \end{aligned}$$

we need to find the rate of convergence for the power function evaluated at $x^*$.

The easiest way to achieve a convergence rate that relates to the smoothness of the kernel is to modify the algorithm so a point is selected uniformly from the domain with probability $\epsilon$, and the minimizer of the lower bound is selected with probability $1 - \epsilon$. This approach is used by Bull [6] to prove convergence rates for expected improvement (EI), but the results are miss-leading as the convergence rates comes from the quasi-uniformity of the points selected uniformly at random and does not depend on how the other points are selected.

## 4.3 Experiments

This section will test Algorithm 4 on a set of popular test problems. We will verify that the sampling pattern is promising and compare estimated convergence rates with convergence rates based on the fill distance.

### 4.3.1 Implementation details

We use MATLAB R2018b for the numerical experiments and rely on *fmincon* to minimize the lower bounds. In particular, we use the SQP method since it works with bound constraints and can make use of gradient information for the lower bounds. We use SQP in a multi-start fashion by starting from a perturbation of each previously evaluated point. This approach was compared to using a genetic algorithm and the multi-start gradient approach consistently did better. The true semi-norm was approximated based on a regular grid consisting of 10,000 points.

### 4.3.2 Sampling pattern and feasible region

This experiment illustrates the sampling pattern of Algorithm 4, using a cubic kernel and a linear tail, on the two-dimensional six-hump camel function, which has 6 stationary points. The true semi-norm was estimated based on a $100 \times 100$ grid and we used an SLHD with 6 points as the experimental design and a total of 250 function evaluations. Figure 4.1 shows the sampling pattern and we see that the algorithm was eventually only sampling close to the two global minima.

It is also clear that regions with large function values were barely explored.

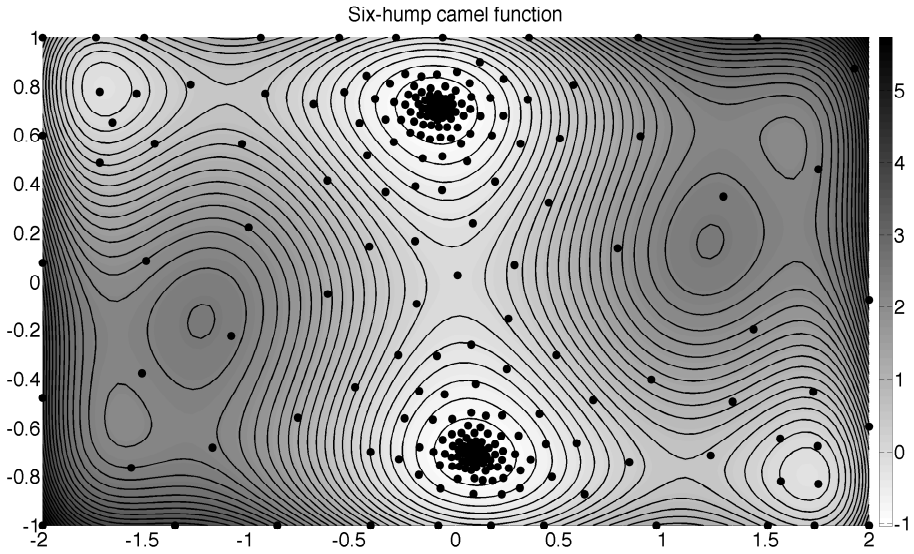



Figure 4.1: The sampling pattern of Algorithm 4 on the six-hump camel function. We used a cubic kernel and a linear tail and a total of 250 evaluations.

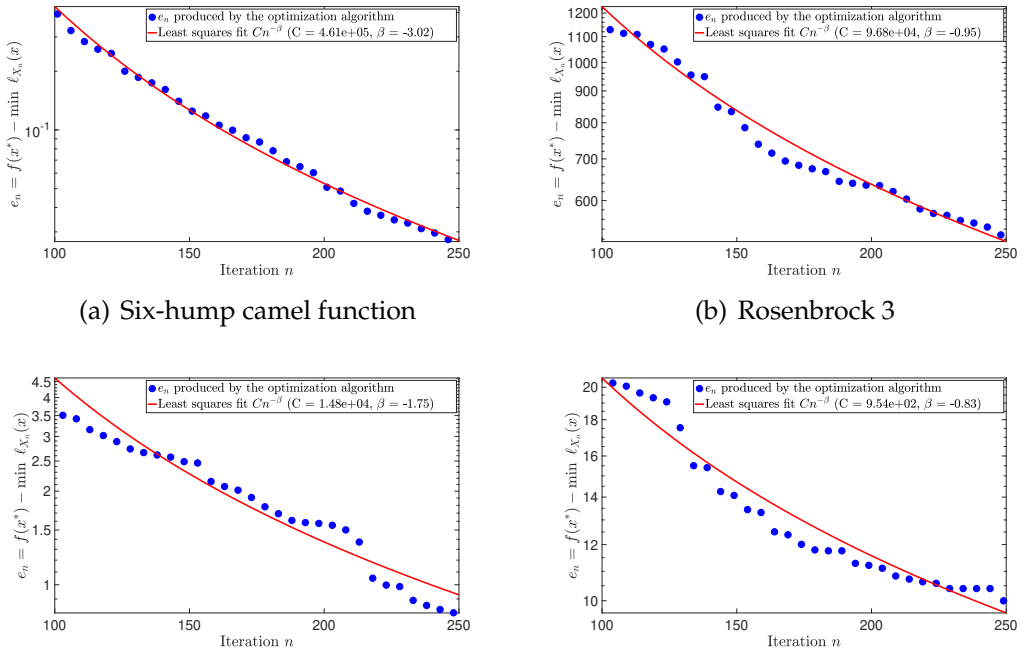
### 4.3.3 Estimated convergence rates

We focus on the cubic kernel which converges at a polynomial rate according to Table 1.1. We consider four different test functions and estimate the true semi-norm in a similar way as in the previous experiments, which makes this experiment limited to low-dimensional functions. We look for a convergence rate of the form

$$e_n = \left| f(x^*) - \min_{x \in \Omega} \ell_{f,X_n}(x) \right| = Cn^{-\beta}.$$

If the points are quasi-uniformly distributed we expect $\beta \approx 2/d$ since $h_{X_n,\Omega} = O(n^{-1/d})$ and the cubic kernel is of order 2. Algorithm 4 may not sample densely, so the convergence rate results from §1.3.2 do not apply, but gives a baseline of

comparison. The constants $C$ and $\beta$ are found by finding the optimal degree-1 polynomial in the least-squares sense that fits the data points $\{\log(n), \log(e_n)\}_{n=1}^{N}$. We ignore the first 100 evaluations since they are not part of the asymptotic region and has too much influence on the resulting fit. We use a total of 250 function evaluations and the experimental design is chosen to be a SLHD with $2(d+1)$ points. The resulting convergence plots can be seen in Figure 4.2.




(a) Six-hump camel function

(b) Rosenbrock 3

(c) Hartman 3

(d) Hartman 6

Figure 4.2: Estimated convergence rates for different functions using Algorithm 4 with a cubic kernel and a linear tail. (Blue dots) Values of $e_n$. (Red line) Least squares fit of the form $Cn^{-\beta}$

Figure 4.2(a) shows the six-hump camel function which is a two-dimensional function with 6 stationary points. The lower bound was within 0.0248 of the global optimum value after 250 evaluations and the exponent $\beta$ was estimated to be 3.02, which is much larger than the expected value of 1 for a quasi-uniform point distribution. The best function value found is within $4.27e-7$ of the global minimum value and the $R^2$ of the linear fit is 0.994.

Figure 4.2(b) shows the 3-dimensional Rosenbrock function, a common test function in optimization. The narrow valley from the local minimum $(0, 0, 0)$ to the global minimum $(1, 1, 1)$ makes it challenging for many optimization algorithms. We notice that the lower bounds are 516.00 from the global minimum value, mainly since the sharp valleys result in a large value of $|f|_{\mathcal{N}_\varphi}$. The best solution found is within $8.1e-3$ of the global minimum value and the estimated exponent is $\beta = 0.95$, which is slightly larger than the expected convergence rate of $2/3$ for a quasi-uniform point distribution. The $R^2$ of the linear fit is 0.939.

Figure 4.2(c) shows the 3-dimensional Hartman3 function, which has 4 local minima. The lower bound is within 0.81 of the global optimum value after 250 evaluations and the estimated value $\beta = 1.75$ is much better than the expected value $2/3$ in the quasi-uniform setting. The function value of the best solution found is within $1.7e-7$ of the global minimum value. The $R^2$ of the linear fit is 0.939.

Figure 4.2(d) shows the 6-dimensional Hartman6 function, which has 6 local minima. Estimating the semi-norm is challenging in six dimensions and we used a regular grid with $5^6$ points. For this reason, the estimated semi-norm may be much smaller than the true semi-norm. The lower bound is within 10.06 of the global minimum value and the estimated value of $\beta = 0.83$ is larger than the expected value of $1/3$ for a quasi-uniform point distribution. The function value of the best solution found is within 0.19 of the global minimum value and the $R^2$ of the linear fit is 0.935.

## 4.4 Conclusion

We have introduced a new RBF-based global optimization algorithm for expensive black-box functions that uses native space semi-norm bounds to pick the next evaluation. We have proved that the algorithm is globally convergent for functions in the native space of the given RBF kernel given an upper bound on the semi-norm. Our numerical experiments demonstrate that Algorithm 4 converges at a rate that is better than what we expect from RBF interpolation theory, and that the sampling pattern on the six-hump camel function looks promising. Future work includes estimating the semi-norm from a global regularity condition, such as Lipschitz as well as extending the algorithm to work with functions outside the native space of the kernel. We also want to prove convergence results that agree with the RBF interpolation theory in §1.3.2 without modifying the algorithm to select additional points uniformly at random.

CHAPTER 5

# PYSOT AND POAP: ASYNCHRONOUS GLOBAL OPTIMIZATION

This chapter describes Plumbing for Optimization with Asynchronous Parallelism (`POAP`) and the Python Surrogate Optimization Toolbox (`pySOT`). `POAP` is an event-driven framework for building and combining asynchronous optimization strategies, designed for global optimization of expensive functions where concurrent function evaluations are appealing. `pySOT` is a collection of synchronous and asynchronous surrogate optimization strategies, implemented in the `POAP` framework. We support the stochastic RBF method by Regis and Shoemaker [58] along with various extensions of this method, as well as several Bayesian optimization methods. We have implemented many different surrogate models, experimental designs, and a large set of test problems.

We have tested that our implementation is consistent with previously reported results and make an extensive comparison between synchronous and asynchronous parallel. We find that asynchrony is never worse than synchrony on several challenging multimodal test problems and conclude that launching evaluations asynchronously is consistently better when increasing the variance in the evaluation time or the number of processors.

## 5.1 Background

Several parallel algorithms have been developed for computationally expensive black-box optimization. Regis and Shoemaker [59] developed a synchronous parallel surrogate optimization algorithm based on radial basis func-

tions (RBFs). They made two assumptions: (i) that the resources are homogeneous and (ii) that the evaluation time is constant. The first assumption does not hold for heterogeneous parallel computing platforms and the second assumption is unlikely to hold in cases where the complexity of evaluating the objective depends spatially on the input. The first assumption can almost always be assessed before the start of the optimization run while the second assumption may not be easy to assess in practice. Another limitation of the work in [59] is that the algorithm does not handle the possibility of worker failures and crashed evaluations. Being able to handle failures is critical in order to run the algorithm on large-scale systems. The natural way of dealing with cases where (i) or (ii) are violated is to launch function evaluations asynchronously, which is illustrated in Figure 5.1.

### 5.1.1 Survey of Software

A library with similar functionality as `POAP` is SCOOP [37], a Python based library for distributing concurrent tasks while internally handling the communication. `POAP` provides similar functionality for global optimization problems and also handles all of the communication internally, which makes it easy to implement asynchronous optimization algorithms.

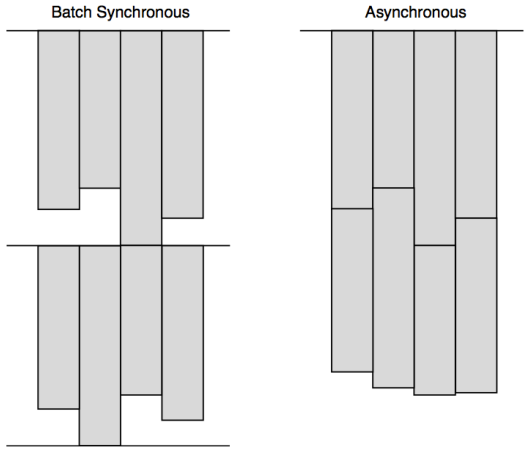



Figure 5.1: Synchronous vs asynchronous parallel

HOPSPACK (Hybrid Optimization Parallel Search PACKage) [52] is a C++ framework for derivative-free optimization problems. HOPSPACK supports parallelism through MPI or multi-threading and supports running multiple optimization solvers simultaneously, a functionality similar to combining strategies in `POAP`. The framework implements an asynchronous pattern search solver and supports non-linear constraints and mixed-integer variables, but there is no support for surrogate optimization.

MATSuMoTo (MATLAB Surrogate Model Toolbox) [50] is an example of a surrogate optimization toolbox. MATSuMoTo is written in MATLAB and has support for computationally expensive, black-box global optimization problems that may have continuous, mixed-integer, or pure integer variables. MATSuMoTo offers a variety of choices for surrogate models and surrogate model mixtures, experimental designs, and auxiliary functions. The framework is not designed to support a large class of surrogate optimization algorithms and the lack of object orientation makes it hard to extend the framework. Parallelism is only supported through MATLAB's Parallel Computing Toolbox and there is no support for asynchrony, combining strategies, or dynamically changing the number of workers.

Nonlinear Optimization by Mesh Adaptive Direct Search (NOMAD) [44] is a library intended for time-consuming black-box simulation with a small number of variables. The library implements mesh adaptive direct search (MADS) and there is support for asynchronous function evaluations using MPI. The framework is fault resilient in the sense that it supports objective function failing to return a valid output. Similar fault resilience is provided by `POAP`, which allows the user to decide what action to take in case of a failure.

Dakota [15] is an extensive toolkit with algorithms for optimization with gradient and nongradient-based methods; uncertainty quantification, nonlinear least squares methods, and sensitivity/variance analysis. These components can be used on their own or with strategies such as surrogate-based optimization, mixed integer nonlinear programming, or optimization under uncertainty. The Dakota toolkit is object-oriented and written in C++ with the intention of being a flexible and extensible interface between simulation codes and there is support for parallel function evaluations.

BayesOpt [47] is a library with Bayesian optimization methods to solve nonlinear optimization problems. Bayesian optimization methods build a posterior distribution to capture the evidence and prior knowledge of the target function. Built in C++, the library is efficient, portable, and flexible. There is support for commonly used methods such as sequential Kriging optimization (SKO), sequential model-based optimization (SMBO), and efficient global optimization (EGO). The software is sequential and there is no support for parallel function evaluations.

RBFOpt [9] is a radial basis function based library that implements and extends the global optimization algorithm proposed by Gutmann [29]. RBFOPT is written in Python and supports asynchronous parallelism through Python's multiprocessing library, but there is no support for MPI. The software is not designed to cover a large class of surrogate optimization methods and there is no support for dynamically changing the number of workers and combining different optimization strategies.

Cornell-MOE is a Python library that implements Bayesian optimization with the expected improvement and knowledge gradient acquisition functions.

The software is built on work that extends these acquisition functions to batch synchronous parallel, both with and without gradient information [80, 81]. There is no support for asynchronous parallelism and it is not possible to dynamically change the number of workers.

## 5.2 The asynchronous algorithm

A surrogate optimization in the flavor of Algorithm 1 on page 18 is easy to implement, but may be inefficient if the evaluation time is not constant. This can be because evaluating the simulation model requires larger computational efforts for some input values (e.g., evaluating a PDE-based objective function may require smaller step sizes for some values of the decision variable $x$). Computation time can also vary because of variation in the computational resources. Dealing with potential function evaluation crashes is less obvious in a synchronous framework, where we may either try to re-evaluate or exclude the points from the batch. Finally, dynamically changing the number of resources is much more straightforward in an asynchronous framework, which we describe next.

Just as in the synchronous parallel case we start by evaluating an experimental design. These points can be evaluated asynchronously, but the fact that we want to evaluate all design points before proceeding to the adaptive phase introduces an undesirable barrier. This becomes an issue if there are straggling workers or if some points take a long time to evaluate. The most natural solution is to generate an experimental design that makes it possible to let workers proceed to the adaptive phase once all points are either completed or pending. To be more precise, assume that we have $p$ workers and that $q$ points are needed

to construct a surrogate model. We can then generate an experimental design with $\geq p + q - 1$ points, which will allow workers to proceed to the adaptive phase once there are no outstanding evaluations in the experimental design. The adaptive phase differs from Algorithm 1 in the sense that we propose a new evaluation as soon as a worker becomes available.

We use an event-driven framework where the master drives the event loop, updates the surrogate, solves the auxiliary problem, etc., and we have $p$ workers available to do function evaluations. The workload of the master is significantly less than of the workers, so we can use the same number of workers as we have available resources. The asynchronous algorithm is illustrated in Algorithm 5.

### 5.2.1 Updating the sampling radius in Stochastic SRBF

We now elaborate on how to pick the value of the sampling radius $\gamma$ that is used to generate the candidate points used in the LMS-RBF and DYCORS methods. We follow the idea in [58] where counters $C_{\text{success}}$ and $C_{\text{fail}}$ are used to track the number of consecutive evaluations with and without significant improvement. This idea is extended to synchronous parallel in [59] by processing a batch at a time. If $C_{\text{success}}$ reaches a tolerance $\mathcal{F}_{\text{success}}$ the sampling radius is doubled and $C_{\text{success}}$ is set to 0. Similarly, if $C_{\text{fail}}$ reaches $\mathcal{F}_{\text{fail}}$ the sampling radius is halved and $C_{\text{fail}}$ is set to 0.

In the asynchronous setting, we update the counters after each completed function evaluation. We do not update the counters for evaluations that were launched before the last time the sampling radius was changed. The reason for this is that these evaluations are based on outdated information. The logic for

Algorithm 5: Asynchronous surrogate optimization algorithm

**Inputs:** Initial design points, $p$ workers
$X \leftarrow \emptyset$, $f_X \leftarrow \emptyset$
Queue initial design

**for** each worker **do** ▷ Launch initial evaluations
    Pop point from queue and dispatch to worker

**while** stopping criterion not met (e.g., $t < t_{\max}$) **do** ▷ Event loop
    **if** evaluation completed **then**
        Update $X$ and $f_X$
    **else if** evaluation failed and retry desired **then**
        Add back to evaluation queue

    **if** worker ready **then**
        **if** evaluation queue empty **then**
            Build surrogate model
            Solve auxiliary problem and add point to the queue
        Pop point from queue and dispatch to worker
**return** $x_{\text{best}}$, $f_{\text{best}}$

Algorithm 6: Sampling radius adjustment routine

**Inputs:** $\gamma$, $f(x_i)$, $x_i$, $f_{\text{best}}$, $x_{\text{best}}$, $C_{\text{success}}$, $C_{\text{fail}}$, $\mathcal{F}_{\text{success}}$, $\mathcal{F}_{\text{fail}}$, $\delta$
**if** $f(x_i) < f_{\text{best}}$ **then**
    $f_{\text{best}} \leftarrow f(x_i)$
    $x_{\text{best}} \leftarrow x_i$
    **if** $f(x_i) < f_{\text{best}} - \delta|f_{\text{best}}|$ **then**
        $C_{\text{succ}} \leftarrow C_{\text{succ}} + 1$
        $C_{\text{fail}} \leftarrow 0$
**else**
    $C_{\text{succ}} \leftarrow 0$
    $C_{\text{fail}} \leftarrow C_{\text{fail}} + 1$

**if** $C_{\text{succ}} = \mathcal{F}_{\text{succ}}$ or $C_{\text{fail}} = \mathcal{F}_{\text{fail}}$ **then**
    $C_{\text{succ}} \leftarrow 0$
    $C_{\text{fail}} \leftarrow 0$
    **if** $C_{\text{succ}} = \mathcal{F}_{\text{succ}}$ **then**
        $\gamma \leftarrow \min(2\gamma, \gamma_{\text{max}})$
    **else**
        $\gamma \leftarrow \max(\gamma/2, \gamma_{\text{min}})$

**return** $\gamma$, $f_{\text{best}}$, $x_{\text{best}}$ $C_{\text{success}}$, $C_{\text{fail}}$

updating the sampling radius and the best solution can be seen in Algorithm 6.

We also follow the recommendations in [58] and [59] to restart the algorithm when we reach a maximum failure tolerance parameter $\mathcal{M}_{\text{fail}}$ or when the sam-

pling radius $\gamma$ drops below $\gamma_{\min}$. Restarting has shown to be successful for LMS-RBF and DYCORS as it can be hard to make progress when the surrogate is very biased towards the current best solution and we may be stuck in a local minimum that is hard to escape. Restarting the algorithm can help avoid this issue. We do not terminate pending evaluations after a restart occurs, but they are not incorporated in the surrogate model or used to adjust the sampling radius when they finish.

## 5.3 POAP implementation

This section describes the Plumbing for Optimization with Asynchronous Parallelism[1] (`POAP`) framework. `POAP` has three main components: a controller that asks workers to run function evaluations, a strategy that proposes new actions, and a set of workers that carry out function evaluations.

### 5.3.1 Controllers

The controller is responsible for accepting or rejecting proposals by the strategy object, controlling and monitoring the workers, and informing the strategy object of relevant events. Examples of relevant events are the processing of a proposal, or status updates on a function evaluation. Interactions between controller and the strategies are organized around proposals and evaluation records. At the beginning of the optimization and on any later change to the system state, the controller requests a proposal from the strategy. The proposal

[1] `POAP` can be downloaded from: https://github.com/dbindel/POAP

consists of an action (evaluate a function, kill a function, or terminate the optimization), a list of parameters, and a list of callback functions to be executed once the proposal is processed. The controller then either accepts the proposal (and sends a command to the worker), or rejects the proposal.

When the controller accepts a proposal to start a function evaluation, it creates an evaluation record to share information about the status of the evaluation with the strategy. The evaluation record includes the evaluation point, the status of the evaluation, the value (if completed), and a list of callback functions to be executed on any update. Once a proposal has been accepted or rejected, the controller processes any pending system events (e.g. completed or canceled function evaluations), notifies the strategy about updates, and requests the next proposed action.

`POAP` comes with a serial controller for objective function evaluations carried out in serial. There is also a threaded controller that dispatches work to a set of workers where each worker is able to handle evaluation and kill requests. The requests are asynchronous in the sense that the workers are not required to complete the evaluation or termination requests. The worker is forced to respond to evaluation requests, but may ignore kill requests. When receiving an evaluation request, the worker should either attempt the evaluation or mark the record as killed. The worker sends status updates back to the controller by updating the relevant record. There is also an extension of the threaded controller that works with MPI and a controller that uses simulated time. The latter is useful for testing asynchronous optimization strategies for different evaluation time distributions.

### 5.3.2 Strategies

The strategy is responsible for choosing new evaluations, killing evaluations, and terminating the optimization run when a stopping criteria is reached. `POAP` provides some basic default strategies based on non-adaptive sampling and serial optimization routines and also some strategies that adapt or combine other strategies.

Different strategies can be composed by combining their control actions, which can be used to let a strategy cycle through a list of optimization strategies and select the most promising of their proposals. Strategies can also subscribe to be informed of all new function evaluations so they incorporate any new function information, even though the evaluation was proposed by another strategy. This makes it possible to start several independent strategies while still allowing each strategy to look at the function information that comes from function evaluations proposed by other strategies. As an example, we can have a local optimizer strategy running a gradient based method where the starting point can be selected based on the best point found by any other strategy. The flexibility of the `POAP` framework makes combined strategies like these straightforward.

### 5.3.3 Workers

The multi-threaded controller employs a set of workers that are capable of managing concurrent function evaluations. Each worker does not provide parallelism on its own, but the worker itself is allowed to exploit parallelism by separate external processes.

The basic worker class can call Python objective functions, which only results in parallelism if the objective function itself allows parallelism. There is also a worker class that uses subprocesses to evaluate objective functions that are not necessarily in Python. The user is responsible for specifying how to evaluate the objective function and parse partial information.

The number of workers can be adjusted dynamically during the optimization process, which is particularly useful in a cloud setting. `POAP` supports running both on the Google Cloud platform (GCP) and the Amazon Web Services (AWS). We support workers connecting to a specified TCP/IP port to communicate with the controller, making it easy to add external resources.

## 5.4 pySOT implementation

The surrogate optimization toolbox[2] (`pySOT`) is a collection of surrogate optimization strategies that can be used with the `POAP` framework. `pySOT` follows the general surrogate optimization framework in Algorithm 1 and allows using asynchrony as was described in Algorithm 5. We illustrate the communication between `POAP` and `pySOT` in Figure 5.2. `pySOT` communicates with `POAP` through the optimization strategy where the `pySOT` strategy is responsible for proposing an action when different events happen. All of the worker communication is handled by the `POAP` controller.

The `pySOT` objects follow an abstract class definition to make sure custom implementations fit the framework design. This is achieved by forcing the objects to inherit an ABC object design, which makes it easy for users to add their

[2] `pySOT` can be downloaded from: https://github.com/dme65/pySOT

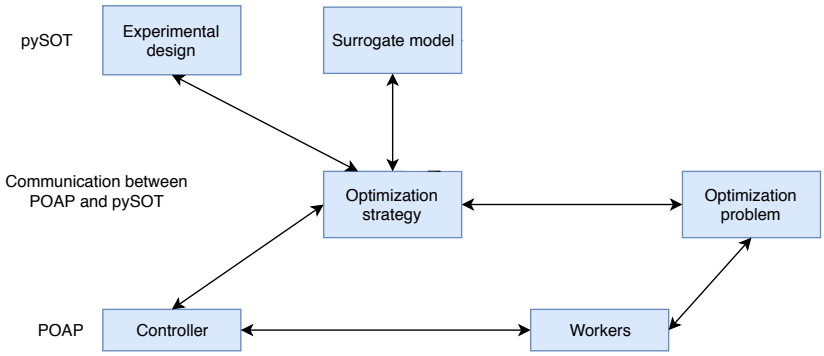



Figure 5.2: Communication between `POAP` and `pySOT`

own implementations. We proceed to describe each object and their role in the `pySOT` framework.

### 5.4.1 Strategies and auxiliary problems

The strategy object follows the `POAP` framework. `pySOT` implements an asynchronous base class for surrogate optimization which serves as a template for all surrogate optimizations in `pySOT`. This base class abstracts out the difference between serial, synchronous parallel, and asynchronous parallel. `pySOT` supports the candidate point methods SRBF and DYCORS from §1.5.2. We also support strategies for the most common acquisition functions from BO: expected improvement (EI) and the lower confidence bound (LCB), which were also introduced in §1.5.2. These acquisition functions can be minimized either using a multi-start gradient method, a genetic algorithm, or by selecting the best point from a set of randomly chosen points within the domain.

### 5.4.2 Experimental design

`pySOT` implements the symmetric Latin hypercube (SLHD), Latin hypercubes (LHD), and 2-factorial designs that were described in §1.5.1. The experimental design is always evaluated first and the asynchronous optimization strategy in `pySOT` is designed to proceed to the adaptive phase as soon as no initial design points are outstanding. Another possibility is to cancel the pending evaluations from the initial phase and proceed to the adaptive phase as soon as possible, but we choose to finish the entire initial design as exploration is important for multi-modal optimization problems. As discussed in the previous section, we must choose enough initial design point to allow building the surrogate model when all points in the initial design are either completed or pending. The experimental design object needs access to the optimization problem in order to round integer variables and potentially attempt to satisfy the analytical constraints.

### 5.4.3 Surrogate models

`pySOT` supports the many popular surrogate models, including RBFs, GPs, MARS, polynomial regression, and support vector regression. We provide our own RBF implementation that uses the incremental factorization update idea that was described in §1.3.4. Support for MARS is provided via py-earth[3] and support for GPs and polynomial regression is provided through scikit-learn [51]. The surrogate model does not need access to any of the other objects, as it just constructs a model based on the evaluated points and their values. The surrogate fitting problem may be ill-conditioned if the domain is scaled poorly,

[3] `https://github.com/scikit-learn-contrib/py-earth`

and we provide wrappers for rescaling the domain to the unit hypercube, which is particularly useful on problems where the bounds are very skewed. We add regularization to the linear system when radial basis functions are used to keep the system well-conditioned. Previous work has shown that hard-capping of function values can be useful to avoid oscillation, where a common choice is to replace all function values above the median by the median function value, and we provide wrappers for this as well.

### 5.4.4 Optimization problems

The optimization problem object specifies the number of dimensions, the number of analytical constraints, and provide methods for evaluating the objective function and the constraints. We provide implementations of many standard test problems which can be used to compare algorithms within the `pySOT` framework.

### 5.4.5 Checkpointing

Checkpointing is important when optimizing an expensive function since the optimization may run for several days or weeks, and it would be devastating if all information was lost due to e.g., a system or power failure. `pySOT` supports a controller wrapper for saving the state of the system each time something changes, making it possible to resume from the latest such snapshot.

## 5.5 Code examples

This section illustrates how `POAP` and `pySOT` can be used to minimize the Ackley test function. We will use the threaded controller and asynchronous function evaluations. The code is based on `pySOT` version 0.2.0 and `POAP` version 0.1.26.

Our goal in this example is to minimize the 10-dimensional Ackley function, which is a common test function in global optimization. We use a symmetric Latin hypercube, an RBF surrogate with a cubic kernel and linear tail, and the DYCORS strategy for generating candidate points, which was described in §1.5.2. Importing the necessary modules can be done as follows:

```
from pySOT.optimization_problems import Ackley
from pySOT.experimental_design import SymmetricLatinHypercube
from pySOT.surrogate import RBFInterpolant, CubicKernel, LinearTail
from pySOT.strategy import DYCORSStrategy
from poap.controller import ThreadController, BasicWorkerThread
import numpy as np
```

We next create objects for the optimization problem, experimental design, and surrogate model.

```
num_threads = 4
max_evals = 500

ackley = Ackley(dim=10)
rbf = RBFInterpolant(dim=ackley.dim, kernel=CubicKernel(),
                     tail=LinearTail(ackley.dim))
slhd = SymmetricLatinHypercube(dim=ackley.dim,
                               num_pts=2*(ackley.dim+1))
```

We are now ready to launch a threaded controller that will run asynchronous evaluations. We create an instance of the DYCORS strategy and append it to the controller.

```
controller = ThreadController()
controller.strategy = \
        DYCORSStrategy(opt_prob=ackley, exp_design=slhd,
                       surrogate=rbf, max_evals=max_evals,
                       asynchronous=True)
```

We need to launch the workers that do function evaluations. In this example we use standard threads and give each worker an objective function handle.

```
for _ in range(num_threads):
    worker = BasicWorkerThread(controller, ackley.eval)
    controller.launch_worker(worker)
```

The workers have been launched and the optimization strategy has been created, so we are ready to start the optimization run. The following code runs the optimizer and prints the best solution.

```
result = controller.run()
print("Best value found: {0}".format(result.value))
print("Best solution found: {0}".format(result.params[0]))
```

We can also plot the progress from the controller using the progress plot function in `pySOT`

```
from pySOT.utils import progress_plot
progress_plot(controller)
```

## 5.6 Experiments

In this section we study the performance of serial, synchronous parallel, and asynchronous parallel when varying the evaluation time distribution and the number of processors. We focus on the DYCORS method using a radial basis function surrogate with a linear tail as both have very low overhead, allowing us to run many trials, each with a large number of function evaluations. Previous work has shown that DYCORS outperforms the competing methods for computationally expensive multi-modal functions in a large number of dimensions [60]

The evaluation times are drawn from a Pareto distribution with probability density function (PDF) given by:

$$\frac{\alpha b^{\alpha}}{x^{\alpha+1}} \chi_{[b,\infty)}(x).$$

The Pareto distribution is heavy-tailed for small values of $\alpha$ and this case is suitable for studying large variance in the evaluation time. We use $b = 1$ so the support is $[1, \infty)$ and use different values of $\alpha$ to achieve different tail behaviors. This setup models homogeneous resources and spatial dependence. We use $\alpha \in \{102, 12, 2.84\}$ which corresponds to standard deviations 0.01, 0.1, and 1.

We run the serial and synchronous parallel versions with their default parameters since both methods showed good results in [58] and [59] respectively. The parameter values used for the asynchronous algorithm are the same as for the synchronous parallel version except for $\mathcal{F}_{\text{fail}}$ which we multiply by $p$ since we count evaluations rather than batches. The parameter values are shown in Table 5.1. We restart the algorithm with a new experimental design if at some point $\gamma = \gamma_{\min}$ and the algorithm has failed to make a significant improvement

in the last $\mathcal{M}_{\text{fail}}$ evaluations. Our experiments use 4, 8, 16, and 32 workers for the parallel algorithms and we give each algorithm an evaluation budget of $50 * 32 = 1600$ evaluations. This is an upper bound for a time budget of 50 units of time. We exclude the overhead from fitting the surrogate and generating candidate points since this is negligible when the function evaluations are truly expensive.

| Parameter | Value |
|---|---|
| $\lvert\Lambda_n\rvert$ (number of candidate points per proposal) | $100d$ |
| $\Upsilon$ (weight pattern) | $\langle 0.3, 0.5, 0.8, 0.95\rangle$ |
| $\kappa$ (number of weights in $\Upsilon$) | 4 |
| $\gamma_{\text{init}}$ (Initial step size) | $0.1\ell(\Omega)$ |
| $\gamma_{\text{min}}$ (minimum step size) | $0.1(1/2)^6\ell(\Omega)$ |
| $\delta_{\text{tol}}$ (radius tolerance) | $0.0025\ell(\Omega)$ |
| $\mathcal{F}_{\text{succ}}$ (threshold parameter for increasing the step size) | 3 |
| $\mathcal{F}_{\text{fail}}$ (tolerance parameter for decreasing step size) | $p\lceil\max(4/p, d/p)\rceil$ |
| $\mathcal{M}_{\text{fail}}$ (maximum failure tolerance parameter) | $4\mathcal{F}_{\text{fail}}$ |

Table 5.1: Parameter values used for the asynchronous algorithm

We consider the multimodal test problems F15-F24 from the BBOB test suite [33]. These problems are challenging and non-separable and we use the 10-dimensional versions for our experiments. The domain for each problem is $[-5, 5]^{10}$, and location of the global minimum and the value at the global optimum are generated randomly depending on what instance is being used, where we use instance 1 for each problem. These problems are not expensive to evaluate, but we pretend they are computationally expensive black-box functions and draw the evaluation time from a Pareto distribution.

We compare progress vs time and progress vs number of evaluations. Comparing progress vs time will show what method does well in practice since we

are often constrained by a time budget rather than an evaluation budget. We will also be able to see the effect of adding more processors, which is expected to be fruitful since exploration is important for multi-modal problems. We compare progress vs number of evaluations to study the importance of information. The serial and synchronous methods are independent of the evaluation time in this case since there is a barrier after each batch. The asynchronous algorithm is affected by the variance, which affects how much information is available at a given iteration. The serial version always has more points incorporated in the surrogate at a given iteration, but explores less than the parallel versions. Figure 5.3 shows the experimental results for F15 and F17.

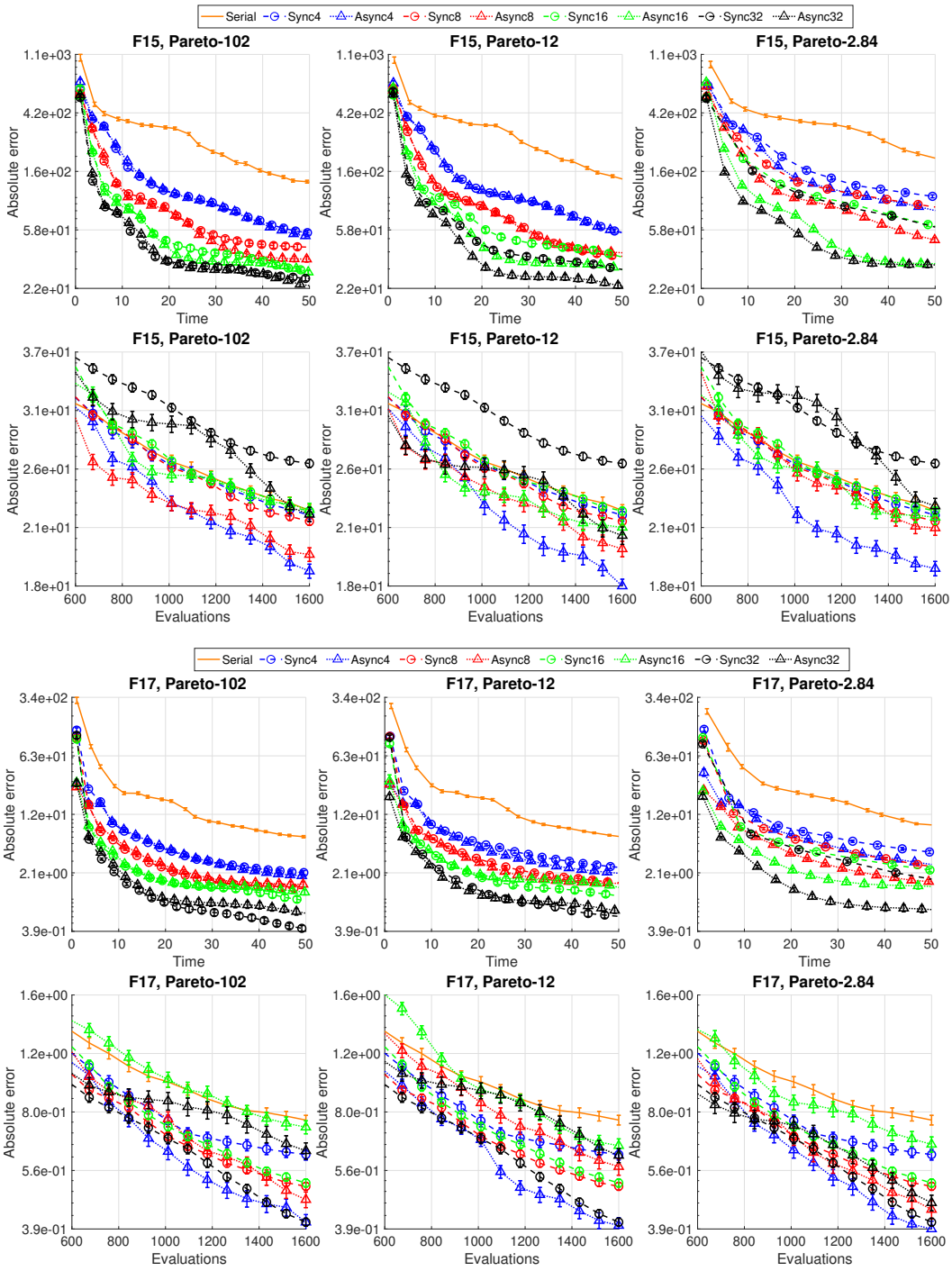



Figure 5.3: Progress vs time and progress vs number of evaluation for F15 and F17. The error bars show the standard error based on 100 trials using 1600 evaluations. The plots with respect to number of evaluations are zoomed in to make the lines easier to distinguish.

These problems are chosen because they show the key points we are trying to convey. The first row in each plot shows absolute error vs time and the second row shows absolute error vs number of evaluations. The absolute error is plotted on a log-scale in all plots to make it easier to interpret the results. This should be taken into account when looking at the error bars, which show the standard error of the estimated mean based on 100 trials. Note that each row has the same range in absolute error to make the comparison easier.

F15 illustrates a case where synchronous and asynchronous parallel perform similarly with respect to time when the variance is small. The difference grows when the variance increases and asynchrony is always superior in the large variance case. Synchrony does slightly better than asynchrony in the small variance case for F17. The results versus number evaluations are interesting and asynchrony with 4 processors is consistently the best choice for both F15 and F17. This is unexpected since the asynchronous versions never have more information than the corresponding synchronous versions, indicating that maximizing information is not as important on multi-modal problems. We also see that the serial version is outperformed by the parallel versions when looking at number of evaluations. This is another indicator that exploration is more important than information.

We can also compare the speedup from using more workers for synchronous and asynchronous parallel. Speedup is measured by the quantity

$$S(p) = \frac{T^*(1)}{T(p)} = \frac{\text{Execution time for fastest serial algorithm}}{\text{Execution time for parallel algorithm with } p \text{ processors}}.$$

This requires knowledge of the fastest serial algorithm which is hard to know given randomized initial conditions. We therefore consider relative speedup where $T^*(1)$ is replaced by $T(1)$. We will measure speedup by running syn-

chronous and asynchronous parallel and compare the results to the serial case. We need to estimate the expected value of the speedup since all of our algorithms are stochastic.

A main problem with speedup tests is choosing a good target value where the speedup is measured. For unimodal problems, such a target can be based on a small neighborhood of the global minimum, but this is unreasonable for multimodal test problems. Our approach is to run each algorithm for 100 trials, compute the intersection of the ranges of function values, and compute the speedup for a set of targets within the intersection. This allows us to see how the speedup depends on different target values. Figure 5.4 shows the speedup for F15 and F17.

We achieve close to linear speedup with asynchrony for small target values when using 4, 8, and 16 processors on F15 in the case of small variance. The speedup is larger for synchrony when we consider 32 processors, but is clearly sub-linear. The speedup is larger for small target values, which indicates that the serial algorithm is more likely to get stuck in a local minimum which triggers a restart. The effect of increasing the variance clearly degrades the performance of synchrony while the results for asynchrony do not change much. The speedup on F17 is generally better for the synchronous algorithm in the case of small variance.

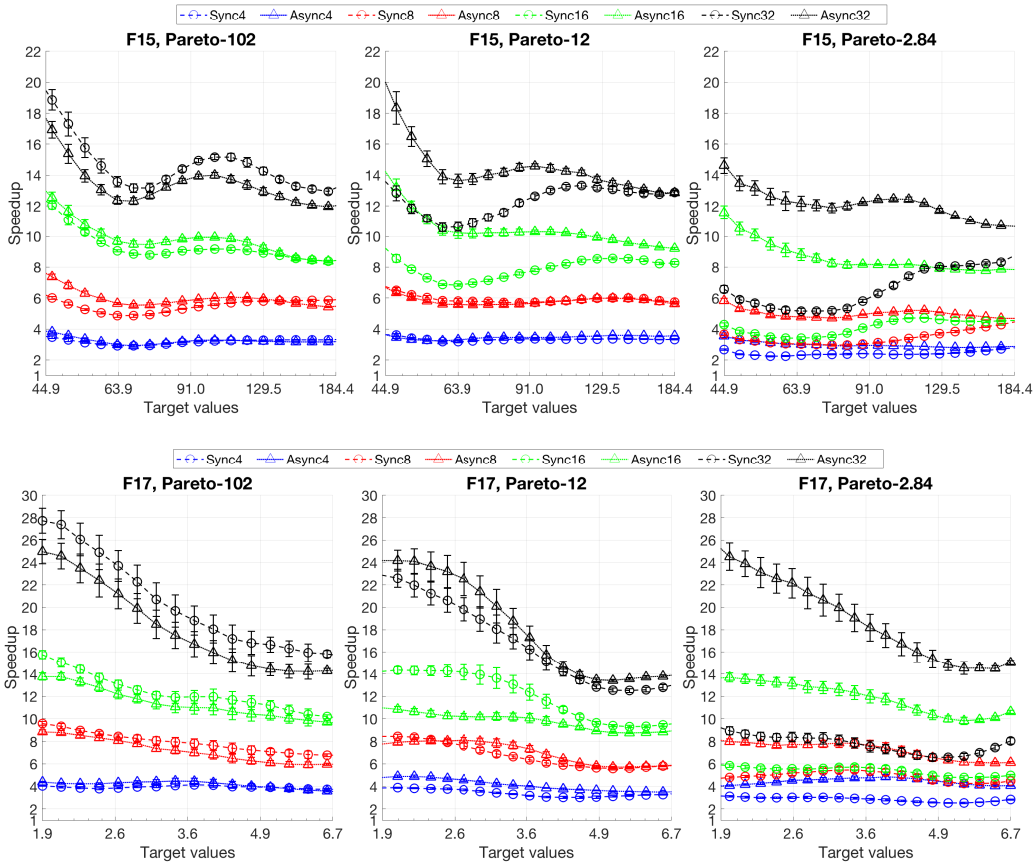



Figure 5.4: Relative speedup for different target values for F15 and F17. The error bars show the standard error based on 100 trials using 1600 evaluations.

## 5.7 Conclusion

We have described our asynchronous software package for surrogate optimization. We introduced the event-driven optimization framework POAP, which provides an easy framework to build and combine new optimization algorithms for computationally expensive functions. `POAP` has three main components, a controller, a strategy, and a collection of workers. The controller accepts or rejects proposals from the strategies, monitors the workers, and informs the

strategy when new events occur. The strategy proposes actions when an event occurs, such as starting a new function evaluation, re-evaluating an input, or terminating the optimization run. The workers perform function evaluations when instructed by the controller and they support partial updates, making it possible to terminate unpromising evaluations. The flexibility of the `POAP` framework makes it easy to combine optimization strategies.

We have also introduced the inter-operable library `pySOT`, which supports most popular surrogate optimization methods. `pySOT` is a collection of synchronous/asynchronous strategies, experimental designs, and surrogate models commonly used in surrogate optimization. It comes with a large set of standard test problems and efficiently serves as a test suite in which new optimization algorithms can be compared to existing methods using asynchronous or synchronous parallel function evaluations. `pySOT`'s object oriented design makes it easy to add new functionality, and there is also support for resuming crashed and terminated runs.

We have introduced a general asynchronous surrogate optimization method for computationally expensive black-box problems that extends the work in [59] to variable function evaluation times and heterogeneous computational resources. We also handle worker failures and evaluation crashes, which was not considered in [59]. Our numerical experiments show that asynchrony performs similarly to synchrony even when the variance in evaluation time is small. Comparing progress versus number of evaluations showed that the serial method, which maximizes the information at each step, does not outperform the parallel methods. This is likely because exploring is more important than maximizing the information for each sample.

A relative speedup analysis indicates good results for the asynchronous method and we achieve near-linear speedups for 4, 8, and 16 processors. The speedup for the synchronous method clearly decreases when the variance of the evaluation time increases, which is expected since this increases idle time. We conclude that for multi-modal problems, asynchrony should be preferred over synchrony and that adding more processors leads to faster convergence.

# CHAPTER 6

# CONCLUSION

In this dissertation, we have developed efficient methods for scaling Gaussian process regression to large and high-dimensional datasets. The methods introduced in Chapter 2 are general and rely only on fast MVMs with the kernel matrix. The biggest challenge for kernel learning is estimating the log determinant and derivatives of the kernel matrix, which we calculate efficiently by combining stochastic trace estimation and the Lanczos process. In addition to log determinants, the methods presented in Chapter 2 can be adapted to fast posterior sampling, diagonal estimation, matrix square roots, and many other standard operations. The proposed methods only depend on fast MVMs—and the structure necessary for fast MVMs often exists, or can be readily created. We made use of SKI [75] in Chapter 2 to create such structure, but other approaches, such as stochastic variational methods [35], can be used or combined with SKI for fast MVMs, as in [76].

To further demonstrate the generality of the approach in Chapter 2, we showed in Chapter 3 how to extend this work to incorporate gradient information. Given a fast MVM with the kernel matrix, we only need to differentiate this approximation to use the framework from Chapter 2. We showed how to do this for SKI, and also SKIP [25], and refer to these methods as D-SKI and D-SKIP. A pivoted Cholesky preconditioner accelerates convergence of the iterative methods and active subspace calculation reveals low-dimensional structure when gradients are available. We also illustrated how these methods can be used to also scale Bayesian optimization to larger evaluation budgets and higher dimensions with derivative information. There are several possible av-

enues for future work. D-SKIP shows promising scalability, but has large overheads, and is expensive for Bayesian optimization as it must be recomputed from scratch with each new data point. We believe kernel function approximation via Chebyshev interpolation and tensor approximation will likely provide similar accuracy with greater efficiency. Extracting low-dimensional structure through active subspace computation is highly effective in our experiments and deserves an independent, more thorough treatment. Finally, our work in scalable Bayesian optimization with gradients represents a step towards the unified view of global optimization methods (i.e. Bayesian optimization) and gradient-based local optimization methods (i.e. BFGS).

Chapter 4 introduced a new optimization algorithm based on the convergence theory for radial basis functions (RBFs). This optimization algorithm has strong theoretical guarantees and we have proved convergence without dense sampling if the target function is in the native space of the RBF kernel and an upper bound on the native space semi-norm is known. Future work includes extending the algorithm to the case when the target function is not in the native space and showing how to estimate the semi-norm given a global regularity condition such as Lipschitz. One possible avenue to extend our algorithm is via mollification of the target function so the mollified function is in the native space of the given kernel. It may also be possible to prove convergence rates that agree with the RBF theory, but proving this is technically challenging as existing theory based on the fill distance does not apply.

Finally, Chapter 5 introduced a computational framework for asynchronous surrogate optimization. Our event-driven optimization framework `POAP` makes it easy to build and combine new optimization algorithms for computationally

expensive functions, while `pySOT` implements many popular surrogate optimization methods in the `POAP` framework. We used `POAP` and `pySOT` to compare the performance of serial, synchronous parallel, and asynchrony on a large set of test problems and motivated why asynchronous parallel should be preferred over synchronous parallel. Using a large number of computational resources makes it possible to do thousands of evaluations, in which case updating the surrogate model becomes a bottleneck. This is especially true when we observe gradient information and future work involves using the results in Chapter 2 and Chapter 3 in this setting.

## BIBLIOGRAPHY


[1] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58(2), 2011.

[2] Z. Bai, M. Fahey, G. Golub, M. Menon, and E. Richter. Computing partial eigenvalue sum in electronic structure calculations. Technical report, 1997.

[3] C. Bekas, E. Kokiopoulou, and Y. Saad. An estimator for the diagonal of a matrix. *Appl. Numer. Math.*, 57(11-12), 2007.

[4] Einat Neumann Ben-Ari and David M Steinberg. Modeling data from computer experiments: an empirical comparison of Kriging with MARS and projection pursuit regression. *Quality Engineering*, 19(4):327–338, 2008.

[5] Martin Dietrich Buhmann. Radial basis functions. *Acta Numerica 2000*, 9: 1–38, 2000.

[6] Adam D Bull. Convergence rates of efficient global optimization algorithms. *The Journal of Machine Learning Research*, 12:2879–2904, 2011.

[7] Puget Sound LiDAR Consortium. Mount Saint Helens LiDAR data. University of Washington, 2002.

[8] Paul G Constantine. *Active subspaces: Emerging ideas for dimension reduction in parameter studies*, volume 2. SIAM, 2015.

[9] Alberto Costa and Giacomo Nannicini. RBFOpt: an open-source library for black-box optimization with costly function evaluations. *Optimization Online*, 4538, 2014.

[10] Jane K Cullum and Ralph A Willoughby. *Lanczos algorithms for large symmetric eigenvalue computations: Vol. I: Theory*. SIAM, 2002.

[11] Philip C Curtis et al. *n*-parameter families and best approximation. *Pacific Journal of Mathematics*, 9(4):1013–1027, 1959.

[12] Kurt Cutajar, Michael Osborne, John Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *International Conference on Machine Learning*, pages 2529–2538, 2016.

[13] Kun Dong, David Eriksson, Hannes Nickisch, David Bindel, and Andrew G Wilson. Scalable log determinants for Gaussian process kernel learning. In *Advances in Neural Information Processing Systems*, pages 6330–6340, 2017.

[14] Tobin A Driscoll and Bengt Fornberg. Interpolation in the limit of increasingly flat radial basis functions. *Computers & Mathematics with Applications*, 43(3-5):413–422, 2002.

[15] Michael S Eldred, Anthony A Giunta, Bart G van Bloemen Waanders, Steven F Wojtkiewicz, William E Hart, and Mario P Alleva. *DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 4.1 reference manual*, 2007.

[16] David Eriksson, Kun Dong, , Eric H Lee, David Bindel, and Andrew G Wilson. Scaling gaussian process regression with derivatives. In *Advances in Neural Information Processing Systems*, 2018.

[17] Gregory E Fasshauer. *Meshfree Approximation Methods with Matlab*, volume 6. World Scientific Publishing Company, 2007.

[18] Gregory E Fasshauer and Jack G Zhang. On choosing "optimal" shape parameters for rbf approximation. *Numerical Algorithms*, 45(1-4):345–368, 2007.

[19] Seth Flaxman, Andrew Wilson, Daniel Neill, Hannes Nickisch, and Alex Smola. Fast Kronecker inference in Gaussian processes with non-Gaussian likelihoods. In *International Conference on Machine Learning*, pages 607–616, 2015.

[20] Bengt Fornberg and Grady Wright. Stable computation of multiquadric interpolants for all values of the shape parameter. *Computers & Mathematics with Applications*, 48(5):853–867, 2004.

[21] Bengt Fornberg, Elisabeth Larsson, and Natasha Flyer. Stable computations with gaussian radial basis functions. *SIAM Journal on Scientific Computing*, 33(2):869–892, 2011.

[22] Alexander Forrester and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[23] Peter I Frazier. A tutorial on Bayesian optimization. *arXiv:1807.02811*, 2018.

[24] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[25] Jacob R Gardner, Geoff Pleiss, Ruihan Wu, Kilian Q Weinberger, and Andrew Gordon Wilson. Product kernel interpolation for scalable Gaussian processes. *arXiv:1802.08903*, 2018.

[26] Amparo Gil, Javier Segura, and Nico Temme. *Numerical methods for special functions*. SIAM, 2007.

[27] David Gingras, Tom Lamarche, Jean-Luc Bedwani, and Érick Dupuis. Rough terrain reconstruction for rover motion planning. In *Computer and Robot Vision (CRV), 2010 Canadian Conference on*, pages 191–198, 2010.

[28] Gene Golub and Gérard Meurant. *Matrices, moments and quadrature with applications*. Princeton University Press, 2010.

[29] H-M Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19(3):201–227, 2001.

[30] Hans-Martin Gutmann. On the semi-norm of radial basis function interpolants. *Journal of Approximation Theory*, 111(2):315–328, 2001.

[31] Raia Hadsell, J Andrew Bagnell, Daniel Huber, and Martial Hebert. Space-carving kernels for accurate rough terrain estimation. *The International Journal of Robotics Research*, 2010.

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

[33] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical report, INRIA, 2009.

[34] Helmut Harbrecht, Michael Peters, and Reinhold Schneider. On the low-rank approximation by the pivoted Cholesky decomposition. *Applied numerical mathematics*, 62(4):428–440, 2012.

[35] J Hensman, N Fusi, and N.D. Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence (UAI)*, 2013.

[36] Nicholas J Higham. *Functions of matrices: theory and computation*. SIAM, 2008.

[37] Yannick Hold-Geoffroy, Olivier Gagnon, and Marc Parizeau. Once you scoop, no need to fork. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, page 60, 2014.

[38] Reiner Horst and Panos M Pardalos. *Handbook of global optimization*, volume 2. Springer Science & Business Media, 2013.

[39] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2), 1990.

[40] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4), 1998.

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

[42] Kurt Konolige, Motilal Agrawal, and Joan Sola. Large-scale visual odometry for rough terrain. In *Robotics research*. Springer, 2010.

[43] Q. Le, T. Sarlos, and A. Smola. Fastfood – computing Hilbert space expansions in loglinear time. In *Proceedings of the 30th International Conference on Machine Learning*, pages 244–252, 2013.

[44] Sébastien Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 37(4), 2011.

[45] Ives Macedo, Joao Paulo Gois, and Luiz Velho. Hermite radial basis functions implicits. In *Computer Graphics Forum*, volume 30, pages 27–42, 2011.

[46] John C Mairhuber. On Haar's theorem concerning Chebychev approximation problems having unique solutions. *Proceedings of the American Mathematical Society*, 7(4):609–615, 1956.

[47] Ruben Martinez-Cantin. BayesOpt: A Bayesian optimization library for nonlinear optimization, experimental design and bandits. *The Journal of Machine Learning Research*, 15(1):3735–3739, 2014.

[48] MATLAB Mapping Toolbox. Matlab mapping toolbox, 2017.

[49] Erik HW Meijering, Karel J Zuiderveld, and Max A Viergever. Image reconstruction by convolution with symmetrical piecewise *n*th-order polynomial kernels. *IEEE transactions on image processing*, 8(2):192–201, 1999.

[50] Juliane Mueller. MATSuMoTo: The MATLAB surrogate model toolbox for computationally expensive black-box global optimization problems. *arXiv:1404.4261*, 2014.

[51] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12:2825–2830, 2011.

[52] Todd D Plantenga. Hopspack 2.0 user manual. *Sandia National Laboratories Technical Report Sandia National Laboratories Technical Report SAND2009-6265*, 2009.

[53] Rodrigo B Platte and Tobin A Driscoll. Polynomials and potential theory

for gaussian radial basis function interpolation. *SIAM Journal on Numerical Analysis*, 43(2):750–766, 2005.

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

[55] Joaquin Quinonero-Candela, Carl Edward Rasmussen, and Christopher KI Williams. Approximation methods for Gaussian process regression. *Large-scale kernel machines*, pages 203–223, 2007.

[56] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for Machine Learning*. The MIT Press, 2006.

[57] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[58] Rommel G Regis and Christine A Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509, 2007.

[59] Rommel G Regis and Christine A Shoemaker. Parallel stochastic global optimization using radial basis functions. *INFORMS Journal on Computing*, 21(3):411–426, 2009.

[60] Rommel G Regis and Christine A Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45(5):529–555, 2013.

[61] Youcef Saad. *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992.

[62] Scott A Sarra and Edward J Kansa. Multiquadric radial basis function approximation methods for the numerical solution of partial differential equations. *Advances in Computational Mechanics*, 2(2), 2009.

[63] Robert Schaback and Holger Wendland. Kernel techniques: from machine learning to meshless methods. *Acta Numerica*, 15:543–639, 2006.

[64] Bernhard W Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–52, 1985.

[65] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems (NIPS)*, volume 18, page 1257. MIT Press, 2006.

[66] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. http://www.sfu.ca/ ssurjano, 2018.

[67] Aimo Törn and Antanas Žilinskas. *Global optimization*. Springer, 1989.

[68] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of tr($F(A)$) via stochastic Lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4), 2017.

[69] Grace Wahba. Improper priors, spline smoothing and the problem of guarding against model errors in regression. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 364–372, 1978.

[70] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando De Freitas. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pages 1778–1784, 2013.

[71] Andrew J Wathen and Shengxin Zhu. On spectral distribution of kernel matrices related to radial basis functions. *Numerical Algorithms*, 70(4):709–726, 2015.

[72] Holger Wendland. *Scattered data approximation*, volume 17. Cambridge university press, 2004.

[73] Stefan M Wild, Rommel G Regis, and Christine A Shoemaker. ORBIT: Optimization by radial basis function interpolation in trust-regions. *SIAM Journal on Scientific Computing*, 30(6):3197–3219, 2008.

[74] Oliver Williams and Andrew Fitzgibbon. Gaussian process implicit surfaces. *Gaussian Proc. in Practice*, pages 1–4, 2007.

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

[76] Andrew G Wilson, Zhiting Hu, Ruslan R Salakhutdinov, and Eric P Xing. Stochastic variational deep kernel learning. In *Advances in Neural Information Processing Systems*, pages 2586–2594, 2016.

[77] Andrew Gordon Wilson. *Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

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

[79] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P

Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 370–378, 2016.

[80] Jian Wu and Peter Frazier. The parallel knowledge gradient method for batch Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 3126–3134, 2016.

[81] Jian Wu, Matthias Poloczek, Andrew G Wilson, and Peter Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems*, pages 5273–5284, 2017.