High-Dimensional Gaussian Process Inference with Derivatives

Filip de Roos [1 2]   **Alexandra Gessner** [1 2]   **Philipp Hennig** [1 2]

Abstract

Although it is widely known that Gaussian processes can be conditioned on observations of the gradient, this functionality is of limited use due to the prohibitive computational cost of $\mathcal{O}(N^3 D^3)$ in data points N and dimension D. The dilemma of gradient observations is that a single one of them comes at the same cost as D independent function evaluations, so the latter are often preferred. Careful scrutiny reveals, however, that derivative observations give rise to highly structured kernel Gram matrices for very general classes of kernels (inter alia, stationary kernels). We show that in the *low-data* regime $N < D$, the Gram matrix can be decomposed in a manner that reduces the cost of inference to $\mathcal{O}(N^2 D + (N^2)^3)$ (i.e., linear in the number of dimensions) and, in special cases, to $\mathcal{O}(N^2 D + N^3)$. This reduction in complexity opens up new use-cases for inference with gradients especially in the high-dimensional regime, where the information-to-cost ratio of gradient observations significantly increases. We demonstrate this potential in a variety of tasks relevant for machine learning, such as optimization and Hamiltonian Monte Carlo with predictive gradients.

1. Introduction

The closure of Gaussian processes (GPs) under linear operations is well-established (Rasmussen & Williams, 2006, Ch. 9.4). Given a Gaussian process $f \sim \mathcal{GP}(\mu, k)$, with mean and covariance function μ and k, respectively, a linear operator $\mathcal{L}$ acting on f induces another Gaussian process $\mathcal{L}f \sim \mathcal{GP}(\mathcal{L}\mu, \mathcal{L}k\mathcal{L}')$ for the operator $\mathcal{L}$ and its adjoint $\mathcal{L}'$. The linearity of GPs has found extensive use both for conditioning on projected data, and to perform inference on linear transformations of f. Differentiation is a linear operation

which has caused considerable interest in GP modeling due to the wide variety of applications in which derivative observations are available. However, each gradient observation of $\nabla f \in \mathbb{R}^D$ induces a block Gram matrix $\nabla k \nabla' \in \mathbb{R}^{D \times D}$. As dimension D and number of observations N grow, inference with gradient information quickly becomes prohibitive with the naïve scaling of $\mathcal{O}(N^3 D^3)$. In other words, one gradient observation comes at the same computational cost as D independent function evaluations and thus becomes increasingly disadvantageous as dimensionality grows. This is not surprising, as the gradient contains D elements and thus bears information about every coordinate. The unfavorable scaling has confined GP inference with derivatives to low-dimensional settings in which the information gained from gradients outweighs the computational overhead (cf. Sec. 3 for a review).

In this work we show that gradient Gram matrices possess structure that enables inversion at cost linear in D. This discovery unlocks the previously prohibitive use of gradient evaluations in high dimensional spaces for nonparametric models. Numerous machine learning algorithms that operate on high-dimensional spaces are guided by gradient information and bear the potential to benefit from an inference mechanism that avoids discarding readily available information. Examples for such applications that we consider in this work comprise optimization, linear algebra, and gradient-informed Markov chain Monte Carlo.

Contributions   We analyze the structure of the Gram matrix with derivative observations for stationary and dot product kernels and report the following discoveries[1]:

- The Gram matrix can be decomposed to allow *exact* inference in $\mathcal{O}(N^2 D + (N^2)^3)$ floating point operations, which is useful in the limit of few observations ($N < D$).

- We introduce an efficient approximate inference scheme to include gradient observations in GPs even as the number of high-dimensional observations increases. It relies on exact matrix-vector multiplication (MVM) and an iterative solver to approximately invert the Gram matrix. This implicit MVM avoids

---

[1]Department of Computer Science, University of Tübingen, Tübingen, Germany [2]Max Planck Institute for Intelligent Systems, Tübingen, Germany. Correspondence to: Filip de Roos <filip.de.roos@tuebingen.mpg.de>.

---

[1]Code repository: https://github.com/fidero/gp-derivative

constructing the whole Gram matrix and thereby reduces the memory requirements from $\mathcal{O}((ND)^2)$ to $\mathcal{O}(N^2 + ND)$.

- We demonstrate the applicability of the improved scaling in the low-data regime on high-dimensional applications ranging from optimization to Hamiltonian Monte Carlo.

- We explore a special case of inference with application to probabilistic linear algebra for which the cost of inference can be further reduced to $\mathcal{O}(N^2 D + N^3)$.

2. Theory

Kernel matrices of Gaussian processes (GPs) built from gradient observations are highly structured. In this section, after reviewing GPs, we show that for standard kernels, the kernel Gram matrix can be decomposed into a Kronecker product with an additive low-rank correction, as exemplified in Fig. 1. Exploiting this structure, *exact* GP inference with gradients is feasible in $\mathcal{O}(N^2 D + (N^2)^3)$ operations instead of $\mathcal{O}((DN)^3)$ when inverting the kernel matrix exactly. Furthermore, the same structure enables storage of $\mathcal{O}(N^2 + ND)$ values instead of $\mathcal{O}((ND)^2)$.

2.1. Gaussian Processes

Definition 1. *A Gaussian process $f \sim \mathcal{GP}(\mu, k)$ is a random process with mean function $\mu : \mathbb{R}^D \mapsto \mathbb{R}$ and covariance function $k : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$ such that f evaluated at a finite set of inputs follow a multi-variate normal distribution (Rasmussen & Williams, 2006, Ch. 2.2).*

GPs are popular nonparametric models with numerous favorable properties, of which we highlight their closure under linear operations. A linear operator acting on a GP results again in a GP. Let $\mathcal{L}, \mathcal{M}$ be linear operators acting on f. Then the joint distribution of $\mathcal{L}f$ and $\mathcal{M}f$ is:

$$\begin{bmatrix} \mathcal{L}f \\ \mathcal{M}f \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} \mathcal{L}\mu \\ \mathcal{M}\mu \end{bmatrix}, \begin{bmatrix} \mathcal{L}k\mathcal{L}' & \mathcal{L}k\mathcal{M}' \\ \mathcal{M}k\mathcal{L}' & \mathcal{M}k\mathcal{M}' \end{bmatrix} \right), \quad (1)$$

where $\mathcal{L}'$ and $\mathcal{M}'$ act on the second argument of the covariance function k. The conditional $\mathcal{L}f \mid \mathcal{M}f$ is obtained with standard Gaussian algebra and requires the inversion of $\mathcal{M}k\mathcal{M}'$.

Examples of linear operators comprise projections, integration, and differentiation. We focus here on inference on either f itself, its gradient $\boldsymbol{g} = \nabla f$, or its Hessian matrix $H = \nabla\nabla^\top f$ conditioned on gradient observations, i.e. $\mathcal{L} = \{\mathrm{Id}, \nabla, \nabla\nabla^\top\}$ and $\mathcal{M} = \nabla$.

Notation We collect gradient observations $\boldsymbol{g}_a \in \mathbb{R}^D$ at locations $\boldsymbol{x}_a \in \mathbb{R}^D$, $a = 1, \ldots, N$ which we vertically stack into the data matrices $X \in \mathbb{R}^{D \times N}$ and $G \in \mathbb{R}^{D \times N}$. The object of interest is the Gram matrix $\nabla K \nabla' \in \mathbb{R}^{DN \times DN}$ where $K = k(X, X)$ and ∇, ∇' act w.r.t. all elements of X. We let subscripts a, b identify indices related to *data points*, e.g., $\boldsymbol{x}_a, \boldsymbol{x}_b$. Superscript indices i, j refer to indices along the input *dimension*. In further abuse of notation we will let the operation $\tilde{X} = X - \boldsymbol{c}$ denote the subtraction of $\boldsymbol{c}$ from each column in X.

2.2. Exploiting Kernel Structure

The efficient inversion of $\nabla K \nabla'$ relies on its somewhat repetitive structure involving a Kronecker product (Fig. 1) caused by application of the product and chain rule of differentiation to the kernel. The Kronecker product $A \otimes B$ produces a matrix with blocks $a_{ij} B$ (cf. Van Loan (2000) and Appendix A for properties of the Kronecker product). Any kernel $k(\boldsymbol{x}_a, \boldsymbol{x}_b)$ with inputs $\boldsymbol{x}_a, \boldsymbol{x}_b \in \mathbb{R}^D$, can be equivalently written on terms of a scalar function r : $\mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$ as $k(r(\boldsymbol{x}_a, \boldsymbol{x}_b)) =: k_{ab}(r)$. Note the general definition of r, which in particular is more general than stationarity. Since k is also a scalar function of $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, r could be equal to k if there is no way to further condense the relationship between $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$.

Definition 2. *Write $k(\boldsymbol{x}_a, \boldsymbol{x}_b) = k_{ab}(r)$ with $r \in \mathbb{R}$. Define $k'_{ab} = \frac{\partial k(\boldsymbol{x}_a, \boldsymbol{x}_b)}{\partial r}$, $k''_{ab} = \frac{\partial^2 k(\boldsymbol{x}_a, \boldsymbol{x}_b)}{\partial r^2}$ and $\partial_a{}^i = \frac{\partial}{\partial \boldsymbol{x}_a{}^i}$ and similarly for $\partial_b{}^j$. The derivatives of k w.r.t. $\partial \boldsymbol{x}_a{}^i$ and $\partial \boldsymbol{x}_b{}^j$ can be written as*

$$\partial_a{}^i k_{ab}(r) = k'_{ab}(r)\, \partial_a{}^i r$$
$$\partial_b{}^j k_{ab}(r) = k'_{ab}(r)\, \partial_b{}^j r$$
$$\partial_a{}^i \partial_b{}^j k(r) = k'_{ab}(r) \cdot \partial_a{}^i \partial_b{}^j r + k''_{ab}(r) \cdot (\partial_a{}^i r)(\partial_b{}^j r)$$

$$(2)$$

This expression is still general but we can already see that the derivatives of k w.r.t. r depend only on the indices a, b of the data points and form $N \times N$ matrices that we call K' and K''. Importantly, they do *not* depend on the dimensional indices i, j. While the abundance of indices invites for a tensor-like implementation, the prime gain comes from writing Eq. (2) in matrix form. Doing so permits linear algebra operations that are not applicable to tensors. We specify the matrix form of the general expression of Eq. (2) for two overarching classes of kernels: the dot product and stationary class of kernels. For these kernels, r is defined as

$$r = (\boldsymbol{x}_a - \boldsymbol{c})^\top \Lambda (\boldsymbol{x}_b - \boldsymbol{c}) \quad \text{(dot product kernels)},$$
$$r = (\boldsymbol{x}_a - \boldsymbol{x}_b)^\top \Lambda (\boldsymbol{x}_a - \boldsymbol{x}_b) \quad \text{(stationary kernels)},$$

with an arbitrary offset $\boldsymbol{c}$ and a symmetric positive definite scaling matrix Λ.

Dot Product Kernels For dot product kernels the Gram matrix of the gradients in Eq. (2) is

$$\partial_a{}^i \partial_b{}^j k(r) = k'_{ab}(r) \cdot \Lambda^{ij} + k''_{ab}(r) \cdot [\Lambda(\boldsymbol{x}_b - \boldsymbol{c})]^i [\Lambda(\boldsymbol{x}_a - \boldsymbol{c})]^j,$$
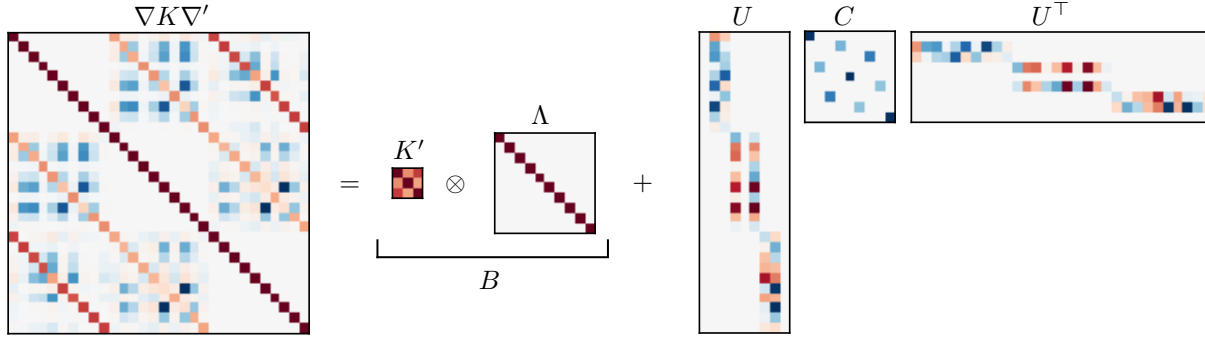
Figure 1. Gram matrix built from three 10-dimensional gradient observations using a stationary isotropic exponential quadratic kernel. Explicit expression *(left)* and its decomposition into a Kronecker product B and low-rank correction UCU^T that allows for efficient inversion using Woodbury's matrix lemma (cf. Sec. 2.3) if $N < D$ *(right)*. Positive values are colored red, negative blue and white indicates 0.

which can be written as a low-rank update to a Kronecker products as (see Appendix B.2 for the derivation)

$$K' \otimes \Lambda + (I \otimes \Lambda(X - \boldsymbol{c})) \, C \, (I \otimes (X - \boldsymbol{c})^\top \Lambda), \quad (3)$$

where $U = I \otimes \Lambda(X - \boldsymbol{c})$ is of size $DN \times N^2$. The matrix $C \in \mathbb{R}^{N^2 \times N^2}$ is a permutation of $\mathrm{diag}\left(\mathrm{vec}(K'')\right)$, i.e. a diagonal matrix that has the elements of K'' on its diagonal, such that $C \, \mathrm{vec}(M) = \mathrm{vec}(K'' \odot M^\top)$, for $M \in \mathbb{R}^{N \times N}$. Here, $\odot$ denotes the Hadamard (element-wise) product.

Stationary kernels The gradient structure of Eq. (2) for stationary kernels looks similar to the dot product case:

$$\partial_a{}^i \partial_b{}^j k(r) = \\ - k'_{ab}(r) \cdot \Lambda^{ij} - k''_{ab}(r) \cdot [\Lambda(\boldsymbol{x}_a - \boldsymbol{x}_b)]^i [\Lambda(\boldsymbol{x}_a - \boldsymbol{x}_b)]^j. \\ (4)$$

Similarly to the dot product kernel, this also takes the matrix form

$$K' \otimes \Lambda + U \, C \, U^\top, \quad (5)$$

where the first term $K' \otimes \Lambda$ as well as C remain unaltered. Compared to Eq. (3), however, the expression of U changes and appears more intricate due to the interchange of subscripts (see Appendix B.3 for more details). It can be written as $U = (I \otimes \Lambda X)L$ with a sparse $N^2 \times N^2$ matrix L that substracts $\Lambda \boldsymbol{x}_a$ from all columns of the ath block of the block diagonal matrix $I \otimes \Lambda X$. Figure 1 illustrates the decomposition for the exponential quadratic a.k.a. radial basis function (RBF) kernel. Explicit expressions for a few common dot product and stationary kernels are found in Appendix B.2.1 and B.3.1.

2.3. Implementation

The structure of the Gram matrix promotes two important tricks that enable efficient inference with gradients. Computational gains come into play when $N < D$, but for any choice of N, the uncovered structure enables massive savings in storage and enables efficient approximate inversion schemes.

Low-data Regime In the high-dimensional regime with a small number of observations $N < D$ the inverse of the Gram matrix can be efficiently obtained from Woodbury's matrix inversion lemma (Woodbury, 1950)

$$\left(B + UCU^\top\right)^{-1} = \\ B^{-1} - B^{-1}U \left(C^{-1} + U^\top B^{-1} U\right)^{-1} U^\top B^{-1} \quad (6)$$

(if the necessary inverses exist), combined with inversion properties of the Kronecker product. If B is cheap to invert and the dimensions of C are smaller than the ones of B, then the above expression can drastically reduce the computational cost of inversion. In our case $B = K' \otimes \Lambda$ for which the inverse $B^{-1} = (K')^{-1} \otimes \Lambda^{-1}$ requires the inverse of the $N \times N$ matrix K'. The main bottleneck is the inversion of the $N^2 \times N^2$ matrix $C^{-1} + U^\top B^{-1} U$ which requires $\mathcal{O}(N^6)$ operations, which is still a benefit over the naïve scaling when $N < D$. The low-rank structure along with properties of Kronecker products leads to a general solution of the linear system $[\nabla K \nabla'] \, \mathrm{vec}(Z) = \mathrm{vec}(G)$ of the form

$$Z = \Lambda^{-1} G (K')^{-1} - \tilde{X} Q \quad (7)$$

for dot product kernels with $\tilde{X} = X - \boldsymbol{c}$ and gradient observations G. Q is the unvectorized solution to

$$(C^{-1} + U^\top B^{-1} U) \, \mathrm{vec}(Q) = \mathrm{vec}(\tilde{X}^\top G (K')^{-1}). \quad (8)$$

Stationary kernels give rise to a similar expression (cf. Appendix C.1).

General Improvements The cubic computational scaling is frequently cited as the main limitation of GP inference, but often the quadratic storage is the real bottleneck. For

gradient inference that is particularly true due to the required $\mathcal{O}((ND)^2)$ memory. A second observation that arises from the decomposition is that the the Gram matrix $\nabla K \nabla'$ is fully defined by the much smaller matrices K', K'' (both $N \times N$), ΛX ($D \times N$) and Λ ($D \times D$, but commonly chosen diagonal or even scalar). Thus, it is sufficient to keep only those in memory instead of building the whole $DN \times DN$ matrix $\nabla K \nabla'$, which requires at most $\mathcal{O}(N^2 + ND + D^2)$ of storage. Importantly, this benefit arises for $D > 1$ and for any choice of N. It is further known how these components act on a matrix of size $D \times N$. For dot product kernels, a multiplication of the Gram matrix with vectorized matrix $V \in \mathbb{R}^{D \times N}$ is obtained by

$$(\nabla K \nabla') \operatorname{vec}(V) = \Lambda V K' + \Lambda \tilde{X} (K'' \odot V^\top \Lambda \tilde{X}) \quad (9)$$

A similar expression is obtained for stationary kernels, see Appendix C.2. This multiplication expression can be used with an iterative linear solver (Gibbs & MacKay, 1997; Gardner et al., 2018a) to exactly solve a linear system in DN iterations, in exact arithmetic. It can also be used to obtain an approximate solution in fewer iterations. The multiplication routine is further amenable to preconditioning which can drastically reduce the required number of iterations (Eriksson et al., 2018) and ensure convergence, as well as popular kernel sparsification techniques to lower the computational cost.

3. Related Work

Exact derivative observations have been used to condition GPs on linearizations of dynamic systems (Solak et al., 2003) as a way to condense information in dense input regions. This required the number of replaced observations to be larger than the input dimension in order to benefit. Derivatives have also been employed to speed up sampling algorithms by querying a surrogate model for gradients (Rasmussen, 2003). In both previous cases the algorithms were restricted to low-dimensional input but showed improvements over baselines despite the computational burden. In Sec. 4.3 we will revisit the idea of sampling in light of our results.

Modern GP models that use gradients always had to rely on various approximations to keep inference tractable. Solin et al. (2018) linearly constrained a GP to explicitly model curl-free magnetic fields (Jidling et al., 2017). This involved using the differentiation operator and was made computationally feasible with a reduced rank eigenfunction expansion (Solin & Särkkä, 2020). Angelis et al. (2020) extended the quadrature Fourier feature expansion (QFF) (Mutny & Krause, 2018) to derivative information. The authors used it to construct a low-rank approximation for efficient inference of ODEs with a high number of observations. Derivatives have also been included in Bayesian optimization but mainly

in low-dimensional spaces (Osborne et al., 2009; Lizotte, 2008), or by relying on a single gradient observation in each iteration (Wu et al., 2017).

A more task-agnostic approach was presented by Eriksson et al. (2018). The authors derived the gradient Gram matrix for the structured kernel interpolation (SKI) approximation (Wilson & Nickisch, 2015), and its extension to products SKIP (Gardner et al., 2018b). This was used in conjunction with fast matrix-vector multiplication on GPUs (Gardner et al., 2018a) and a subspace discovery algorithm to make inference efficient. Tej et al. (2020) used a similar approach but further incorporated Bayesian quadrature with gradient inference to infer a noisy policy gradient for reinforcement learning to speed up training.

An obvious application of gradients for inference is in optimization and in some cases linear algebra. These are two fields we will discuss further in Sec. 4. Probabilistic versions of linear algebra and quasi-Newton algorithms can be constructed by modeling the Hessian with a matrix-variate normal distribution and update the belief from gradient observations (Hennig, 2015; Wills & Schön, 2019; de Roos & Hennig, 2019; Wenger & Hennig, 2020). In Sec. 4.2 we will connect this to GP inference for a special kernel. Inference in such models has cost $\mathcal{O}(N^2 D + N^3)$.

Extending classic quasi-Newton algorithms to a nonparametric Hessian estimate has been done by Hennig & Kiefel (2013) and followed up by Hennig (2013). The authors modeled the elements of the Hessian using a high-dimensional GP with the RBF kernel and a special matrix-variate structure to allow cost-efficient inference. They also generalized the traditional secant equation to integrate the Hessian along a path for observations, which was possible due to the closed-form integral expression of the RBF kernel. Wills & Schön (2017) expanded this line of work in two directions. They used the same setup as Hennig & Kiefel (2013) but explicitly encoded symmetry of the Hessian estimate. The authors also considered modeling the joint distribution of function, gradient and Hessian ($[f, \boldsymbol{g}, H]$) for system identification in the presence of significant noise, and where the computational requirement of inference was less critical. In Section 4.1 we present two similar optimization strategies that utilize exact efficient gradient inference for nonparametric optimization.

4. Applications

Section 2 showed how gradient inference for GPs can be considerably accelerated when $N < D$. We outline three applications that rely on gradients in high dimensions and that can benefit from a gradient surrogate: optimization, probabilistic linear algebra, and sampling.

4.1. Optimization

Unconstrained optimization of a scalar function $f(\boldsymbol{x})$: $\mathbb{R}^D \to \mathbb{R}$ consists of locating an input $\boldsymbol{x}_*$ such that $f(\boldsymbol{x}_*)$ attains an optimal value, here this will constitute a minimum. This occurs at a point where $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$. We focus on Hessian inference from gradients in quasi-Newton methods and then suggest a new method that allows inferring the minimum from gradient evaluations. Pseudocode for an optimization algorithm that uses the inference procedure is available in Alg. 1.

4.1.1. HESSIAN INFERENCE

Quasi-Newton methods are a popular group of algorithms that includes the widely known BFGS rule (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). These algorithms either estimate the Hessian $H(\boldsymbol{x}) = \nabla \nabla^\top f(\boldsymbol{x})$ or its inverse from gradients. A step direction at iteration t is then determined as $\boldsymbol{d}_t = -[H(\boldsymbol{x}_t)]^{-1} \nabla f(\boldsymbol{x}_t)$. Hennig & Kiefel (2013) showed how popular quasi-Newton methods can be interpreted as inference with a matrix-variate Gaussian distribution conditioned on gradient information. Here we extend this idea to the nonparametric setting by inferring the Hessian from observed gradients. In terms of Eq. (1), we consider the linear operator $\mathcal{L}$ as the second derivative, i.e., the Hessian for multivariate functions. Once a solution Z_b^l to $(\nabla K \nabla^\top) \operatorname{vec}(Z) = \operatorname{vec}(G)$ has been obtained as presented in Section 2.3, it is possible to infer the mean of the Hessian at a point $\boldsymbol{x}_a$

$$[\bar{H}(\boldsymbol{x}_a)]^{ij} = \sum_{bl} (\partial_a^i \partial_a^j \partial_b^l k) Z_b^l. \tag{10}$$

This requires the third derivative of the kernel matrix and an additional partial derivative of Eq. (2) which results in

$$\begin{aligned}
\partial_a^i (\partial_a^j \partial_b^l k(r)) &= k''_{ab} \cdot \Lambda^{jl} (\partial_a^i r) + k''_{ab} \cdot \Lambda^{il} (\partial_a^j r) \\
&\quad + k''_{ab} (\partial_a^i \partial_a^j r)(\partial_b^l r) \\
&\quad + k'''_{ab} (\partial_a^i r)(\partial_a^j r)(\partial_b^l r).
\end{aligned} \tag{11}$$

With these derivatives, the posterior mean of the Hessian in Eq. (10) takes the form

$$\bar{H}(\boldsymbol{x}_a) = \begin{bmatrix} \Lambda \tilde{X}, \Lambda Z \end{bmatrix} \begin{bmatrix} M & \hat{M} \\ \hat{M} & 0 \end{bmatrix} \begin{bmatrix} \tilde{X}^\top \Lambda \\ Z^\top \Lambda \end{bmatrix} + \Lambda \cdot \operatorname{Tr}(\breve{M}) \tag{12}$$

with M, $\hat{M}$ and $\breve{M}$ diagonal matrices of size $N \times N$ containing expressions of k'' and k''', found in Appendix D alongside a derivation. $\tilde{X}$ is either $(\boldsymbol{x}_a - X)$ for stationary kernels or $(X - \boldsymbol{c})$ for the dot-product kernels. The posterior mean of the Hessian is of diagonal + low-rank structure for a diagonal Λ, which is common for quasi-Newton algorithms. The matrix inversion lemma, Eq. (6), can then be applied to efficiently determine the new step direction. On a high

level this means that once Z in Eq. (7) has been found, then the cost of inferring the Hessian with a GP and inverting it is similar to that of standard quasi-Newton algorithms.

4.1.2. INFERRING THE OPTIMUM

The standard operation of Gaussian process regression is to learn a mapping $f(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$. With the gradient inference it is now possible to learn a nonparametric mapping $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}^D$, *but* this mapping can also be reversed to learn an input that corresponds to a gradient. In this way it is possible to learn $\boldsymbol{x}(\boldsymbol{g})$ and we can evaluate where the model believes $\boldsymbol{x}(\boldsymbol{g} = 0)$, i.e, the optimum $\boldsymbol{x}_*$, lies to construct a new step direction. The posterior mean of $\boldsymbol{x}_*$ conditioned on the evaluation points X at previous gradients G is

$$\begin{aligned}
\bar{\boldsymbol{x}}_* &= \boldsymbol{x}_t + [\nabla K(0, G) \nabla](\nabla K(G, G) \nabla)^{-1}(X - \boldsymbol{x}_t) \\
&= \boldsymbol{x}_t + \Lambda Z K'_{b*} + \Lambda \tilde{G}(K''_{b*} \odot (Z^\top \Lambda \tilde{\boldsymbol{g}}_*)).
\end{aligned} \tag{13}$$

Here we included a prior mean in the inference which corresponds to the location of the current iteration $\boldsymbol{x}_t$ and all the inference has been flipped, i.e., gradients are inputs to the kernel and previous points of evaluation are observations. This leads to a new step direction determined by $\boldsymbol{d}_{t+1} = \bar{\boldsymbol{x}}_* - \boldsymbol{x}_t$. For dot product kernels $\tilde{G} \in \mathbb{R}^{D \times N} = G - \boldsymbol{c}$ and $\tilde{\boldsymbol{g}}_* = -\boldsymbol{c}$. For stationary kernels $\tilde{G} = (\boldsymbol{g}_* - G) = -G$ and $Z^\top \Lambda \tilde{\boldsymbol{g}}_*$ is replaced by $\sum_l Z_b^l \cdot (\Lambda G)_b^l$, derivations in Appendix E.1.

4.2. Probabilistic Linear Algebra

Assume the function we want to optimize is

$$f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_*)^\top A(\boldsymbol{x} - \boldsymbol{x}_*), \tag{14}$$

with $A \in \mathbb{R}^{D \times D}$ a symmetric and positive definite matrix. Finding the minimum is equivalent to solving the linear system $A\boldsymbol{x} = \boldsymbol{b}$, because the gradient $\nabla f(\boldsymbol{x}) = A(\boldsymbol{x} - \boldsymbol{x}_*)$ is zero when $A\boldsymbol{x} = A\boldsymbol{x}_* := \boldsymbol{b}$. To model this function we use the second order polynomial kernel

$$k(\boldsymbol{x}_a, \boldsymbol{x}_b) = \frac{1}{2} \left[(\boldsymbol{x}_a - \boldsymbol{c})^\top \Lambda (\boldsymbol{x}_b - \boldsymbol{c}) \right]^2,$$

and we include a prior mean of the gradient $\boldsymbol{g}_c = \nabla f(\boldsymbol{c}) = A(\boldsymbol{c} - \boldsymbol{x}_*)$. For this setup the overall computational cost decreases from $\mathcal{O}(N^2 D + (N^2)^3)$ to $\mathcal{O}(N^2 D + N^3)$ because Eq. (8) has the analytical solution

$$Q = \frac{1}{2}(\tilde{X}^\top \Lambda \tilde{X})^{-1}(\tilde{X}^\top A \tilde{X}),$$

which only requires the inverse of an $N \times N$ matrix instead of an $N^2 \times N^2$. The appearance of $(\tilde{X}^\top A \tilde{X})$ stems from

$$\begin{aligned}
\tilde{X}^\top (G - \boldsymbol{g}_c) = \tilde{X}^\top (A(X - \boldsymbol{x}_*) - A(\boldsymbol{c} - \boldsymbol{x}_*)) = \\
= \tilde{X}^\top (A(X - \boldsymbol{c})) = \tilde{X}^\top A \tilde{X}.
\end{aligned}$$

Algorithm 1 GP-[H/X] Optimization

> **Input:** data $\boldsymbol{x}_0$, $f(\cdot)$, gradient $g(\cdot)$, kernel k, size m
> $\boldsymbol{d}_0 = -g(\boldsymbol{x}_0)$
> **repeat**
>> $\alpha = \text{LineSearch}(\boldsymbol{d}_t, f(\cdot), g(\cdot))$
>> $\boldsymbol{x}_t \mathrel{+}= \alpha \boldsymbol{d}_t$
>> $\boldsymbol{g}_t = \text{g}(\boldsymbol{x}_t)$
>> $H_t = \text{inferH}(\boldsymbol{x}_t \mid X, G)$ {Eq. (12)}
>> $\boldsymbol{d}_t = -H_t^{-1}\boldsymbol{g}_t$ {quasi-Newton step}
>> $\text{updateData}(k, \boldsymbol{x}_t, \boldsymbol{g}_t)$ {Keep last m observations}
>> $\boldsymbol{d}_t = \text{inferMin}(\boldsymbol{0} \mid X - \boldsymbol{x}_t, G)$ {Eq. (13)}
>> **if** $\boldsymbol{d}_t^\top \boldsymbol{g}_t > 0$ **then**
>>> $\boldsymbol{d}_t = -\boldsymbol{d}_t$ {Ensure descent}
>>
>> **end if**
>
> **until** converged

It is now possible to apply the Hessian and optimum inference from Sec. 4.1 specifically to linear algebra at reduced cost. If the Hessian inference (cf. Sec. 4.1.1) is used in this setting, then it leads to a matrix-based probabilistic linear solver (Bartels et al., 2019; Hennig, 2015). If instead the reversed inference on the optimum is used (cf. Sec. 4.1.2), it will lead to an algorithm reminiscent of the solution-based probabilistic linear solvers (Bartels et al., 2019; Cockayne et al., 2019). A full comparison is beyond the scope of this paper and is left for future work.

4.3. Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) (Duane et al., 1987), is a Markov chain Monte Carlo algorithm that overcomes random walk behavior by introducing gradient information into the sampling procedure (Neal et al., 2011; Betancourt, 2017). The key idea is to augment the state space by a momentum variable $\boldsymbol{p}$ and simulate the dynamics of a fictitious particle of mass m using Hamiltonian mechanics from physics in order to propose new states. The potential energy E of states $\boldsymbol{x}$ relates to the target density P as $P(\boldsymbol{x}) \propto \exp(-E(\boldsymbol{x}))$. The Hamiltonian represents the overall energy of the system, i.e., potential and kinetic contributions

$$H(\boldsymbol{x}, \boldsymbol{p}) = E(\boldsymbol{x}) + \frac{\boldsymbol{p}^\top \boldsymbol{p}}{2m}, \qquad (15)$$

and is a conserved quantity. The joint density of $\boldsymbol{x}$ and $\boldsymbol{p}$ is $P(\boldsymbol{x}, \boldsymbol{p}) \propto e^{-H(\boldsymbol{x}, \boldsymbol{p})}$. Hamiltonian dynamics are solutions to the Hamiltonian equations of motion

$$\frac{d\boldsymbol{x}}{dt} = \nabla_{\boldsymbol{p}} H = \frac{\boldsymbol{p}}{m} \quad \text{and} \quad \frac{d\boldsymbol{p}}{dt} = \nabla_{\boldsymbol{x}} H = -\nabla_{\boldsymbol{x}} E. \quad (16)$$

New states are proposed by numerically simulating trajectories for T steps and stepsize ϵ using a leapfrog integrator which alternates updates on $\boldsymbol{p}$ and $\boldsymbol{x}$. The theoretical acceptance rate is 1 due to energy conservation; in practice,

the discrete solver alters the energy and the new state is accepted or rejected according to a standard Metropolis acceptance criterion. In this iterative procedure, the gradient of the potential energy $\nabla_{\boldsymbol{x}} E$ (but not E itself) has to be evaluated repeatedly.

HMC is therefore inappropriate to simulate from probabilistic models in which the likelihood is costly to evaluate. This setting arises, e.g., when evaluations of the likelihood rely on simulations, or for large datasets $y_{1:M}$ such that $E(\boldsymbol{x}) = -\sum_{i=1}^{M} \log p(y_i \mid \boldsymbol{x}) - \log p(\boldsymbol{x})$. For the latter case, subsampling has been proposed. It gives rise to stochastic gradients, but still produces valid states (Welling & Teh, 2011; Chen et al., 2014). Betancourt (2015) advises against the use of subsampling in HMC because the discrepancy between the true and subsampled gradient grows with dimension. Wrong gradients may yield trajectories that differ significantly from trajectories of constant energy and yield very low acceptance rates and thus, poor performance.

An alternative is to construct a surrogate over the gradient of the potential energy. Rasmussen (2003) used GPs to jointly model the potential energy and its gradients. More recently, Li et al. (2019) obtained better performance with a shallow neural network that is trained on gradient observations during early phases of the sampling procedure. With novel gradient inference routines we revisit the idea to replace $\nabla_{\boldsymbol{x}} E$ by a GP gradient model that is trained on spatially diverse evaluations of the gradient during early phases of the sampling.

5. Experiments

In the preceding section we outlined three applications where nonparametric models could benefit from efficient gradient inference in high dimensions. These ideas have been explored in previous work with the focus of improving traditional baselines, but always with various tricks to circumvent the expensive gradient inference. Since the purpose of this paper is to enable gradient inference and not develop new competing algorithms, the presented experiments are meant as a proof-of-concept to assess the feasibility of high-dimensional gradient inference for these algorithms. To this end, the algorithms only used available gradient information in concordance with the baseline. Details and parameters for reproducibility of all experiments are in Appendix F.

5.1. Linear Algebra

Consider the linear algebra, i.e., quadratic optimization, problem in Eq. (14). Quadratic problems are ubiquitous in machine learning and engineering applications, since they form a cornerstone of nonlinear optimization methods. In our setting, they are particularly interesting due to the computational benefits highlighted in section 4.2. There has already been plenty of work studying the performance
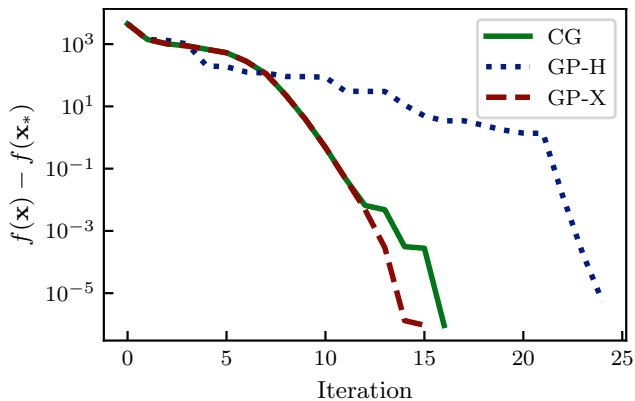
Figure 2. Optimization of a 100-dimensional quadratic function, Eq. (14), using Alg. 1 with a quadratic kernel as outlined in Sec. 4.2. The new solution-based inference shows performance similar to CG. The presented Hessian-based algorithm uses a fixed $c = 0$ which compromises the performance.
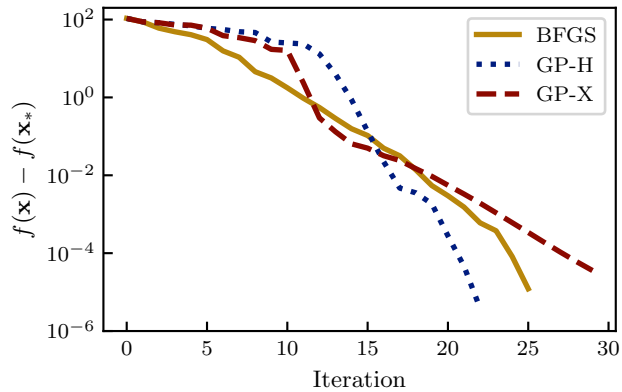

Figure 3. Comparison of Alg. 1 with an isotropic RBF kernel against `scipy`'s implementation of BFGS on a 100-dimensional version of Eq. (17). All algorithms shared the same line search routine and show similar performance.

of probabilistic linear algebra routines (Wenger & Hennig, 2020; Bartels et al., 2019; Cockayne et al., 2019), of which the proposed Hessian inference for linear algebra is already known (Hennig, 2015). We include a synthetic example of the kind Eq. (14) to test the new reversed inference on the solution in Eq. (13). Figure 2 compares the convergence of the gold-standard method of conjugate gradients (CG) (Hestenes et al., 1952) with Alg. 1 using the efficient inference of section 4.2. The matrix A was generated to have spectrum with approximately the 30 largest eigenvalues in $[1, 100]$ and the rest distributed around 0.5. The GP-algorithm retained all the observations to operate similarly to other probabilistic linear algebra routines. In particular, the probabilistic methods also the optimal step length $\alpha_i = -\boldsymbol{d}_i^\top \boldsymbol{g}_i / \boldsymbol{d}_i^\top A \boldsymbol{d}_i$ that is used by CG.

5.2. Nonlinear Optimization

The prospect of utilizing a nonparametric model for optimization is more interesting to evaluate in the nonlinear setting. In Fig. 3 the convergence of both versions of Alg. 1 is compared to `scipy`'s BFGS implementation. The nonparametric models use an isotropic RBF kernel with the last 2 observations for inference. All algorithms share the same line search routine. The function to be minimized is a relaxed version of a 100-dimensional Rosenbrock function (Rosenbrock, 1960)

$$f(\boldsymbol{x}) = \sum_{i=1}^{D-1} x_i^2 + 2 \cdot (x_{i+1} - x_i^2)^2. \tag{17}$$

A hyperplane of the function can be seen on the left in Fig. 6 for the first two dimensions with every other dimension evaluated at 0.

5.3. Gradient Surrogate Hamiltonian Monte Carlo

For HMC, we take a similar approach to the one taken by Rasmussen (2003) and build a global surrogate model on the gradient of the potential energy $E(\boldsymbol{x}) = -\log P(\boldsymbol{x})$. Our model differs in that we predict the gradient ∇E only from previous gradient observations, not including function evaluations as does Rasmussen (2003).

We construct a synthetic 100 dimensional target density that is banana-shaped in two dimensions and Gaussian in all the other dimensions. Fig. 4 shows the conditional density in the two non-Gaussian dimensions, together with projected samples that were collected using standard HMC and HMC using a GP gradient surrogate, which we denote as GPG-HMC. For training of GPG-HMC, we assign a budget $N = \lfloor \sqrt{D} \rfloor$ and run HMC until $N/2$ points are found that are more than a kernel lengthscale apart. Then we switch into the surrogate mode in which the true ∇E is queried only if a new location sufficiently far from the previous ones is found to condition the GP on until the budget is reached.

In Fig. 4, the isotropic square exponential kernel is aligned with the intrinsic dimensions of the problem. Therefore, we also consider 10 arbitrary rotations of the same problem by applying a random orthonormal matrix on the input and repeat each configuration for 10 different initializations. We find that over 2000 samples, HMC has an acceptance rate of 0.46 ± 0.02 and GPG-HMC achieves 0.50 ± 0.02 using the gradient surrogate. GPG-HMC was conditioned on $N = 10$ gradient observations collected during the first 650 ± 82 iterations of HMC. The higher acceptance rate is related to the mismatch between the estimated and true gradient that tends to cause a skewed distribution of ΔH towards positive values. The acceptance criterion still queries the true potential energy E, thus GPG-HMC produces valid sam-
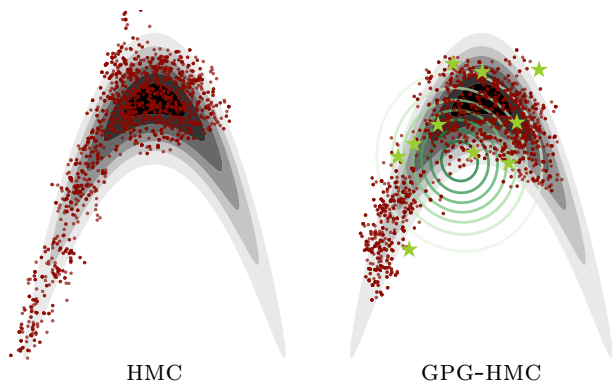
Figure 4. 2000 samples drawn with HMC *(left)* and GPG-HMC on a 100 dimensional problem. Displayed is a projection onto 2 dimensions, all other dimensions are Gaussian. Acceptance rates are 0.51 (HMC) and 0.39 (GPG-HMC). GPG-HMC uses 372 iterations with HMC with acceptance rate 0.57 of which it selects 10 points for training, displayed as ★. Elliptical contours indicate the posterior on the target inferred from the 10 gradient evaluations.

ples of e^{-E}. As E gets increasingly expensive to evaluate, GPG-HMC thus offers a lightweight surrogate that drastically reduces the number of calls to the true gradient.

5.4. Runtime Comparison

To better understand the computational saving from the presented decomposition we compare to the CPU runtime of building and solving $\nabla K \nabla' Z = \nabla f(X)$ with the traditional Cholesky decomposition. The results are presented in Fig. 5 in terms of the ratio of observations to dimension for different dimensions. For the larger dimensions a maximum size of the Gram matrix was $50\,000$ at which point the hardware ran out of memory for the Cholesky decomposition. The Woodbury decomposition can fit more observations than the full Cholesky for a limited memory budget in the *high*-dimensional limit. To generate the data we used the same setup as in Sec. 5.2 for different dimensions and random data in $[-2, 2]^D$. To alleviate numerical problems an identity matrix was added to the Gram matrix so all eigenvalues are larger than 1. Each data point in Fig. 5 is the average of three repetitions of a problem. In low D and at very small N/D our naïve `python` implementation suffers from overhead of pythonic operations compared to the vectorized implementation of the Cholesky decomposition in `scipy`. For larger dimensions with a small number of observations the computational cost can reduce by several orders of magnitude.

Another aspect of the decomposition is the MVM presented in Section 2.3 which can reduce computations and memory requirements. A qualitative evaluation is available in Fig. 6. The right plot shows a hyperplane with function values inferred from gradient observations evaluated at $N = 1000$
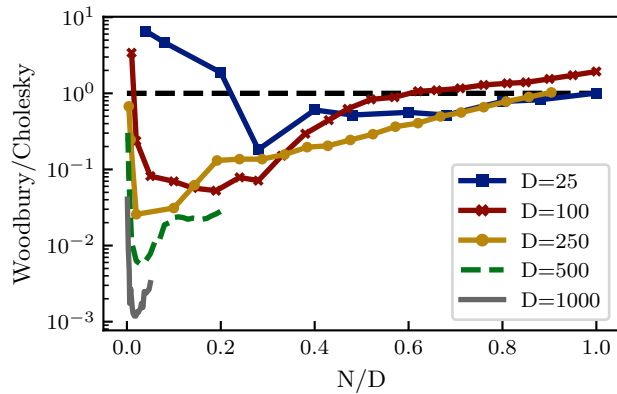

Figure 5. Relative CPU runtime comparison of the proposed Woodbury and traditional Cholesky decomposition to solve $\nabla K(X, X) \nabla' \mathrm{vec}(Z) = \mathrm{vec}(\nabla f(X))$. For different dimensions (D) a range of observations (N) is presented for an isotropic RBF kernel.
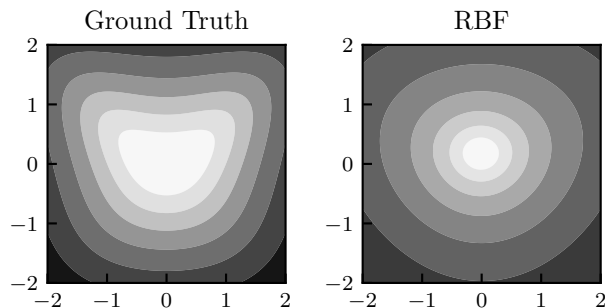

Figure 6. The first two dimensions of Eq. (17) along with the inferred curvature from 1000 randomly distributed samples. The inferred function has identified the minimum and a slight elongation of the function but not the minute details of the shape.

uniformly randomly distributed evaluations in the hypercube $\boldsymbol{x}_n^i \in [-2, 2]^{100}$. Constructing the Gram matrix for these observations would require $(1000 \cdot 100)^2$ floating point numbers, which for double precision would amount to $> 74\,\mathrm{GB}$ of memory.

The multiplication in Eq. (9) was used in conjunction with an iterative linear solver to approximately solve the linear system. This approach required storage of $3ND + 3N^2$ numbers (CG requirements and intermediate matrices included) amounting to a total of only 25 MB of RAM. The solver ran for 520 iterations until a relative tolerance of 10^{-6} was reached, which took 4.9 seconds on a 2.2GHz 8-core processor. Extrapolating this time to $100 \cdot 1000$ iterations (the time to theoretically solve the linear system exactly) would yield approximately 16 minutes. Such iterative methods are sensitive to roundoff errors and are not guaranteed to converge for such large matrices without preconditioning. We did not employ preconditioning as our focus was on

the computational feasibility rather than the best possible model. It will also require research into efficient utilization of the MVM structure for preconditioning and is therefore left for future work. The required number of iterations to reach convergence vary with the lengthscale of the kernel and chosen tolerance. For this experiment a lengthscale of $\ell^2 = 10 \cdot D$ was used with the isotropic RBF kernel, i.e., the inverse lengthscale matrix $\Lambda = 10^{-3} \cdot I$.

6. Discussion and Future Work

We have presented how structure inherent in the kernel Gram matrix can be exploited to lower the cost of GP inference with gradients from cubic to linear in the dimension. This technical observation principally opens up entirely new perspectives for high-dimensional applications in which gradient inference has previously been dismissed as prohibitive. We demonstrate on a conceptual level the great potential of this reformulation on various algorithms. The major intention behind the paper, however, is to spark research to overhaul algorithms that operate on high-dimensional spaces and leverage gradient information.

The speed-up in terms of dimensionality does not come without limitations. Our proposed decomposition compromises the number of permissible gradient evaluations compared to the naïve approach to gradient inference. Hence, our method is applicable only in the low-data regime in which $N < D$. This property is unproblematic in applications that benefit from a local gradient model, e.g., in optimization. Nevertheless, we also found a remedy for the computational burden when $N > D$ using iterative schemes. Furthermore, the structure we uncovered allows storing the quantities that are necessary to multiply the Gram matrix with an arbitrary vector. We thus showed that global models of the gradient are possible when a low-confidence gradient belief is sufficient. This is of particular interest for GP implementations that leverage the massive parallelization available on GPUs where available memory often becomes the bottleneck.

The most efficient numerical algorithms use knowledge about their input to speed up the execution. Explicit structural knowledge is usually reflected in hard-coded algorithms, e.g., linear solvers for matrices with specific properties that are known a priori. Structure can also be included in probabilistic numerical methods where the chosen model encodes known symmetries and constraints. At the same time, these methods are robust towards numeric uncertainty or noise, which can be included in the probabilistic model. Since Gaussian processes form a cornerstone of probabilistic numerical methods (Hennig et al., 2015), our framework allows the incorporation of additional functional constraints into numerical algorithms for high-dimensional data. Actions taken by such algorithms are then better suited to the problem at hand. The cheap inclusion of GP gradient information in numerical routines might therefore enable new perspectives for algorithms with an underlying probabilistic model.

References

Angelis, E., Wenk, P., Schölkopf, B., Bauer, S., and Krause, A. Sleipnir: Deterministic and provably accurate feature expansion for Gaussian process regression with derivatives. *arXiv preprint*, 2020.

Bartels, S., Cockayne, J., Ipsen, I., and Hennig, P. Probabilistic linear solvers: A unifying view. *Statistics and Computing*, 29, 2019.

Betancourt, M. The fundamental incompatibility of scalable Hamiltonian Monte Carlo and naive data subsampling. In *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, 2015.

Betancourt, M. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint*, 2017.

Broyden, C. G. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6, 1970.

Chen, T., Fox, E., and Guestrin, C. Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*. PMLR, 2014.

Cockayne, J., Oates, C. J., Ipsen, I. C., Girolami, M., et al. A Bayesian conjugate gradient method. *Bayesian Analysis*, 14, 2019.

de Roos, F. and Hennig, P. Active probabilistic inference on matrices for pre-conditioning in stochastic optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*. PMLR, 2019.

Duane, S., Kennedy, A., Pendleton, B. J., and Roweth, D. Hybrid Monte Carlo. *Physics Letters B*, 195, 1987.

Eriksson, D., Dong, K., Lee, E., Bindel, D., and Wilson, A. G. Scaling Gaussian process regression with derivatives. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Fletcher, R. A new approach to variable metric algorithms. *The computer journal*, 13, 1970.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix Gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, volume 31, 2018a.

Gardner, J., Pleiss, G., Wu, R., Weinberger, K., and Wilson, A. Product kernel interpolation for scalable Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, 2018b.

Gibbs, M. and MacKay, D. Efficient implementation of Gaussian processes, 1997.

Goldfarb, D. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24, 1970.

Hennig, P. Fast probabilistic optimization from noisy gradients. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*. PMLR, 2013.

Hennig, P. Probabilistic interpretation of linear solvers. *SIAM Journal on Optimization*, 25, 2015.

Hennig, P. and Kiefel, M. Quasi-Newton method: A new direction. *Journal of Machine Learning Research*, 14, 2013.

Hennig, P., Osborne, M. A., and Girolami, M. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2179):20150142, 2015.

Hestenes, M. R., Stiefel, E., et al. Methods of conjugate gradients for solving linear systems. volume 49, 1952.

Jidling, C., Wahlström, N., Wills, A., and Schön, T. B. Linearly constrained Gaussian processes. *Advances in Neural Information Processing Systems*, 30, 2017.

Li, L., Holbrook, A., Shahbaba, B., and Baldi, P. Neural network gradient Hamiltonian Monte Carlo. *Computational statistics*, 34, 2019.

Lizotte, D. J. *Practical Bayesian optimization*. PhD thesis, University of Alberta, 2008.

Mutny, M. and Krause, A. Efficient high dimensional Bayesian optimization with additivity and quadrature Fourier features. *Advances in Neural Information Processing Systems*, 31, 2018.

Neal, R. M. et al. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2011.

Osborne, M. A., Garnett, R., and Roberts, S. J. Gaussian processes for global optimization. In *International conference on learning and intelligent optimization*, volume 3, 2009.

Rasmussen, C. and Williams, C. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Rasmussen, C. E. Gaussian processes to speed up hybrid Monte Carlo for expensive Bayesian integrals. In *Seventh Valencia international meeting*, volume 7 of *Bayesian Statistics*, 2003.

Rosenbrock, H. H. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3, 1960.

Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*, 24, 1970.

Solak, E., Murray-Smith, R., Leithead, W., Leith, D., and Rasmussen, C. Derivative observations in Gaussian process models of dynamic systems. In *Advances in Neural Information Processing Systems*, volume 15, 2003.

Solin, A. and Särkkä, S. Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, 30, 2020.

Solin, A., Kok, M., Wahlström, N., Schön, T. B., and Särkkä, S. Modeling and interpolation of the ambient magnetic field by Gaussian processes. *IEEE Transactions on robotics*, 34, 2018.

Tej, A. R., Azizzadenesheli, K., Ghavamzadeh, M., Anandkumar, A., and Yue, Y. Deep Bayesian quadrature policy optimization. *arXiv preprint*, 2020.

Van Loan, C. F. The ubiquitous Kronecker product. *Journal of computational and applied mathematics*, 123, 2000.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

Wenger, J. and Hennig, P. Probabilistic linear solvers for machine learning. In *Advances in Neural Information Processing Systems*, 2020.

Wills, A. and Schön, T. Stochastic quasi-Newton with line-search regularization. *arXiv preprint*, 2019.

Wills, A. G. and Schön, T. B. On the construction of probabilistic Newton-type algorithms. In *Conference on Decision and Control*, volume 56. IEEE, 2017.

Wilson, A. and Nickisch, H. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Wilson, A. G. and Adams, R. P. Gaussian process kernels for pattern discovery and extrapolation.

Woodbury, M. A. *Inverting modified matrices*. Statistical Research Group, 1950.

Wu, J., Poloczek, M., Wilson, A. G., and Frazier, P. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems*, 2017.