Scalable First-Order Bayesian Optimization via Structured Automatic Differentiation

Sebastian Ament [1] **Carla Gomes** [1]

Abstract

Bayesian Optimization (BO) has shown great promise for the global optimization of functions that are expensive to evaluate, but despite many successes, standard approaches can struggle in high dimensions. To improve the performance of BO, prior work suggested incorporating gradient information into a Gaussian process surrogate of the objective, giving rise to kernel matrices of size $nd \times nd$ for n observations in d dimensions. Naïvely multiplying with (resp. inverting) these matrices requires $\mathcal{O}(n^2d^2)$ (resp. $\mathcal{O}(n^3d^3)$) operations, which becomes infeasible for moderate dimensions and sample sizes. Here, we observe that a wide range of kernels gives rise to structured matrices, enabling an *exact* $\mathcal{O}(n^2d)$ matrix-vector multiply for gradient observations and $\mathcal{O}(n^2d^2)$ for Hessian observations. Beyond canonical kernel classes, we derive a programmatic approach to leveraging this type of structure for transformations and combinations of the discussed kernel classes, which constitutes a structure-aware automatic differentiation algorithm. Our methods apply to virtually all canonical kernels and automatically extend to complex kernels, like the neural network, radial basis function network, and spectral mixture kernels without any additional derivations, enabling flexible, problem-dependent modeling while scaling first-order BO to high d.

1. Introduction

Bayesian Optimization (BO) has demonstrated tremendous promise for the global optimization of functions, in particular those that are expensive to evaluate (Shahriari et al., 2016; Frazier, 2018). Instantiations of BO can be found in Active Learning (AL) (Settles, 2009; Tuia et al., 2011; Fu et al., 2013), the optimal design of experiments (Chaloner and Verdinelli, 1995; Foster et al., 2019; Zheng et al., 2020), and Optimal Learning (Powell and Ryzhov, 2012). Its applications range widely from the optimization of hyper-parameters of complex machine learning models (Snoek et al., 2012) to the sciences and engineering as Attia et al. (2020), who optimized charging protocols to maximize battery life. Li et al. (2018) reported that random search with only twice as many samples can outperform standard BO methods on a certain hyper-parameter optimization task. This lead Ahmed et al. (2016) to advocate for first-order BO (FOBO) as a critical improvement, a call that recently received theoretical heft due to Shekhar and Javidi (2021), who proved that FOBO achieves an *exponential* improvement on the expected regret of standard BO for multi-armed bandit problems as a function of the number of observations n and dimensionality d of the input.

At the same time, differentiable programming and automatic differentiation (AD), which enable the calculation of gradients through complex numerical programs, have become an integral part of machine learning research (Innes et al., 2017; Wang et al., 2018; Baydin et al., 2018) and practice, perhaps best illustrated by PyTorch (Paszke et al., 2019) and Tensorflow (Abadi et al., 2015), both of which include AD engines. Certainly, AD has powered an increasing pace of model development by automating the error-prone writing of derivative code and is thus a natural complement to FOBO, if only to compute the gradients of the objective.

On a high level, most BO approaches build a surrogate model of an objective with a few potentially noisy observations and make informed choices about further queries based on predictive values and uncertainties of the surrogate. In principle, any functional form could be employed as a surrogate, and indeed Wang and Shang (2014), Snoek et al. (2015), and Gal et al. (2017) use deep neural networks for AL and BO. However, Gaussian Processes (GP) are currently the most commonly used models for research and applications of BO because they work well with little data and permit closed-form posterior inference. Fortunately, GPs are closed under differentiation with benign assumptions, see Section A, and maintain their analytical properties when conditioned on gradient information (Solak et al., 2003).

[1]Department of Computer Science, Cornell University, Ithaca, NY, 14850, USA. Correspondence to: Sebastian Ament <ament@cs.cornell.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Nonetheless, naïvely incorporating gradients leads to kernel matrices of size $nd \times nd$, for n observations in d dimensions, which restricts the possible problem sizes and dimensions, a problem that needs to be overcome to make FOBO applicable to a wide array of problems. Further, as the performance of GPs chiefly depends on their covariance kernel, it is important to give researchers and practitioners flexibility in this choice. Herein, it is our primary goal to enable *scalabe inference* for GPs in the context of FOBO, while maintaining *modeling flexibility* via matrix-structure-aware AD.

Contributions We 1) derive analytical block-data-sparse structures for a large class of gradient kernel matrices, allowing for an exact $\mathcal{O}(n^2d)$ multiply in Section 3, 2) propose an AD framework that programmatically computes the data-sparse block structures for transformations, and algebraic combinations of kernels and make our implementation publicly available[1]. In Section 3.3, we further 3) derive analogous structures for kernel matrices that arise from conditioning on Hessian information, reducing the complexity from $\mathcal{O}(n^2d^4)$ for the naïve approach to $\mathcal{O}(n^2d^2)$, 4) provide numerical experiments that demonstrate the improved scaling and delineate the problem sizes for which the proposed methods are applicable in Section 4.1, 5) compare against existing techniques in Section 4.2 and 6) use the proposed methods for Bayesian Optimization in Section 4.3.

2. Related Work

Gaussian Processes Inference for GPs has traditionally been based on matrix factorizations, but recently, methods based on iterative solvers have been developed, which can scale up to a million data points without approximations (Wang et al., 2019) by leveraging the parallelism of modern hardware (Dong et al., 2017; Gardner et al., 2018a). Extending the approximate matrix-vector multiplication algorithms of Wilson and Nickisch (2015) and Gardner et al. (2018b), Eriksson et al. (2018) proposed an approximate method for GPs with derivative information which scales quasi-linearly in n for separable product kernels whose constituents are stationary. De Roos et al. (2021) proposed an elegant direct method for GPs with derivatives that scales linearly in the dimensionality but sextically – $\mathcal{O}(n^6 + n^2d)$ – with the number of data points and also derive an efficient multiply for dot-product and isotropic kernels whose inputs can be scaled by a diagonal matrix. Wu et al. (2017b) used GPs with gradients for BO and proposed keeping only a single directional derivative to reduce the computational cost. Padidar et al. (2021) proposed a similar strategy, retaining only relevant directional derivatives, to scale a variational inference scheme for GPs with derivatives. Notably, incorporating gradient information into GPs is not only useful for BO: Solak et al. (2003) put forward the integration of gradient information for GP models of dynamical systems, Riihimäki and Vehtari (2010) used "virtual" derivative observations to include monotonicity constraints into GPs, and Solin et al. (2018) employed the derivatives of a GP to model curl-free magnetic fields and their physical constraints.

Automatic Differentiation To disambiguate several sometimes conflated terms, we quote Baydin et al. (2018), who defined AD as "a specific family of techniques that computes derivatives through accumulation of values during code execution to generate numerical derivative evaluations rather than derivative expressions". It enables the computation of derivatives up to machine precision while maintaining the speed of numerical operations. Practical implementations of AD include forward-mode differentiation techniques based on operator overloading (Revels et al., 2016), the ∂P system of Innes (2018), which is able to generate compiled derivative code of differentiable components of the Julia language, as well as the reverse-mode differentiation technologies of PyTorch (Paszke et al., 2019) and Tensorflow (Abadi et al., 2015). Maclaurin et al. (2015) put forward an algorithm for computing gradients of models w.r.t. their *hyper-parameters* using reverse-mode auto-differentiation, enabling the use of FOBO to optimize a model's generalization performance. Among others, Verma (1998) explored the exploitation of structure, primarily sparsity, in the automatic computation of Jacobian and Hessian matrices. However, the existing work is not directly applicable here, since it does not treat the more general *data-sparse* structures of Section 3. For a review of automatic differentiation (AD) techniques, see (Griewank and Walther, 2008).

Bayesian Optimization Bayesian Optimization (BO) has been applied to a diverse set of problems, and of particular interest to the machine learning community is the optimization of hyper-parameters of complex models (Klein et al., 2017). Spurring much interest in BO, Snoek et al. (2012) demonstrated that BO is an effective tool for the optimization of hyper-parameters of deep neural networks. Hennig and Schuler (2012) proposed entropy search for global optimization, a technique that employs GPs to compute a distribution over the potential optimum of a function. Wang et al. (2013) proposed efficient BO with random embeddings which scales to very high-dimensional problems by exploiting lower-dimensional structures. Mutny and Krause (2018) assumed an additive structure to scale BO to high dimensions. Eriksson et al. (2018) used their fast approximate inference technique for FOBO in combination with an active subspaces method (Constantine et al., 2014) in order to reduce the dimensionality of the optimization problem and to speed up convergence. Martinez-Cantin et al. (2018) enabled BO in the presence of outliers by employing a heavy-tailed likelihood distribution. Malkomes and Garnett

[1] `github.com/SebastianAment/CovarianceFunctions.jl`

(2018) used BO in model space to choose surrogate models for use in a primary BO loop. Wu and Frazier (2019) presented a two-step lookahead method for BO. Eriksson et al. (2019) put forth `TuRBO`, leveraging a set of local models for the global optimization of high-dimensional functions. BO is also applied to hierarchical reinforcement learning (Brochu et al., 2010; Prabuchandran et al., 2021). Existing BO libraries include Dragonfly (Kandasamy et al., 2020), BayesOpt (Martinez-Cantin, 2014), and BoTorch (Balandat et al., 2020). For a review of BO, see (Frazier, 2018).

3. Methods

3.1. Preliminaries

We first provide definitions and set up notation and central quantities for the rest of the paper.

Definition 3.1. A random function f is a Gaussian process with a mean μ and covariance function k if and only if all of its finite-dimensional marginal distributions are multivariate Gaussian distributions. In particular, f is a Gaussian process if and only if for any finite set of inputs $\{\mathbf{x}_i\}$,

$$\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}),$$

where $f_i = f(\mathbf{x}_i)$, $\mu_i = \mu(\mathbf{x}_i)$ and $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. In this case, we write $f \sim \mathcal{GP}(\mu, k)$.

When defining kernel functions, $\mathbf{x}$ and $\mathbf{y}$ will denote the first and second inputs, $\mathbf{r} = \mathbf{x} - \mathbf{y}$ their difference, $\mathbf{I}_d$ the d-dimensional identity matrix, and $\mathbf{1}_d$ the all-ones vector of length d. The gradient and Jacobian operators with respect to $\mathbf{x}$ will be denoted by $\nabla_{\mathbf{x}}$ and $\mathbf{J}_{\mathbf{x}}$, respectively.

The G Operator The focus of the present work is the matrix-valued operator $\mathbf{G} = \nabla_{\mathbf{x}}\nabla_{\mathbf{y}}^\top$ that acts on kernel functions $k(\mathbf{x}, \mathbf{y})$ and whose entries are $G_{ij} = \partial_{x_i}\partial_{y_j}$. We will show that $\mathbf{G}[k]$ is highly structured and data-sparse for a vast space of kernel functions and present an automatic structure-aware algorithm for the computation of $\mathbf{G}$. The kernel matrix $\mathbf{K}^\nabla = \mathbf{G}[k](\mathbf{X})$ that arises from the evaluation of $\mathbf{G}[k]$ on the data $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_n]$ can be seen as a block matrix whose (i, j)th block is $\mathbf{K}^\nabla_{ij} = \mathbf{G}[k](\mathbf{x}_i, \mathbf{x}_j)$. For isotropic and dot-product kernels, De Roos et al. (2021) discovered that $\mathbf{K}^\nabla$ has the structure $\mathbf{K}^\nabla = k'(\mathbf{X}) \otimes \mathbf{I}_d + [\text{rank-}n^2 \text{ matrix}]$, which allows a linear-in-d direct inversion, though the resulting $\mathcal{O}(n^6)$-scaling only applies to the low-data regime. Rather than deriving similar global structure, we focus on efficient structure for the blocks $\mathbf{G}[k](\mathbf{x}_i, \mathbf{x}_j)$, which is more readily amenable to a fully lazy implementation with $\mathcal{O}(1)$ memory complexity, and the synthesis of several derivative orders, see Sec. D for details. Last, we stress that our goal here is to focus on the *subset* of transformations that arise in most kernel functions, and *not* the derivation of a fully general structured AD engine for the computation of the $\mathbf{G}$ operator.

3.2. Gradient Kernel Structure

In this section, we derive novel structured representations of $\mathbf{G}[k]$ for a large class of kernels k. The only similar previously known structures are for isotropic and dot-product kernels derived by De Roos et al. (2021).

Input Types The majority of canonical covariance kernels can be written as

$$k(\mathbf{x}, \mathbf{y}) = f(\text{proto}(\mathbf{x}, \mathbf{y})),$$

where $\text{proto}(\mathbf{x}, \mathbf{y}) = (\mathbf{r} \cdot \mathbf{r})$, $(\mathbf{c} \cdot \mathbf{r})$, or $(\mathbf{x} \cdot \mathbf{y})$, f is a scalar-valued function, and $\mathbf{c} \in \mathbb{R}^d$. The first two types make up most of commonly used stationary covariance functions, while the last constitutes the basis of many popular non-stationary kernels. We call the choice of proto isotropic, stationary linear functional, and dot product, respectively. First, we note that $\mathbf{G}[\text{proto}]$ is simple for all three choices:

$$\mathbf{G}[\mathbf{r} \cdot \mathbf{r}] = -\mathbf{I}_d, \quad \mathbf{G}[\mathbf{c} \cdot \mathbf{r}] = \mathbf{0}_{d \times d}, \quad \text{and} \quad \mathbf{G}[\mathbf{x} \cdot \mathbf{y}] = \mathbf{I}_d.$$

Kernels with the first and third input type are ubiquitous and include the exponentiated quadratic, rational quadratic, Matérn, and polynomial kernels. An important example of the second type is the cosine kernel, which has been used to approximate stationary kernels (Rahimi et al., 2007; Lázaro-Gredilla et al., 2010; Gal and Turner, 2015) and is also a part of the spectral mixture kernel (Wilson and Adams, 2013). In the following, we systematically treat most of the kernels and transformations in (Rasmussen and Williams, 2005) to greatly expand the class of kernels for which structured representations are available.

A Chain Rule Many kernels can be expressed as $k = f \circ g$ where g is scalar-valued. For these types of kernels, we have

$$\mathbf{G}[f \circ g] = (f' \circ g)\, \mathbf{G}[g] + (f'' \circ g)\, \nabla_{\mathbf{x}}[g] \nabla_{\mathbf{y}}[g]^\top.$$

That is, $\mathbf{G}[f \circ g]$ is a rank-one correction to $\mathbf{G}[g]$. If $\mathbf{G}[g]$ is structured with $\mathcal{O}(d)$ data, $\mathbf{G}[f \circ g]$ inherits this property. As an immediate consequence, $\mathbf{G}[k]$ permits a matrix-vector multiply in $\mathcal{O}(d)$ time for all isotropic, stationary, and dot-product kernels that fall under the categories outlined above. However, there are combinations and transformations of these base kernels that give rise to more complex kernels and enable more flexible, problem-dependent modeling.

Sums and Products First, covariance kernels are closed under addition and multiplication. If all summands or coefficients are of the the same input-type, the sum kernel has the same input type since $(f \circ \text{proto}) + (g \circ \text{proto}) = (f + g) \circ \text{proto}$ and similarly for products, so that no special treatment is necessary beside the chain rule above. An interesting case occurs when we combine kernels of different input types or more complex composite kernels. For

$k = \sum_i^r k_i$, we trivially have $\mathbf{G}[k] = \sum_i^r \mathbf{G}[k_i]$, and so the complexity of multiplying with $\mathbf{G}[k]$ is $\mathcal{O}(dr)$. For product kernels $k(\mathbf{x}, \mathbf{y}) = g(\mathbf{x}, \mathbf{y})h(\mathbf{x}, \mathbf{y})$, we have

$$\mathbf{G}[k] = \mathbf{G}[g]h + g\mathbf{G}[h] + \nabla_\mathbf{x}[g]\ \nabla_\mathbf{y}[h]^\top + \nabla_\mathbf{x}[h]\ \nabla_\mathbf{y}[g]^\top,$$

which is a rank-two correction to the sum of the scaled constituent gradient kernels elements – $\mathbf{G}g$ and $\mathbf{G}h$ – and therefore only adds $\mathcal{O}(d)$ operations to the multiplication with the constituent elements. In general, the application of $\mathbf{G}$ to a product of r kernels $k = \prod_i^r k_i$ gives rise to a rank-r correction to the sum of the constituent gradient kernels:

$$\mathbf{G}[k] = \sum_{i=1}^{r} \mathbf{G}[k_i]p_i + \mathbf{J}_\mathbf{x}[\mathbf{k}]^\top\ \mathbf{P}\ \mathbf{J}_\mathbf{y}[\mathbf{k}], \tag{1}$$

where $p_i = \prod_{j\neq i} k_i$ and $P_{ij} = \prod_{t\neq i,j} k_t$, whose formation would generally be $\mathcal{O}(r^2)$. However, if $k_i \neq 0$ for all i, we have $p_i = k/k_i$ and $\mathbf{P} = k\ \mathbf{D}_\mathbf{k}^{-1}\ (\mathbf{1}_r\mathbf{1}_r^\top - \mathbf{I}_r)\ \mathbf{D}_\mathbf{k}^{-1}$, where $\mathbf{k} = [k_1, \ldots, k_r]$, and $\mathbf{D}_\mathbf{k}$ is the diagonal matrix with $\mathbf{k}$ on the diagonal. A matrix-vector multiplication with (1) can thus be computed in $\mathcal{O}(dr)$. If $r \sim d$, the expression is generally not data-sparse unless the Jacobians are, which is the case for the following special type of kernel product.

Direct Sums and Products Given a set of d kernels $\{k_i\}$ each of which acts on a different input dimension, we can define their direct product (resp. sum) as $k(\mathbf{x}, \mathbf{y}) = \prod_i k_i(x_i, y_i)$ (resp. $\sum_i k_i(x_i, y_i)$), where x_i corresponds to the dimension on which k_i acts. This separable structure gives rise to sparse differential operators $\mathbf{G}k$ and $\mathbf{J}_\mathbf{x}k$ that are zero except for

$$[\mathbf{G}k_i]_{ii} = [\partial_{x_i}\partial_{y_i}k_i]\prod_{j\neq i} k_j, \quad \text{and} \quad [\mathbf{J}_\mathbf{x}\mathbf{k}]_{ii} = \partial_{x_i}k_i.$$

For direct sums, $\mathbf{G}k$ is then simply diagonal: $\mathbf{G}_{ii}k = \partial_{x_i}\partial_{y_i}k_i$. For direct products, substituting these sparse expressions into the general product rule (1) above yields a rank-one update to a diagonal matrix. Therefore, the computational complexity of multiplying a vector with $\mathbf{G}[k](\mathbf{x}, \mathbf{y})$ for separable kernels is $\mathcal{O}(d)$. Notably, the above structure can be readily generalized for block-separable kernels, whose constituent kernels act on more than one dimension. The $\mathcal{O}(d)$ complexity is also attained as long as every constituent kernel only applies to a constant number of dimensions as $d \to \infty$, or itself allows a multiply that is linear in the dimensionality of the space on which it acts.

Vertical Rescaling If $k(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})h(\mathbf{x}, \mathbf{y})f(\mathbf{y})$ for a scalar-valued f, then

$$\mathbf{G}[k](\mathbf{x}, \mathbf{y}) = f(\mathbf{x})\mathbf{G}[h](\mathbf{x}, \mathbf{y})f(\mathbf{y}) +$$
$$\nabla_\mathbf{x}\begin{bmatrix} f(\mathbf{x}) & k(\mathbf{x}, \mathbf{y}) \end{bmatrix}\begin{bmatrix} h(\mathbf{x}, \mathbf{y}) & f(\mathbf{y}) \\ f(\mathbf{x}) & 0 \end{bmatrix}\nabla_\mathbf{y}\begin{bmatrix} f(\mathbf{y}) & k(\mathbf{x}, \mathbf{y}) \end{bmatrix}^\top$$

Again, $\mathbf{G}[k]$ is a low-rank (rank two) correction to $\mathbf{G}[h]$.

Warping The so called "warping" of inputs to GPs is an important technique for the incorporation of non-trivial problem structure, especially of a non-stationary nature (Snelson et al., 2004; Lázaro-Gredilla, 2012; Marmin et al., 2018). In particular, given some potentially vector-valued warping function $\mathbf{u} : \mathbf{R}^d \to \mathbf{R}^r$ a warped kernel can be written as $k(\mathbf{x}, \mathbf{y}) = h(\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{y}))$, which leads to

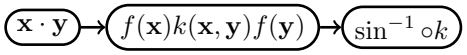
$$\mathbf{G}[k](\mathbf{x}, \mathbf{y}) = \mathbf{J}[\mathbf{u}](\mathbf{x})^\top\ \mathbf{G}[h](\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{y}))\ \mathbf{J}[\mathbf{u}](\mathbf{y}).$$

We can factor out the Jacobian factors as block-diagonal matrices $\text{diag}(\mathbf{J}[\mathbf{u}](\mathbf{X}))_{ii} = \mathbf{J}[\mathbf{u}](\mathbf{x}_i)$ from the gradient kernel matrix $\mathbf{K}^\nabla$, leading to an efficient representation:

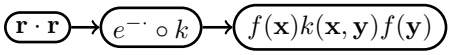
$$\mathbf{K}^\nabla = \text{diag}(\mathbf{J}[\mathbf{u}](\mathbf{X}))^\top\ \mathbf{H}^\nabla\ \text{diag}(\mathbf{J}[\mathbf{u}](\mathbf{X})).$$

Taking advantage of the above structure, the complexity of multiplication with the gradient kernel matrix can be reduced to $\mathcal{O}(n^2r + ndr)$, which is $\mathcal{O}(n^2d)$ for $n > d \geq r$. Important examples of warping functions are energetic norms or inner products of the form $\mathbf{r}^\top\mathbf{E}\mathbf{r}$ or $\mathbf{x}^\top\mathbf{E}\mathbf{y}$ for some positive semi-definite matrix $\mathbf{E}$. In this case, we can factor $\mathbf{E} = \mathbf{U}^\top\mathbf{U}$ in a pre-computation that is independent of n using a pivoted Cholesky decomposition using $\mathcal{O}(dr^2)$ operations for a rank r matrix, and let $\mathbf{u}(\mathbf{x}) = \mathbf{U}\mathbf{x}$, so that $\mathbf{J}[\mathbf{u}] = \mathbf{U}$. This gives rise to a Kronecker product structure in the Jacobian scaling matrix $\text{diag}(\mathbf{J}[\mathbf{u}](\mathbf{X})) = \mathbf{I}_n \otimes \mathbf{U}$, and enables subspace search techniques for BO, like the ones of Wang et al. (2013), Eriksson et al. (2018), and Kirschner et al. (2019), to take advantage of the structures proposed here. If $\mathbf{E}$ is diagonal as for automatic relevance determination (ARD), one can simply use $\mathbf{U} = \sqrt{\mathbf{E}}$, and the complexity of multiplying with $\mathbf{K}^\nabla$ is $\mathcal{O}(n^2d + nd)$. Notably, the matrix structure and its scaling also extend to complex warping functions $\mathbf{u}$, like Wilson et al. (2016)'s deep kernel learning model.

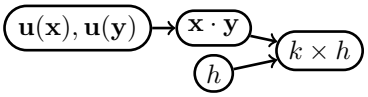
Composite Kernels Systematic application of the rules and data-sparse representations of $\mathbf{G}k$ for the transformations and compositions of kernels above gives rise to similar representations for many more complex kernels. Examples include the neural network kernel $\arcsin(\tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}})$, where $\tilde{\mathbf{x}} = \mathbf{x}/\sqrt{\|\mathbf{x}\|_2^2 + 1}$, the RBF-network kernel $\exp(-\|\mathbf{x}\|^2 - \mathbf{r}\cdot\mathbf{r}/2 - \|\mathbf{y}\|^2)$, the spectral mixture kernel of Wilson and Adams (2013), and the kernel $\boldsymbol{\phi}(\mathbf{x})^\top\mathbf{W}\boldsymbol{\phi}(\mathbf{y})h(\mathbf{x}, \mathbf{y})$ corresponding to a linear regression with variable coefficients, where $\boldsymbol{\phi}(\mathbf{x})$ are the regression features, $\mathbf{W}$ is the prior covariance of the weights, and h is a secondary kernel controlling the variability of the weights (Rasmussen and Williams, 2005). See Figure 1 for a depiction of these kernels' computational graphs, where each node represents a computation that we treated in this section. These examples highlight the generality of the proposed approach, since it applies *without specializations* to these kernels, and is simultaneously the first to enable a linear-in-d multiply with their gradient kernel matrices $\mathbf{K}^\nabla$.



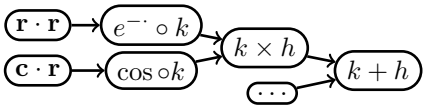
(a) Neural Network with $f(\mathbf{x}) = (\mathbf{x} \cdot \mathbf{x} + 1)^{-1/2}$



(b) RBF Network with $f(\mathbf{x}) = e^{-\mathbf{x} \cdot \mathbf{x}}$



(c) Variable Linear Regression



(d) Spectral Mixture

Figure 1: Computational graphs of composite kernels whose gradient kernel matrix can be expressed with the data-sparse structured expressions derived in Section 3.2. Inside a node, k and h refer to kernels computed by previous nodes.

3.3. Hessian Kernel Structure

Under appropriate differentiability assumptions (see Sec. A), we can condition a GP on Hessian information. However, incorporating second-order information into GPs has so far – except for one and two-dimensional test problems by Wu et al. (2017a) – not been explored. This is likely due to the prohibitive $\mathcal{O}(n^2d^4)$ scaling for a matrix multiply with the associated covariance matrix and $\mathcal{O}(n^3d^6)$ scaling for direct matrix inversion. In addition to the special structure for the gradient-Hessian cross-covariance, already reported by De Roos et al. (2021), we derive a structured representation of the Hessian-Hessian covariance for isotropic kernels, enabling efficient computations with second-order information. In particular, letting $\mathbf{h}_\mathbf{x} = \text{vec}(\mathbf{H}_\mathbf{x})$ where $\mathbf{H}_\mathbf{x}$ is the Hessian w.r.t. $\mathbf{x}$ and $r = \mathbf{r} \cdot \mathbf{r}$:

$$\begin{aligned}\mathbf{h}_\mathbf{x} \nabla_\mathbf{y}^\top k(\mathbf{x},\mathbf{y}) &= -f''(r)(\mathbf{I}_d \otimes \mathbf{r} + \mathbf{r} \otimes \mathbf{I}_d) \\ &\quad - [f''(r)\text{vec}(\mathbf{I}_d) + f'''(r)\text{vec}(\mathbf{r}\mathbf{r}^\top)]\mathbf{r}^\top, \text{ and} \\ \mathbf{h}_\mathbf{x}\mathbf{h}_\mathbf{y}^\top k(\mathbf{x},\mathbf{y}) &= (\mathbf{I}_{d^2} + \mathbf{S}_{dd})[f''(r)\mathbf{I}_{d^2} \\ &\quad + f'''(r)(\mathbf{r}\mathbf{r}^\top \oplus \mathbf{r}\mathbf{r}^\top)] + \mathbf{V}\mathbf{C}\mathbf{V}^\top,\end{aligned}$$

where $\mathbf{V} = \begin{bmatrix}\text{vec}(\mathbf{I}_d) & \text{vec}(\mathbf{r}\mathbf{r}^\top)\end{bmatrix} \in \mathbb{R}^{d^2 \times 2}$, $\mathbf{C} \in \mathbb{R}^{2\times 2}$ with $\mathbf{C}_{ij} = \partial^{(i+j)} f(r)$, $\mathbf{S}_{dd}$ is the "shuffle" matrix that satisfies $\mathbf{S}_{dd}\text{vec}(\mathbf{A}) = \text{vec}(\mathbf{A}^\top)$, and $\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{B}$ is the Kronecker sum. Thus, it is possible to multiply with covariance matrices that arise from conditioning on second-order information in $\mathcal{O}(n^2d^2)$, which is *linear* in the $\mathcal{O}(d^2)$ amount of information contained in the Hessian matrix and therefore optimal with respect to the dimensionality. This is an attractive complexity in moderate dimensionality since Hessian observations are highly informative of a function's local behavior. For derivations of the second-order covariances for more kernel types and transformations, see Sec. C.

3.4. An Implementation: `CovarianceFunctions.jl`

To take advantage of the analytical observations above in an *automatic* fashion, several technical challenges need to be overcome. First, we need a representation of the computational graph of a kernel function that is built from the basic constituents and transformations that we outlined above, akin to Figure 1. Second, we need to build matrix-free representations of the gradient kernel matrices to maintain data-sparse structure. Here, we briefly describe how we designed `CovarianceFunctions.jl`, an implementation of the structured AD technique that is enabled by the analytical derivations above, and supporting libraries, all written in Julia (Bezanson et al., 2017).

`CovarianceFunctions.jl` represents kernels at the level of user-defined types. It is in principle possible to hook into the abstract syntax tree (AST) to recognize these types of structures more generally (Innes, 2018), but this would undoubtedly come at the cost of increased complexity. It is unclear if this generality would have applications outside of the scope of this work. A user can readily extend the framework with a new kernel type if it can not already be expressed as a combinations and transformations of existing kernels. All that is necessary is the definition of its evaluation and the following short function: `input_trait` returns the type of input the kernel depends on: isotropic, dot-product, or the stationary type $\mathbf{c} \cdot \mathbf{r}$ and automatically detects homogeneous products and sums of kernels with these input types. As an example, for the rational quadratic kernel, we have

$$\texttt{input\_trait}(\texttt{RQ}_\alpha) = \texttt{Isotropic}()$$

Our implementation uses `ForwardDiff.jl` (Revels et al., 2016) to compute the regular derivatives and gradients that arise in the structured expressions to achieve a high level of generality and for a robust fall-back implementation of all the relevant operators in case no structure can be inferred in the input kernel. Even though the memory requirements of the n^2 data-sparse blocks are much reduced to the dense case, a machine can nevertheless run out of memory if the number of samples n gets very large and all blocks are stored in memory. To scale the method up to very large n, our implementation employs *lazy evaluation* of the gradient kernel matrix to achieve a constant, $\mathcal{O}(1)$, memory complexity for a matrix-vector multiply.

The main benefit of this system is that researchers and practitioners of BO *do not need to derive a special structured rep-*

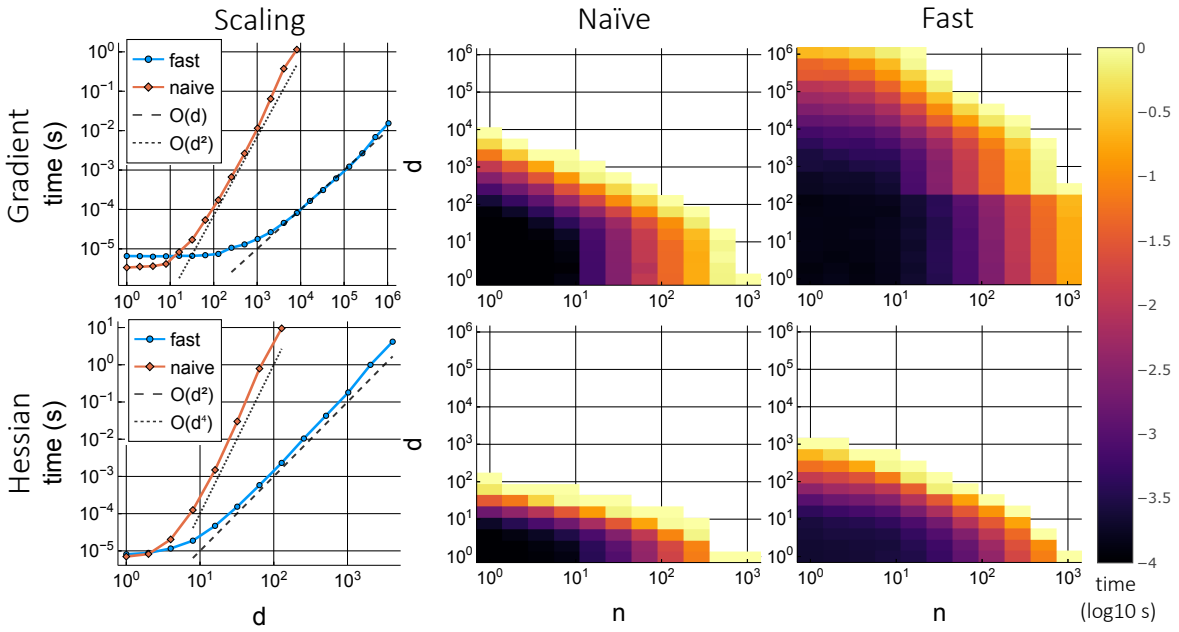


Figure 2: Benchmarks of matrix-vector-multiplications with the gradient (top) and Hessian kernel matrices (bottom) using a rational quadratic kernel. The scaling experiments (left) exhibit the predicted $\mathcal{O}(d)$ (resp. $\mathcal{O}(d^2)$) scaling of the fast algorithm for the gradient (resp. Hessian) kernel matrix with $n = 1$. The heat maps for the naïve (middle) and fast (right) algorithms show the execution time (color) as a function of n and d. The fast methods exhibit a much larger region of sub-second run-times in the n-d space, a proxy for the efficient applicability of the methods to problems of a particular size. Note that both axes are exponential; even the visually modest improvements for the Hessian allow between one to two orders of magnitude higher dimensionality than the naïve approach given the same run-time.

resentation for each kernel they want to use for an accurate modeling of their problem. As an example, the structured AD rules of Section 3.2 obviate the special derivation of the neural network kernel in Section B, In our view, this has the potential to greatly increase the uptake of first-order BO techniques outside of the immediate field of specialists.

4. Experiments

4.1. Scaling on Synthetic Data

First, we study the practical scaling of our implementation of the proposed methods with respect to both dimension and number of observations. See Figure 2 for experimental results using the non-separable rational quadratic kernel. Importantly, we observe virtually the same scaling behavior for more complex kernels like the neural network kernel. Further, we stress that the scaling results are virtually indistinguishable for different kernels, see also Figure 3 for similar experiments using the exponentiated dot-product and more complex neural network kernel.

The exponentiated dot-product kernel $\exp(\mathbf{x} \cdot \mathbf{y})$ was recently used by Karvonen et al. (2021) to derive a probabilistic Taylor-type expansion of multi-variate functions. The neural network kernel is derived as the limit of a neural network with one hidden layer, as the number of hidden units goes to infinity (Rasmussen and Williams, 2005). The scaling plots on the left of Figure 2 and 3 were created with a single thread to minimize constants, while the heat-maps on the right were run with 24 threads on 12 cores in parallel to highlight the applicability of the methods on a modern parallel architecture.

4.2. Comparison to Prior Work

Existing Libraries While popular libraries like GPyTorch, GPFlow, and Scikit-Learn have efficient implementations for the generic GP inference problem, they do *not* offer efficient inference *with gradient observations*, see Table 1 (Gardner et al., 2018a; De G. Matthews et al., 2017; Pedregosa et al., 2011). Highlighting the novelty of our work, GPyTorch contains only two implementations for this case – RBFKernelGrad and PolynomialKernelGrad – both with the naïve $\mathcal{O}(n^2d^2)$ matrix-vector multiplication complexity, hand-written work that is both obviated and outperformed by our structure-aware AD engine, see Figure 4. Thus, BoTorch, which depends on GPyTorch, does not yet support efficient FOBO. Neither GPFlow nor SKLearn contain any implementations of gradient kernels. Dragonfly and BayesOpt do not support gradient observations.

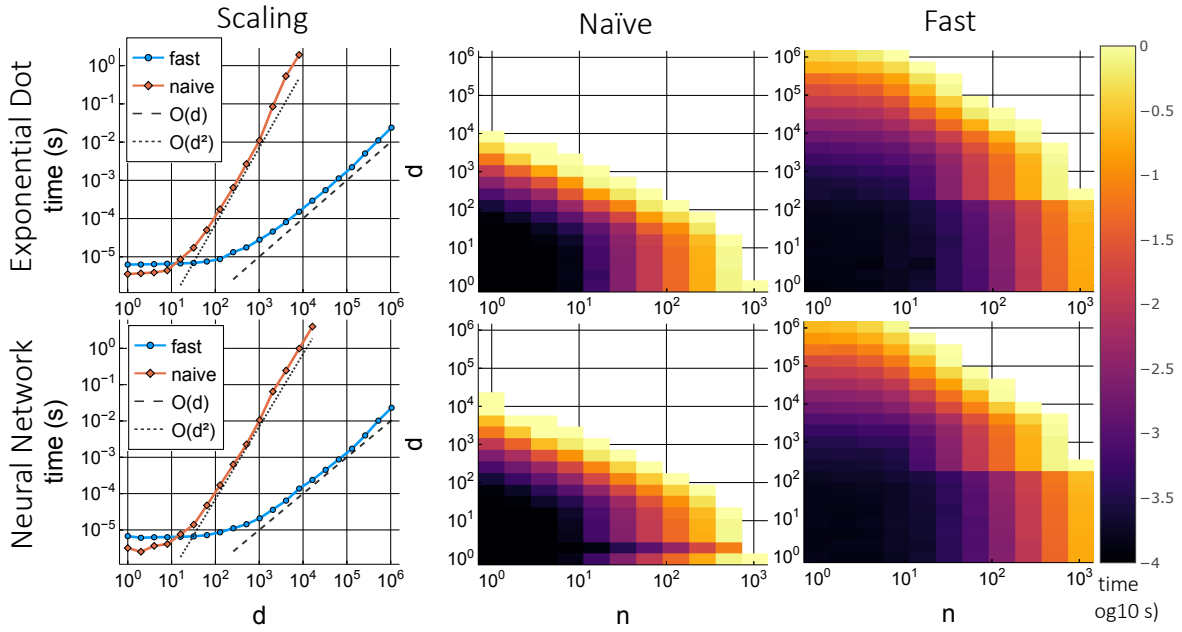


Figure 3: Benchmarks of matrix-vector multiplications with the gradient kernel matrices using the exponentiated dot-product (top) and neural network kernels (bottom). Compared to the performance of the rational quadratic kernel in Figure 2, the results for the composite neural network kernel are virtually indistinguishable and also exhibit the fast $\mathcal{O}(d)$ scaling.

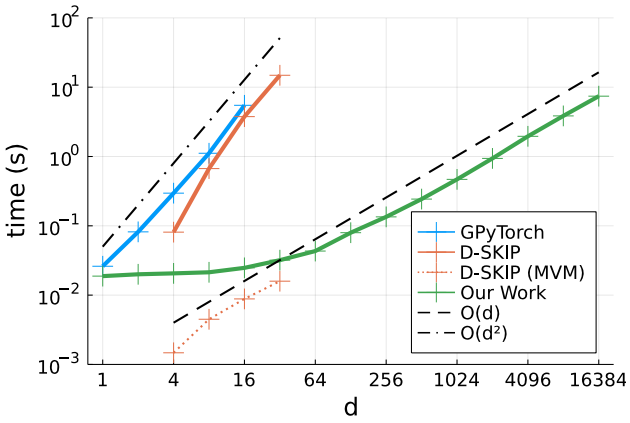


Figure 4: Time to first MVM of GPyTorch, D-SKIP, and our work for RBF gradient kernel matrices with $n = 1024$.

Table 1: MVM complexity with select gradient kernel matrices. SM = spectral mixture kernel, NN = neural network kernel. *See the discussion on the right about D-SKIP's complexity.

	RBF	SM	NN
GPFlow / SKLearn	✗	✗	✗
GPyTorch	$\mathcal{O}(n^2d^2)$	✗	✗
(Eriksson et al., 2018)	$\mathcal{O}(nd^2)^*$	✗	✗
(De Roos et al., 2021)	$\mathcal{O}(n^2d)$	✗	✗
Our work	$\mathcal{O}(n^2d)$	$\mathcal{O}(n^2d)$	$\mathcal{O}(n^2d)$

Eriksson et al. (2018)'s D-SKIP D-SKIP is an *approximate* method and requires that the kernel can be expressed as a separable product and further, that the resulting constituent kernel matrices have a low rank. In contrast our method is *mathematically exact* and applies to a large class of kernels without restriction. D-SKIP needs an upfront cost of $\mathcal{O}(d^2(n + m \log m + r^3 n \log d))$, followed by a matrix-vector multiplication (MVM) cost of $\mathcal{O}(dr^2n)$ for constituent kernel matrices of rank r and m inducing points per dimension. For a constant rank r, D-SKIP's MVM scales both linearly in n *and* d, while the method proposed herein scales quadratically in n. See Figure 4 for a comparison of D-SKIP's real-world performance, where D-SKIP's MVM scales linearly in d, but the required pre-processing scales quadratically in d and dominates the total runtime. Note that D-SKIP's implementation is restricted to $d > 4$, since D-SKI is faster in this regime. For $d \leq 32$, D-SKIP's pure MVM times are faster than our method, whose runtime grows sublinearly until $d = 64$ because it takes advantage of vector registers and SIMD instructions. Notably, the linear extrapolation of D-SKIP's pure MVM times without preprocessing is within a small factor (< 2) of the timings of our work for $d \geq 64$, implying that if D-SKIP were applied to higher dimensions, the pure MVM times of both methods would be comparable for a moderately large number of observations ($n = 1024$). Figure 6 in Section E shows that D-SKIP is approximate and looses accuracy as d increases, while our method is accurate to machine precision.

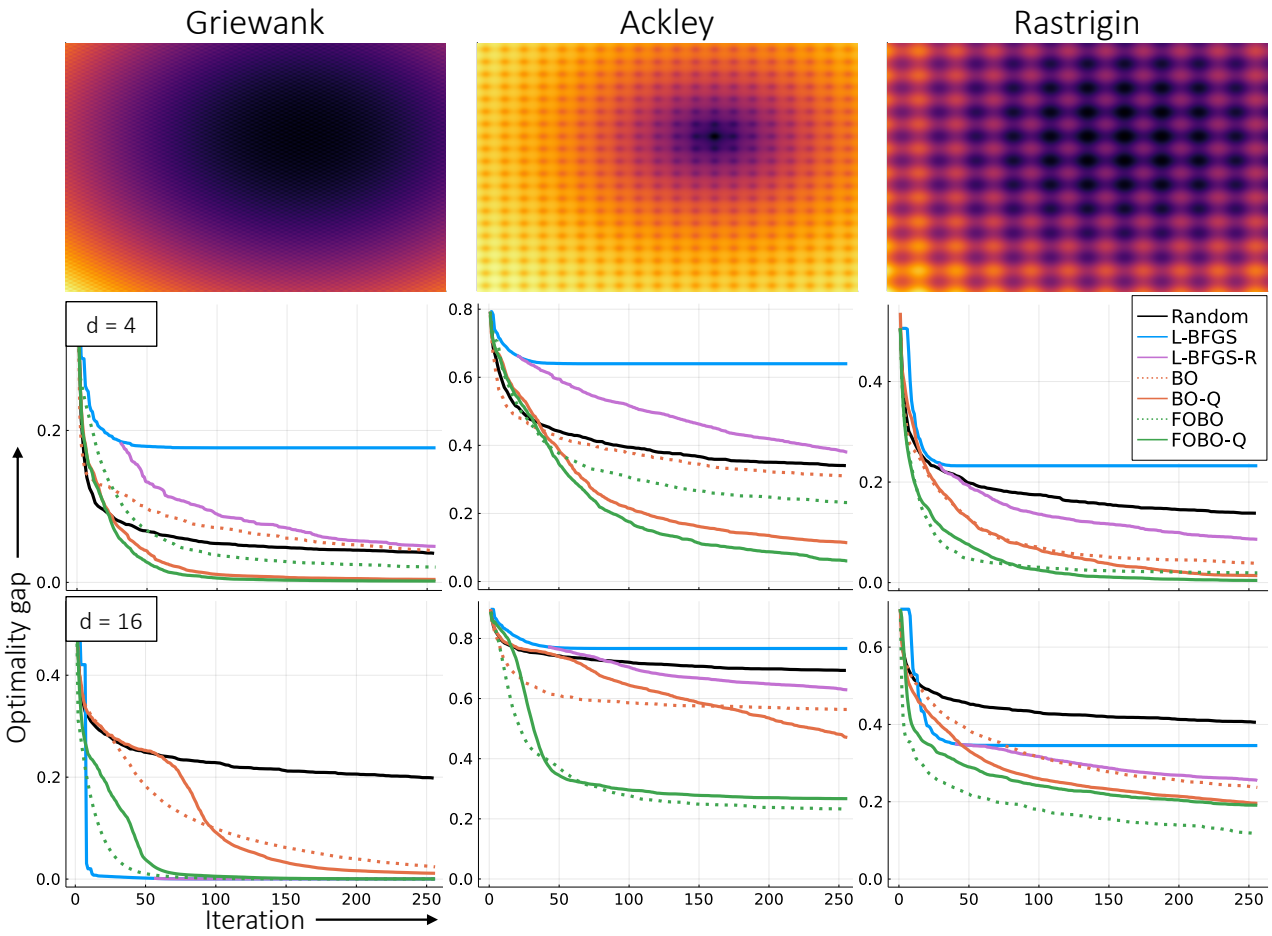


Figure 5: Average optimality gap of optimization algorithms on three non-convex test functions: Griewank, Ackley, and Rastrigin (plotted in 2D in top row). We compare random sampling (black), L-BFGS (blue), L-BFGS with random restarts after local convergence is detected (L-BFGS-R in purple), BO (dotted orange), BO with the quadratic mixture kernel of Section 4.3 (BO-Q in solid orange), FOBO (dotted green), and FOBO with the quadratic mixture kernel (FOBO-Q in solid green). All the BO variants use the expected improvement acquisition function and each line is the average optimality gap over 128 independent experiments. Notably, FOBO-Q outperforms on all 4D problems, L-BFGS converges most rapidly on the 16D Griewank function because its many local minima vanish as d increases so that the purely local search results in the fastest global convergence, and FOBO achieves the best optimality gap on the 16D Ackley and Rastrigin functions.

4.3. Bayesian Optimization

Shekhar and Javidi (2021) proved that gradient information can lead to an exponential improvement in regret for multi-armed bandit problems, compared to zeroth-order BO, by employing a two-stage procedure, the first of which hones in to a locally quadratic optimum of the objective. Inspired by this result and studying the qualitative appearance of many test functions of Bingham and Surjanovic (2013), a promising model for these objectives f is a sum of functions $f = g + h$, where g is a quadratic function and h is a potentially quickly varying, non-convex function. Since h is arbitrary, the model does not restrict the space of functions, but offers a useful inductive bias for problems with a globally quadratic structure "perturbed" by a non-convex function. Assuming the minimum of the quadratic function coincides or is close to a global minimum to the objective, this structure can be exploited to accelerate convergence.

Herein, we model g with a GP with the kernel $(\mathbf{x} \cdot \mathbf{y} + c)^2$, a distribution over quadratic functions whose stationary points are regularized by c, while h is assumed to be drawn from a GP with a Matérn-5/2 kernel k, to model quickly varying deviations from g. Then $f = g + h \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{y}) + (\mathbf{x} \cdot \mathbf{y} + c)^2)$. Notably, the resulting kernel is a composite kernel with isotropic and dot-product constituents, and a quadratic transformation. Without exploiting this structure automatically as proposed above, one would have to derive a new fast multiply by hand, slowing down the application of this model to BO.

Algorithm 1 Bayesian Optimization with Restarts

Input: acquisition a, objective f, prior $p \sim \mathcal{GP}$
Ouput: potential minimizer $\mathbf{x}^*$
initialize empty $\mathbf{X}$, $\mathbf{y}$
while budget not exhausted **do**
  $c \leftarrow p \mid \mathbf{X}, \mathbf{y}$ {compute conditional process}
  $\mathbf{x}_t \leftarrow \textit{local} \arg\min_{\mathbf{z}} a(c, \mathbf{z})$ {L-BFGS started at $\mathbf{x}_t$}
  if $\min_{\mathbf{z} \in \mathbf{X}} \|\mathbf{x}_t - \mathbf{z}\| < \epsilon$ **then**
    $\mathbf{x}_t \leftarrow$ random point in domain of f
  end if
  append $\mathbf{x}$ to $\mathbf{X}$ and $f(\mathbf{x}_t)$ to $\mathbf{y}$
end while
$\mathbf{x}^* \leftarrow \arg\min_{\mathbf{z} \in \mathbf{X}} f(\mathbf{z})$

Notably, this model is similar to the one employed by Gal et al. (2017), who used a quadratic function as the prior mean, requiring a separate optimization or marginalization of the location and covariance of the mean function. In contrast, we model the quadratic component with a specialized kernel, whose treatment only requires linear operations.

We benchmark both Bayesian and canonical optimization algorithms with and without gradient information on some of the test functions given by Bingham and Surjanovic (2013), namely, the Griewank, Ackley, and Rastrigin functions. See Section F for the definitions of the test functions. For all functions, we scaled the input domains to lie in $[-1, 1]^d$, scaled the output to lie in $[0, 1]$, and shifted the global optimum of all functions to $\mathbf{1}_d/4$. The top row of Figure 5 shows plots of the non-convex functions in two dimensions.

Figure 5 shows the average optimality gap over 128 independent experiments in four and sixteen dimensions for the following strategies: random sampling (black), L-BFGS (blue), L-BFGS with random restarts after local convergence is detected (L-BFGS-R in purple), BO (dotted orange), BO with the quadratic mixture kernel of Section 4.3 (BO-Q in solid orange), FOBO (dotted green), and FOBO with the quadratic mixture kernel (FOBO-Q in solid green). The FOBO variants incorporate both value and gradient observations, see Section D. All the BO variants use the expected improvement acquisition function which is numerically optimized w.r.t. the next observation point using L-BFGS. If the proposed next observation lies within 10^{-4} of any previously observed point, we choose a random point instead (see Algorithm 1), similar to the L-BFGS-R strategy. This helps escape local minima in the acquisition function and improves the performance of all BO algorithms.

FOBO-Q outperforms on all 4D problems, L-BFGS converges most rapidly on the 16D Griewank function because its many local minima vanish as d increases so that the purely local search results in the fastest convergence, and FOBO achieves the best optimality gap on the 16D Ackley and Rastrigin functions. While the Rastrigin function contains a quadratic component analytically, its inference appears to become more difficult as the dimension increases, leading FOBO to outperform the Q variant. But surprisingly, the Q variants outperform on the Ackley function for $d = 4$, even though it does *not* contain a quadratic component.

5. Conclusion

Limitations Incorporating gradient information improves the performance of BO, but the global optimization of non-convex functions remains *NP-hard* (see Section G) and can't be expected to be solved in general. For example, the optimum of the 16D Ackley function is elusive for all methods, likely because its domain of attraction shrinks exponentially with d. While we derived structured representations for kernel matrices arising from Hessian observations, we primarily focused on first-order information. We demonstrated the improved computational scaling and feasibility of computing with Hessian observations in our experiments but did not use this for BO. We leave a more comprehensive comparison of first and second-order BO to future work. Our main goal here was to enable such investigations in the first place, by providing the required theoretical advances, and practical infrastructure through `CovarianceFunctions.jl`.

Future Work 1) We are excited at the prospect of linking Maclaurin et al. (2015)'s algorithm for the computation of hyper-parameter gradients with the technology prosed here, *enabling efficient FOBO of hyper-parameters*. 2) While the methods proposed here are exact and enable a linear-in-d MVM complexity, $\mathcal{O}(n^2)$ can still become expensive with a large number of observations n. We believe that analysis-based fast algorithms like Ryan et al. (2022)'s Fast Kernel Transform could be derived for gradient kernels and hold promise in low dimensions. Further, BO trajectories can yield redundant information particularly when honing in on a minimum, which could be exploited using sparse linear solvers like the ones of Ament and Gomes (2021). 3) Hessian observations could be especially useful for Bayesian Quadrature, since the benefit of second-order information for integration is established: the Laplace approximation is used to estimate integrals in Bayesian statistics and relies on a single Hessian observation at the mode of the distribution.

Summary Bayesian Optimization has proven promising in numerous applications and is an active area of research. Herein, we provided exact methods with an $\mathcal{O}(n^2 d)$ MVM complexity for kernel matrices arising from n gradient observations in d dimensions and a large class of kernels, enabling first-order BO to scale to high dimensions. In addition, we derived structures that allow for an $\mathcal{O}(n^2 d^2)$ MVM with Hessian kernel matrices, making future investigations into second-order BO and Bayesian Quadrature possible.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Adler, R. J. (1981). *The Geometry of Random Fields*. Society for Industrial and Applied Mathematics.

Ahmed, M. O., Shahriari, B., and Schmidt, M. (2016). Do we need "harmless" bayesian optimization and "first-order" bayesian optimization. *NIPS BayesOpt*.

Ament, S. E. and Gomes, C. P. (2021). Sparse bayesian learning via stepwise regression. In *International Conference on Machine Learning*, pages 264–274. PMLR.

Attia, P. M., Grover, A., Jin, N., Severson, K. A., Markov, T. M., Liao, Y.-H., Chen, M. H., Cheong, B., Perkins, N., Yang, Z., et al. (2020). Closed-loop optimization of fast-charging protocols for batteries with machine learning. *Nature*, 578(7795):397–402.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2020). BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Bingham, D. and Surjanovic, S. (2013). Optimization test problems. http://www.sfu.ca/~ssurjano/optimization.html. Accessed: 2021-05-18.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Bull, A. D. (2011). Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(10).

Chaloner, K. and Verdinelli, I. (1995). Bayesian experimental design: A review. *Statistical Science*, pages 273–304.

Constantine, P. G., Dow, E., and Wang, Q. (2014). Active subspace methods in theory and practice: applications to kriging surfaces. *SIAM Journal on Scientific Computing*, 36(4):A1500–A1524.

De G. Matthews, A. G., Van Der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). Gpflow: A gaussian process library using tensorflow. *J. Mach. Learn. Res.*, 18(1):1299–1304.

De Roos, F., Gessner, A., and Hennig, P. (2021). High-dimensional gaussian process inference with derivatives. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2535–2545. PMLR.

Dong, K., Eriksson, D., Nickisch, H., Bindel, D., and Wilson, A. G. (2017). Scalable log determinants for gaussian process kernel learning. In *Advances in Neural Information Processing Systems*, pages 6327–6337.

Eriksson, D., Dong, K., Lee, E., Bindel, D., and Wilson, A. G. (2018). Scaling gaussian process regression with derivatives. In *Advances in Neural Information Processing Systems*, pages 6867–6877.

Eriksson, D., Pearce, M., Gardner, J., Turner, R. D., and Poloczek, M. (2019). Scalable global optimization via local bayesian optimization. In Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Foster, A., Jankowiak, M., Bingham, E., Horsfall, P., Teh, Y. W., Rainforth, T., and Goodman, N. (2019). Variational bayesian optimal experimental design. In Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Fu, Y., Zhu, X., and Li, B. (2013). A survey on instance selection for active learning. *Knowledge and information systems*, 35(2):249–283.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR.

Gal, Y. and Turner, R. (2015). Improving the gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs. In *International Conference on Machine Learning*, pages 655–664. PMLR.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018a). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586.

Gardner, J., Pleiss, G., Wu, R., Weinberger, K., and Wilson, A. (2018b). Product kernel interpolation for scalable gaussian processes. 84:1407–1416.

Griewank, A. and Walther, A. (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM.

Hennig, P. and Schuler, C. J. (2012). Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6).

Innes, M. (2018). Don't unroll adjoint: Differentiating ssa-form programs. *CoRR*, abs/1810.07951.

Innes, M., Barber, D., Besard, T., Bradbury, J., Churavy, V., Danisch, S., Edelman, A., Karpinski, S., Malmaud, J., Revels, J., Shah, V., Stenetorp, P., and Yuret, D. (2017). On machine learning and programming languages. https://julialang.org/blog/2017/12/ml-pl/. Accessed: 2021-05-18.

Kandasamy, K., Vysyaraju, K. R., Neiswanger, W., Paria, B., Collins, C. R., Schneider, J., Poczos, B., and Xing, E. P. (2020). Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21(81):1–27.

Karvonen, T., Cockayne, J., Tronarp, F., and Särkkä, S. (2021). A probabilistic taylor expansion with applications in filtering and differential equations. *CoRR*, abs/2102.00877.

Kirschner, J., Mutny, M., Hiller, N., Ischebeck, R., and Krause, A. (2019). Adaptive and safe bayesian optimization in high dimensions via one-dimensional subspaces. In *International Conference on Machine Learning*, pages 3429–3438. PMLR.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR.

Lázaro-Gredilla, M. (2012). Bayesian warped gaussian processes. *Advances in Neural Information Processing Systems*, 25:1619–1627.

Lázaro-Gredilla, M., Quinonero-Candela, J., Rasmussen, C. E., and Figueiras-Vidal, A. R. (2010). Sparse spectrum gaussian process regression. *The Journal of Machine Learning Research*, 11:1865–1881.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52.

Maclaurin, D., Duvenaud, D., and Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR.

Malkomes, G. and Garnett, R. (2018). Automating bayesian optimization with bayesian optimization. *Advances in Neural Information Processing Systems*, 31:5984–5994.

Marmin, S., Ginsbourger, D., Baccou, J., and Liandrat, J. (2018). Warped gaussian processes and derivative-based sequential designs for functions with heterogeneous variations. *SIAM/ASA Journal on Uncertainty Quantification*, 6(3):991–1018.

Martinez-Cantin, R. (2014). Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research*, 15(115):3915–3919.

Martinez-Cantin, R., Tee, K., and McCourt, M. (2018). Practical bayesian optimization in the presence of outliers. In *International Conference on Artificial Intelligence and Statistics*, pages 1722–1731. PMLR.

Mutny, M. and Krause, A. (2018). Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Paciorek, C. (2003). *Nonstationary Gaussian Processes for Regression and Spatial Modelling*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Padidar, M., Zhu, X., Huang, L., Gardner, J. R., and Bindel, D. (2021). Scaling gaussian processes with derivative information using variational inference. *arXiv preprint arXiv:2107.04061*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Powell, W. B. and Ryzhov, I. O. (2012). *Optimal learning*, volume 841. John Wiley & Sons.

Prabuchandran, K. J., Santosh, P., Chandramouli, K., and Shalabh, B. (2021). Novel first order bayesian optimization with an application to reinforcement learning. *Applied Intelligence*, 51:1–15.

Rahimi, A., Recht, B., et al. (2007). Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-mode automatic differentiation in Julia. *arXiv:1607.07892 [cs.MS]*.

Riihimäki, J. and Vehtari, A. (2010). Gaussian processes with monotonicity information. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 645–652. JMLR Workshop and Conference Proceedings.

Ryan, J. P., Ament, S. E., Gomes, C. P., and Damle, A. (2022). The fast kernel transform. In *International Conference on Artificial Intelligence and Statistics*, pages 11669–11690. PMLR.

Schoenberg, I. J. (1938). Metric spaces and completely monotone functions. *Annals of Mathematics*, 39(4):pp. 811–841.

Settles, B. (2009). Active learning literature survey.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Shekhar, S. and Javidi, T. (2021). Significance of gradient information in bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 2836–2844. PMLR.

Snelson, E., Rasmussen, C. E., and Ghahramani, Z. (2004). Warped gaussian processes. *Advances in neural information processing systems*, 16:337–344.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR.

Solak, E., Murray-Smith, R., Leithead, W. E., Leith, D. J., and Rasmussen, C. E. (2003). Derivative observations in gaussian process models of dynamic systems.

Solin, A., Kok, M., Wahlström, N., Schön, T. B., and Särkkä, S. (2018). Modeling and interpolation of the ambient magnetic field by gaussian processes. *IEEE Transactions on Robotics*, 34(4):1112–1127.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2012). Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265.

Törn, A. and Zilinskas, A. (1989). Global optimization.

Tuia, D., Volpi, M., Copa, L., Kanevski, M., and Munoz-Mari, J. (2011). A survey of active learning algorithms for supervised remote sensing image classification. *IEEE Journal of Selected Topics in Signal Processing*, 5(3):606–617.

Verma, A. (1998). Structured automatic differentiation. Technical report, Cornell University.

Wang, D. and Shang, Y. (2014). A new active labeling method for deep learning. In *2014 International joint conference on neural networks (IJCNN)*, pages 112–119. IEEE.

Wang, F., Decker, J., Wu, X., Essertel, G., and Rompf, T. (2018). Backpropagation with callbacks: Foundations for efficient and expressive differentiable programming. *Advances in Neural Information Processing Systems*, 31:10180–10191.

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q., and Wilson, A. G. (2019). Exact gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pages 14648–14659.

Wang, Z., Zoghi, M., Hutter, F., Matheson, D., De Freitas, N., et al. (2013). Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pages 1778–1784.

Wilson, A. and Adams, R. (2013). Gaussian process kernels for pattern discovery and extrapolation. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1067–1075, Atlanta, Georgia, USA. PMLR.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR.

Wu, A., Aoi, M. C., and Pillow, J. W. (2017a). Exploiting gradients and hessians in bayesian optimization and bayesian quadrature. *arXiv preprint arXiv:1704.00060*.

Wu, J. and Frazier, P. (2019). Practical two-step lookahead bayesian optimization. In Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Wu, J., Poloczek, M., Wilson, A. G., and Frazier, P. (2017b). Bayesian optimization with gradients. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zheng, S., Hayden, D., Pacheco, J., and Fisher III, J. W. (2020). Sequential bayesian experimental design with variable cost structure. *Advances in Neural Information Processing Systems*, 33.

A. Differentiability of Gaussian Processes

Summarized in (Paciorek, 2003), originally due to Adler.

Theorem A.1 (Mean-Square Differentiability, Adler (1981))**.** *A Gaussian process with covariance function k has a mean-square partial derivative at $\mathbf{z}$ if and only if $[\partial_{x_i}\partial_{y_i}k](\mathbf{z},\mathbf{z})$ exists and is finite.*

Proof. See proof of Theorem 2.2.2 in (Adler, 1981). □

Theorem A.2 (Sample-Path Continuity, Adler (1981))**.** *A stochastic process $z : \mathbb{R}^d \to \mathbb{R}$ is sample-path continuous if for $\eta > \alpha > 0$,*

$$E|z(\mathbf{x}+\mathbf{r}) - z(\mathbf{x})|^\alpha \leq \frac{c\|\mathbf{r}\|^{2d}}{|\log\|\mathbf{r}\||^{1+\eta}}.$$

Proof. See Corrollary of Theorem 3.2.5 in (Adler, 1981). □

Theorem A.3 (Sample-Path Continuity, Adler (1981))**.** *A Gaussian process $z : \mathbb{R}^d \to \mathbb{R}$ is sample-path continuous if for $\eta > \alpha > 0$,*

$$E|z(\mathbf{x}) - z(\mathbf{y})|^2 \leq \frac{c}{|\log\|\mathbf{x}-\mathbf{y}\||^{1+\eta}}.$$

Proof. See Corrollary of Theorem 3.2.5 in (Adler, 1981). □

The following result states that every positive-definite isotropic kernel can be expressed as a scale-mixture of Gaussian kernels, which aids the derivation of their differentiability properties. In particular,

Theorem A.4 (Schoenberg (1938))**.** *Suppose an isotropic kernel function k is positive-definite on a Hilbert space. Then there is a non-decreasing and bounded H such that*

$$k(\tau) = \int_0^\infty \exp(-\tau^2 s)\, dH(s). \tag{2}$$

We refer to s as the scale parameter.

Proof. This is due to Theorem 2 of Schoenberg (1938). □

Sample function (or almost sure) differentiability is a stronger property that requires a more subtle analysis. Paciorek (2003) provided the following result guaranteeing path differentiability for isotropic kernels.

Theorem A.5 (Sample-Path Differentiability, Paciorek (2003))**.** *Consider a Gaussian process with an isotropic covariance function k, and suppose $H(s)$ is related to k as in Theorem A.4. If the $2m$ moments $(E_H[s], ..., E_H[s^{2m}])$ of the scale parameter s are finite, the process is m times sample-path differentiable.*

Proof. This is essentially Theorem 10 in (Paciorek, 2003). □

Paciorek (2003) further uses this result to prove that the exponentiated quadratic and rational quadratic kernels are infinitely and Matérn kernels are finitely sample-path differentiable. A number of kernels like the exponential and the Matérn-1/2 kernels do not give rise to differentiable paths and can thus not be used in conjunction with gradient information. Notably, even the Matérn-3/2 kernel, which has a differentiable mean function, does not give rise to differentiable paths.

B. Explicit Derivation of Gradient Structure of Neural Network Kernel

The point of this section is to show an explicit derivation of the gradient structure of the neural network kernel, *which is obviated by the structure-deriving AD engine proposed in this work.*

Another interesting class of kernels are those that arise from analyzing the limit of certain neural network architectures as the number of hidden units tends to infinity. It is known that a number of neural networks converge to a Gaussian process under such a limit. For example, if the error function is chosen as the non-linearity for a neural network with one hidden layer, the kernel of the limiting process has the following form:

$$k_{NN}(\mathbf{x},\mathbf{y}) \stackrel{\text{def}}{=} \sin^{-1}\left(\frac{\mathbf{x}^\top\mathbf{y}}{\sqrt{n(\mathbf{x})n(\mathbf{y})}}\right),$$

where $n(\mathbf{x}) \stackrel{\text{def}}{=} 1 + \mathbf{x}^\top\mathbf{x}$. Formally, this is similar but not equivalent to the inner product kernels discussed above. The kernel gives rise to the following more complex gradient kernel structure

$$[\nabla_{\mathbf{x}}k](\mathbf{x},\mathbf{y}) = \tilde{f}'(r)\left(\mathbf{y} - \frac{r}{n(\mathbf{x})}\mathbf{x}\right),$$

where $\tilde{f}'(r) \stackrel{\text{def}}{=} f'(r)/\sqrt{n(\mathbf{x})n(\mathbf{y})}$, and

$$[\mathbf{G}k](\mathbf{x},\mathbf{y}) = \tilde{f}'(r)\mathbf{I}_d + \begin{bmatrix}\mathbf{x} & \mathbf{y}\end{bmatrix} \mathbf{A} \begin{bmatrix}\mathbf{x} & \mathbf{y}\end{bmatrix}^\top,$$

where

$$\mathbf{A} \stackrel{\text{def}}{=} \begin{bmatrix} -\frac{g(r)}{n(\mathbf{x})} & \frac{g(r)r}{n(\mathbf{x})n(\mathbf{y})} \\ \tilde{f}''(r) & -\frac{g(r)}{n(\mathbf{y})} \end{bmatrix}, \qquad \tilde{f}''(r) \stackrel{\text{def}}{=} \frac{f''(r)}{n(\mathbf{x})n(\mathbf{y})}, \qquad \text{and} \qquad g(r) \stackrel{\text{def}}{=} \left(\tilde{f}'(r) + \tilde{f}''(r)r\right).$$

Notably, this is a rank-two correction to the identity, compared to the rank-one corrections for isotropic and dot-product kernels above.

C. Hessian Structure

Note that for arbitrary vectors $\mathbf{a}, \mathbf{b}$, not necessarily of the same length, $\mathbf{a} \otimes \mathbf{b} = \text{vec}(\mathbf{b}\mathbf{a}^\top)$. This will come in handy to simplify certain expressions in the following.

Dot-Product Kernels First, note that

$$\nabla_{\mathbf{y}}^\top \text{vec}(\mathbf{y}\mathbf{y}^\top) = \mathbf{I}_d \otimes \mathbf{y} + \mathbf{y} \otimes \mathbf{I}_d \qquad \nabla_{\mathbf{y}}\nabla_{\mathbf{y}}^\top \text{vec}(\mathbf{y}\mathbf{y}^\top) = \mathbf{S}_{dd} + \mathbf{I}_{d^2}.$$

Where $\mathbf{S}_{dd}$ is a "shuffle" matrix such that $\mathbf{S}_{dd}\text{vec}(\mathbf{A}) = \text{vec}(\mathbf{A}^\top)$, and for square matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{m\times m}$, the Kronecker sum is defined as $\mathbf{A} \oplus \mathbf{B} \stackrel{\text{def}}{=} \mathbf{A} \otimes \mathbf{I}_m + \mathbf{I}_n \otimes \mathbf{B}$. Then for dot-product kernels, we have

$$[\mathbf{h}_{\mathbf{x}}k](\mathbf{x},\mathbf{y}) = f''(r)\text{vec}(\mathbf{y}\mathbf{y}^\top).$$

$$[\mathbf{h}_{\mathbf{x}}\nabla_{\mathbf{y}}^\top k](\mathbf{x},\mathbf{y}) = f''(r)(\mathbf{I}_d \otimes \mathbf{y} + \mathbf{y} \otimes \mathbf{I}_d) + f'''(r)\text{vec}(\mathbf{y}\mathbf{y}^\top)\mathbf{x}^\top.$$

$$[\mathbf{h}_{\mathbf{y}}^\top\mathbf{h}_{\mathbf{x}}k](\mathbf{x},\mathbf{y}) = (\mathbf{I}_{d^2} + \mathbf{S}_{dd})[f''(r)\mathbf{I}_{d^2} + f'''(r)(\mathbf{y}\mathbf{x}^\top \oplus \mathbf{y}\mathbf{x}^\top)] + f''''(r)\text{vec}(\mathbf{y}\mathbf{y}^\top)\text{vec}(\mathbf{x}\mathbf{x}^\top)^\top.$$

Isotropic Kernels Then for isotropic product kernels with $r = \|\mathbf{r}\|_2^2$, we have

$$\mathbf{J}_{\mathbf{x}}\text{vec}(\mathbf{r}\mathbf{r}^\top) = \mathbf{I}_d \otimes \mathbf{r} + \mathbf{r} \otimes \mathbf{I}_d \qquad \mathbf{H}_{\mathbf{y}}\text{vec}(\mathbf{r}\mathbf{r}^\top) = \mathbf{S}_{dd} + \mathbf{I}_{d^2}.$$

Which implies

$$[\mathbf{h}_{\mathbf{x}}k](\mathbf{x},\mathbf{y}) = f'(r)\text{vec}(\mathbf{I}_d) + f''(r)\text{vec}(\mathbf{r}\mathbf{r}^\top).$$

$$[\mathbf{h}_{\mathbf{x}}\nabla_{\mathbf{y}}^\top k](\mathbf{x},\mathbf{y}) = -f''(r)(\mathbf{I}_d \otimes \mathbf{r} + \mathbf{r} \otimes \mathbf{I}_d) - [f''(r)\text{vec}(\mathbf{I}_d) + f'''(r)\text{vec}(\mathbf{r}\mathbf{r}^\top)]\mathbf{r}^\top.$$

$$\begin{aligned}\mathbf{h}_{\mathbf{y}}^\top\mathbf{h}_{\mathbf{x}}k(\mathbf{x},\mathbf{y}) = (\mathbf{I}_{d^2} + \mathbf{S}_{dd})[f''(r)\mathbf{I}_{d^2} + f'''(r)(\mathbf{r}\mathbf{r}^\top \oplus \mathbf{r}\mathbf{r}^\top)] \\ + \begin{bmatrix}\text{vec}(\mathbf{I}_d) & \text{vec}(\mathbf{r}\mathbf{r}^\top)\end{bmatrix} \begin{bmatrix} f''(r) & f'''(r) \\ f'''(r) & f''''(r)\end{bmatrix} \begin{bmatrix}\text{vec}(\mathbf{I}_d) & \text{vec}(\mathbf{r}\mathbf{r}^\top)\end{bmatrix}^\top.\end{aligned}$$

A Chain Rule $k(\mathbf{x},\mathbf{y}) = (f \circ g)(\mathbf{x},\mathbf{y})$.

$$[\mathbf{h}_{\mathbf{x}}k](\mathbf{x},\mathbf{y}) = f'(r)\mathbf{h}_{\mathbf{x}}[g] + f''(r)\text{vec}(\nabla_{\mathbf{x}}g\nabla_{\mathbf{x}}g^\top).$$

$$[\mathbf{h}_{\mathbf{x}}\nabla_{\mathbf{y}}^\top k](\mathbf{x},\mathbf{y}) = f''(r)(\mathbf{H}_{\mathbf{x}}g \otimes \nabla_{\mathbf{y}}g + \nabla_{\mathbf{y}}g \otimes \mathbf{H}_{\mathbf{x}}g) + [f''(r)\mathbf{h}_{\mathbf{x}}[g]) + f'''(r)\text{vec}(\nabla_{\mathbf{x}}g\nabla_{\mathbf{x}}g^\top)]\nabla_{\mathbf{y}}g^\top.$$

$$\begin{aligned}\mathbf{h}_{\mathbf{x}}\mathbf{h}_{\mathbf{y}}^\top k(\mathbf{x},\mathbf{y}) = (\mathbf{I}_{d^2} + \mathbf{S}_{dd})[f''(r)\mathbf{I}_{d^2} + f'''(r)(\nabla_{\mathbf{x}}g\nabla_{\mathbf{x}}g^\top \oplus \nabla_{\mathbf{y}}g\nabla_{\mathbf{y}}g^\top)] \\ + \begin{bmatrix}\mathbf{h}_{\mathbf{x}}g & \text{vec}(\nabla_{\mathbf{x}}g\nabla_{\mathbf{x}}g^\top)\end{bmatrix} \begin{bmatrix} f''(r) & f'''(r) \\ f'''(r) & f''''(r)\end{bmatrix} \begin{bmatrix}\mathbf{h}_{\mathbf{y}}g & \text{vec}(\nabla_{\mathbf{y}}g\nabla_{\mathbf{y}}g^\top)\end{bmatrix}^\top.\end{aligned}$$

Vertical Scaling $k(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})h(\mathbf{x}, \mathbf{y})f(\mathbf{y})$ for a scalar-valued f, then

$$
\begin{aligned}
\mathbf{h}_\mathbf{x} k(\mathbf{x}, \mathbf{y}) =& \mathbf{h}_\mathbf{x}[f(\mathbf{x})h(\mathbf{x}, \mathbf{y})]f(\mathbf{y}) \\
=& [f(\mathbf{x})\mathbf{h}_\mathbf{x}[h](\mathbf{x}, \mathbf{y}) \\
&+ \mathbf{h}[f](\mathbf{x})h(\mathbf{x}, \mathbf{y}) \\
&+ \nabla_\mathbf{x}[h](\mathbf{x}, \mathbf{y}) \otimes \nabla[f](\mathbf{x}) \\
&+ \nabla[f](\mathbf{x}) \otimes \nabla_\mathbf{x}[h](\mathbf{x}, \mathbf{y})]\ f(\mathbf{y}) \\
[\mathbf{h}_\mathbf{x}\nabla_\mathbf{y}^\top k](\mathbf{x}, \mathbf{y}) =& [f(\mathbf{x})[\mathbf{h}_\mathbf{x}\nabla_\mathbf{y}^\top h](\mathbf{x}, \mathbf{y}) \\
&+ \mathbf{h}[f](\mathbf{x})[\nabla_\mathbf{y}^\top h](\mathbf{x}, \mathbf{y}) \\
&+ \mathbf{G}[h](\mathbf{x}, \mathbf{y}) \otimes \nabla[f](\mathbf{x}) \\
&+ \nabla[f](\mathbf{x}) \otimes \mathbf{G}[h](\mathbf{x}, \mathbf{y})]\ f(\mathbf{y}) \\
&+ \mathbf{h}_\mathbf{x}[f(\mathbf{x})h(\mathbf{x}, \mathbf{y})]\nabla_\mathbf{y}^\top\ f(\mathbf{y}) \\
[\mathbf{h}_\mathbf{x}\mathbf{h}_\mathbf{y}^\top k](\mathbf{x}, \mathbf{y}) =& [f(\mathbf{x})[\mathbf{h}_\mathbf{x}\mathbf{h}_\mathbf{y}^\top h](\mathbf{x}, \mathbf{y}) \\
&+ \mathbf{h}[f](\mathbf{x})[\mathbf{h}_\mathbf{y}^\top h](\mathbf{x}, \mathbf{y}) \\
&+ \mathbf{G}[h](\mathbf{x}, \mathbf{y}) \otimes \nabla[f](\mathbf{x})\nabla^\top[f](\mathbf{y}) \\
&+ \nabla[f](\mathbf{x})\nabla^\top[f](\mathbf{y}) \otimes \mathbf{G}[h](\mathbf{x}, \mathbf{y})]\ f(\mathbf{y}) \\
&+ \mathbf{h}_\mathbf{x}[f(\mathbf{x})h(\mathbf{x}, \mathbf{y})]\mathbf{h}_\mathbf{y}^\top\ f(\mathbf{y})
\end{aligned}
$$

Again, we observe a structured representation of the Hessian-kernel elements which permit a multiply in $\mathcal{O}(d^2)$ operations.

Warping $k(\mathbf{x}, \mathbf{y}) = h(\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{y}))$,

$$
\begin{aligned}
\mathbf{h}_\mathbf{x} k(\mathbf{x}, \mathbf{y}) &= (\mathbf{J} \otimes \mathbf{J})^\top[\mathbf{u}](\mathbf{x})\ [\mathbf{h}_\mathbf{x} h](\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{y})) \\
[\mathbf{h}_\mathbf{x}\nabla_\mathbf{y}^\top k](\mathbf{x}, \mathbf{y}) &= (\mathbf{J} \otimes \mathbf{J})^\top[\mathbf{u}](\mathbf{x})\ [\mathbf{h}_\mathbf{x}\nabla_\mathbf{y}^\top h](\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{y}))\ \mathbf{J}[\mathbf{u}](\mathbf{y}) \\
[\mathbf{h}_\mathbf{x}\mathbf{h}_\mathbf{y}^\top k](\mathbf{x}, \mathbf{y}) &= (\mathbf{J} \otimes \mathbf{J})^\top[\mathbf{u}](\mathbf{x})\ [\mathbf{h}_\mathbf{x}\mathbf{h}_\mathbf{y}^\top h](\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{y}))\ (\mathbf{J} \otimes \mathbf{J})[\mathbf{u}](\mathbf{y}).
\end{aligned}
$$

We therefore see that $\mathbf{K}^\mathbf{H} = \mathbf{h}_\mathbf{x}\mathbf{h}_\mathbf{y}^\top k(\mathbf{X}) = \mathbf{D}_\mathbf{J}[\mathbf{h}_\mathbf{x}\mathbf{h}_\mathbf{y}^\top h](\mathbf{X})\mathbf{D}_\mathbf{J}$, where $\mathbf{D}_\mathbf{J}$ is the block-diagonal matrix whose i^{th} block is equal to $(\mathbf{J} \otimes \mathbf{J})[\mathbf{u}](\mathbf{x}_i) = \mathbf{J}[\mathbf{u}](\mathbf{x}_i) \otimes \mathbf{J}[\mathbf{u}](\mathbf{x}_i)$. Note that for linearly warped kernels for which $\mathbf{u}(\mathbf{x}) = \mathbf{U}\mathbf{x}$, where $\mathbf{U} \in \mathbb{R}^{r \times d}$, we have $(\mathbf{J} \otimes \mathbf{J})[\mathbf{u}](\mathbf{x}_i) = \mathbf{U} \otimes \mathbf{U}$ so that we can multiply with the kernel matrix $\mathbf{K}^\mathbf{H}$ in $\mathcal{O}(n^2r^2 + n(d^2r + r^2d))$. The complexity is due to the following property of Kronecker product:

$$(\mathbf{U} \otimes \mathbf{U})\text{vec}(\mathbf{H}) = \text{vec}(\mathbf{U}\mathbf{H}\mathbf{U}^\top),$$

which can be computed in $\mathcal{O}(d^2r + r^2d)$ for every of the n Hessian observations.

D. Combining Derivative Orders

Combining observations of the function values and its first and second derivatives is straightforward via the following block-structured kernel:

$$
\begin{bmatrix}
k & \nabla_\mathbf{y}[k]^\top & \mathbf{h}_\mathbf{y}[k]^\top \\
\nabla_\mathbf{x}[k] & \mathbf{G}[k] & \mathbf{J}_\mathbf{x}[\mathbf{h}_\mathbf{y}[k]] \\
\mathbf{h}_\mathbf{x}[k] & \mathbf{J}_\mathbf{y}[\mathbf{h}_\mathbf{x}[k]] & \mathbf{H}[k]
\end{bmatrix}.
$$

If all constituent blocks permit a fast multiply - $\mathcal{O}(d)$ for gradient and $\mathcal{O}(d^2)$ for Hessian-related blocks - the entire structure permits a $\mathcal{O}(d^2)$ multiply, even though the naïve cost is $\mathcal{O}(d^4)$. If only value and gradient observations are required, only the top-left two-by-two block is necessary, which can be carried out in $\mathcal{O}(d)$ in the structured case and which we implemented as the `ValueGradientKernel`.

Discussion Recall that the computational complexity of multiplying with the gradient and Hessian kernel matrices is $\mathcal{O}(n^2d)$ and $\mathcal{O}(n^2d^2)$, respectively. Thus, the gradient-based method can only make a factor of $\mathcal{O}(\sqrt{d})$ more observations than the Hessian-based method for the same computational cost. Therefore, it is computationally easier to incorporate additional information at a single point than it is to combine first-order information at several points. Since the Hessian contains d times more pieces of information than the gradient, the former could be more efficient in certain scenarios. We compare the scaling of the multiplications arising from first- and second-order data experimentally in Section 4.1 but leave a comprehensive comparison of first-order and second-order BO to future work. The main goal of the current work is to enable such investigations in the first place by providing the required theoretical advances and practical infrastructure, see `CovarianceFunctions.jl`.

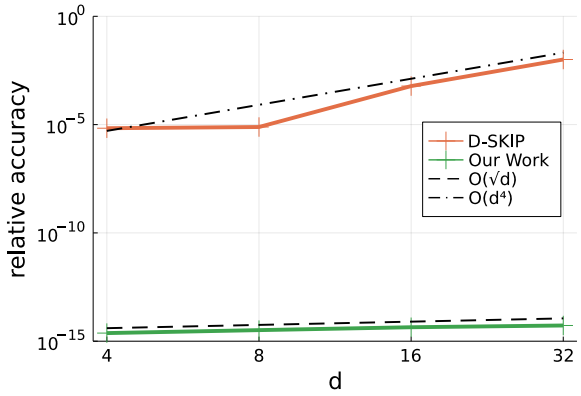


Figure 6: Relative accuracy of MVMs using D-SKIP and our work for RBF gradient kernel matrices with $n = 1024$.

E. Accuracy Comparison to D-SKIP

Figure 6 shows the relative accuracy of MVMs using D-SKIP and our work for RBF gradient kernel matrices with $n = 1024$. Note that the indicated asymptotic lines $\mathcal{O}(\sqrt{d})$ and $\mathcal{O}(d^4)$ are there for comparison only and do not indicate a theoretically expected error scaling.

F. Test Functions for Bayesian Optimization

Rosenbrock

$$\text{Rosenbrock}_d(\mathbf{x}) = \sum_{i}^{d-1} (x_i - a)^2 + b(x_{i+1} - x_i^2)^2$$

For our experiments, we let $a = 0$, $b = 10$ and evaluate it on $x_i \in [-3, 3]$.

Ackley The Ackley function in d dimensions is given by

$$\text{Ackley}_d(\mathbf{x}) = \exp(1) + a - a \exp\left(-b\|\mathbf{x}\|_2/\sqrt{d}\right) - \exp\left(\sum_{i=1}^{d} \cos(cx_i)/d\right),$$

where $a = 20, b = 0.2$, and $c = 2\pi$. The function is usually evaluated on the hypercube $x_i \in [-32.768, 32.768]$ and has a single global minimum at the origin. We evaluated it on $[-10, 10]$ for our experiments.

Rastrigin The Rastrigin function is defined by

$$\text{Rastrigin}_d(\mathbf{x}) = 10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)],$$

is usually evaluated on the hypercube $x_i \in [-5.12, 5.12]$ and has a single global minimum of 0 at the origin.

Drop-Wave The Drop-Wave function is defined by

$$\text{DropWave}_d(\mathbf{x}) = -\frac{1 + \cos(12\|\mathbf{x}\|_2)}{\|\mathbf{x}\|_2^2 + 2},$$

is usually evaluated on the two-dimensional square $x_i \in [-5.12, 5.12]$ and has a single global minimum of -1 at the origin. We note however, that it is straightforwardly generalized to arbitrary input dimensions, since it only depends on the Euclidean norm of the input.

Griewank The Griewank function is defined by

$$\text{Griewank}_d(\mathbf{x}) = \|\mathbf{x}\|_2^2/4000 - \prod_i \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

is usually evaluated on the two-dimensional square $x_i \in [-600, 600]$ and has a single global minimum of 0 at the origin. We note however, that it is straightforwardly generalized to arbitrary input dimensions, since it only dependents on the Euclidean norm of the input. We evaluated it on $x_i \in [-200, 200]$ in our experiments.

G. Theoretical Background for Global and Bayesian Optimization

Global Optimization Törn and Zilinskas (1989) noted that the global approximation and optimization of continuous functions is a hard problem in general. They proved that, for any algorithm, there are multimodal functions that give rise to an arbitrarily large error after a finite number of samples, and that a global optimization algorithm converges to the optimum of any continuous function on a compact set, *if and only if it eventually samples the set densely*.

If one imposes certain structure on the set of functions, finite-sample guarantees become feasible. For example, function whose modulus of continuity $\omega(\delta) = \max_{d(x,y)<\delta} |f(x) - f(y)|$ is bounded, like Lipschitz-continuous functions, admit the following finite-sample bound. Let A be the compact set on which the function f is to be minimized and $\{x_i\} \subset A$ be the observed samples, then

$$\min_{1\leq i\leq n} f(x_i) - \min_{x\in A} f(x) < \omega(d_n),$$

where d is the metric on A and $d_n = \max_{x\in A} \min_{1\leq i\leq n} d(x, x_i)$ is the dispersion of the samples. By extension, this implies that continuously differentiable functions on a compact set admit a similar bound, since the suprema of their derivatives are attained and finite.

Bayesian Optimization Regarding Bayesian optimization algorithms, Törn and Zilinskas (1989) noted that "even if [BO] is very attractive theoretically it is too complicated for algorithmic realization. Because of the fairly cumbersome computations involving operations with the inverse of the *covariance matrix* and complicated auxiliary optimization problems. The resort has been to use simplified models." The techniques put forth in Section 3 of this article make significant strides in reducing this complexity. Bull (2011) provides theoretical results for the convergence of Bayesian optimization algorithms based on Gaussian processes and the Expected Improvement (EI) acquisition function. The results hold for noiseless observations of the function to be optimized. Srinivas et al. (2012) proved regret bounds for the multi-armed bandit problem for which the payoff function is drawn from a GP with particular kernel functions and the upper-confidence bound acquisition function. Shekhar and Javidi (2021) recently proved that zeroth-order BO has a regret lower bound that increases exponentially with the dimension, while first-order BO can achieve a much better regret bound of $\mathcal{O}(d \log^2 n)$, where d is the dimensionality of the input and n is the number of observations, using a two-stage procedure: the first stage identifies a locally quadratic neighborhood around a presumed optimum, while the second stage takes local gradient steps based on stochastic estimates of the gradient.