Large-Scale Gaussian Processes via Alternating Projection

Kaiwen Wu[1] **Jonathan Wenger**[2] **Haydn Jones**[1] **Geoff Pleiss**[3,4] **Jacob R. Gardner**[1]
[1]University of Pennsylvania [2]Columbia University [3]University of British Columbia [4]Vector Institute

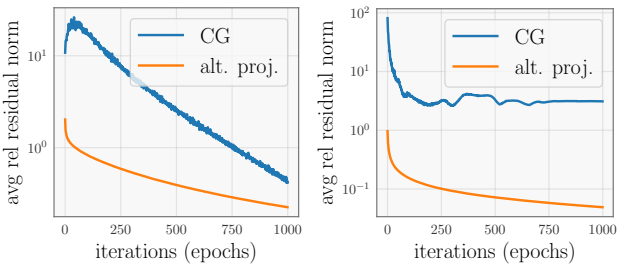
Abstract

Training and inference in Gaussian processes (GPs) require solving linear systems with $n \times n$ kernel matrices. To address the prohibitive $\mathcal{O}(n^3)$ time complexity, recent work has employed fast iterative methods, like conjugate gradients (CG). However, as datasets increase in magnitude, the kernel matrices become increasingly ill-conditioned and still require $\mathcal{O}(n^2)$ space without partitioning. Thus, while CG increases the size of datasets GPs can be trained on, modern datasets reach scales beyond its applicability. In this work, we propose an iterative method which only accesses subblocks of the kernel matrix, effectively enabling mini-batching. Our algorithm, based on alternating projection, has $\mathcal{O}(n)$ per-iteration time and space complexity, solving many of the practical challenges of scaling GPs to very large datasets. Theoretically, we prove the method enjoys linear convergence. Empirically, we demonstrate its fast convergence in practice and robustness to ill-conditioning. On large-scale benchmark datasets with up to four million data points, our approach accelerates GP training and inference by speed-up factors up to 27× and 72×, respectively, compared to CG.

1 INTRODUCTION

Scaling Gaussian process (GP) models to large datasets has been a central research topic in probabilistic machine learning for nearly two decades. The primary challenge is the cubic complexity of computing both the marginal log likelihood (MLL) during training and the predictive distribution at test time. Over the years, this problem has been addressed both from a modeling perspective (e.g., Hensman et al., 2013, 2015; Titsias, 2009; Snelson and Ghahramani, 2005; Salimbeni et al., 2018; Jankowiak et al., 2020; Katzfuss and Guinness, 2021) and from a numerical methods perspective (e.g., Cutajar et al., 2016; Pleiss et al., 2018; Gardner et al., 2018; Wang et al., 2019; Maddox et al., 2022), and contemporary work even unifies these perspectives to a degree (Artemev et al., 2021; Wenger et al., 2022b). In recent years, numerical methods have increasingly relied on matrix-free iterative methods, which access the kernel matrix through matrix-vector multiplications. These iterations are suitable for GPU acceleration (Gardner et al., 2018) and have shown success on medium to moderately large datasets (Wang et al., 2019), outperforming modeling-based approaches such as stochastic variational GPs (SVGP) (Hensman et al., 2013).

Most GP training and inference approaches based on iterative methods use classic general-purpose al-



(a) 3droad **(b)** house electric

Figure 1: Convergence of alternating projection and (preconditioned) conjugate gradient. The x-axis is the number iterations for CG and the number epochs for alternating projection. Both methods are initialized at zero, but CG increases the residual after the first iteration. **Left:** While the asymptotic convergence rate of CG can be faster than alternating projection, CG does not find a better solution than alternating projection in the first 1000 iterations. **Right:** CG struggles with convergence due to ill-conditioning and does not reach the tolerance $\epsilon = 1$. In contrast, alternating projection convergences. See §4 for more details.

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

gorithms for matrix solves, such as conjugate gradients (CG) (Cutajar et al., 2016; Gardner et al., 2018; Wang et al., 2019), MINRES (Pleiss et al., 2020), or (stochastic) gradient descent (Lin et al., 2023). There is reason to believe that such algorithms are suboptimal for modern hardware-accelerated Gaussian processes. For example, CG was purpose-built for sparse linear systems that require high-precision solutions. Neither of these properties applies to GP regression: the necessary solves involve dense covariance matrices, and tasks such as hyperparameter optimization can be performed with extremely coarse-grained solves (Wang et al., 2019; Maddox et al., 2022). These characteristics of large-scale dense operations and low precision amenability are in line with existing trends in machine learning (Courbariaux et al., 2015; Micikevicius et al., 2018), but ultimately place Gaussian processes at odds with much of the literature on numerical methods.

Much in the way that deep learning has been revolutionized by purpose-built optimizers that exploit properties of neural networks (Kingma and Ba, 2015; Loshchilov and Hutter, 2019), this paper aims to accelerate GPs with a purpose-built method leveraging (coarse-grained) covariance matrix solves on modern hardware. We introduce an iterative method to compute gradients of the marginal log-likelihood (MLL) and the posterior mean, that improves over CG in the following ways: 1) it requires $\mathcal{O}(n)$ computation per iteration rather than CG's $\mathcal{O}(n^2)$; 2) it converges rapidly and monotonically in its early stages (but does not necessarily obtain higher precision than CG); and 3) it demonstrates improved numerical stability in floating point arithmetic.

Contributions. We propose an iterative method for Gaussian process training and inference. The method computes the marginal log-likelihood derivative and posterior mean via alternating projection. Each iteration of the algorithm accesses a subblock of the kernel matrix, has linear time and memory complexity, and decreases the residual near-monotonically after every epoch. We prove that the algorithm converges linearly at a rate no slower than gradient descent, despite never operating on the full kernel matrix. Empirically, our method achieves a speed-up of up to 27× over CG-based hyperparameter training and of up to 72× over CG-based inference on a wide range of datasets. As a demonstration of its scalability and robustness to ill-conditioning, we are able to train Gaussian processes on 4 million data points, the largest dataset reported in the literature to-date without using inducing points or similar modeling approximations—to the best of our knowledge. We find that our method outperforms the stochastic variational Gaussian process by a significant margin at this scale.

2 SETUP AND BACKGROUND

Notation. Let $(\mathbf{X}, \mathbf{y})$ be a training set of n training inputs $\mathbf{X} = (\mathbf{x}_1 \quad \cdots \quad \mathbf{x}_n)^\top \in \mathcal{X} \subseteq \mathbb{R}^{n \times d}$ and labels $\mathbf{y} = (y_1 \quad \cdots \quad y_n)^\top \in \mathbb{R}^n$. Let the set $\{1, 2, \ldots, n\}$ be denoted by $[n]$. Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and an index set $I \subseteq [n]$, $\mathbf{A}_I = \mathbf{A}_{I,:}$ is the $|I| \times n$ row-indexed submatrix, $\mathbf{A}_{:,I}$ the $n \times |I|$ column-indexed submatrix, and $\mathbf{A}_{I,I}$ is the $|I| \times |I|$ principal submatrix. We use similar indexing notations for vectors.

Let $\mathbf{E} \in \mathbb{R}^{n \times n}$ be the identity matrix. $\mathbf{E}_I$ denotes the $|I| \times n$ submatrix formed by rows indexed by I. Notice that multiplication with $\mathbf{E}_I \in \mathbb{R}^{|I| \times n}$ selects rows and columns: $\mathbf{E}_I \mathbf{A} = \mathbf{A}_I$ and $\mathbf{A}\mathbf{E}_I^\top = \mathbf{A}_{:,I}$ for any $n \times n$ matrix $\mathbf{A}$. For a vector $\mathbf{u} \in \mathbb{R}^{|I|}$, left multiplying $\mathbf{E}_I^\top$ maps $\mathbf{u}$ to a n-dimensional vector $\mathbf{v}$, such that $\mathbf{v}_I = \mathbf{u}$ and the entries outside I are zeros: $\mathbf{v}_{[n] \setminus I} = 0$, where $[n] \setminus I$ is the complement of I.

Now, let $f : \mathcal{X} \to \mathbb{R}$ be a latent function, and let $k_{\boldsymbol{\theta}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a (known) positive definite kernel function with hyperparameters $\boldsymbol{\theta}$. We write $\mathbf{f} = f(\mathbf{X}) = (f(\mathbf{x}_1) \quad \cdots \quad f(\mathbf{x}_n))^\top \in \mathbb{R}^n$. Similarly, $k_{\boldsymbol{\theta}}(\mathbf{X}, \cdot) : \mathcal{X} \to \mathbb{R}^n$ denotes the vector-valued function given by $(k(\mathbf{x}_1, \cdot) \quad \cdots \quad k(\mathbf{x}_n, \cdot))^\top$, and $\mathbf{K}_{\boldsymbol{\theta}} \in \mathbb{R}^{n \times n}$ is the Gram matrix with $[\mathbf{K}_{\boldsymbol{\theta}}]_{ij} = k_{\boldsymbol{\theta}}(\mathbf{x}_i, \mathbf{x}_j)$. We omit the subscript $\boldsymbol{\theta}$ unless the context needs it.

Gaussian Process Regression. In supervised GP regression, we assume a response-generating function f that is Gaussian process distributed a priori—i.e. $f \sim \mathcal{GP}(\mu, k_{\boldsymbol{\theta}})$. For simplicity of presentation, we assume without loss of generality an exact observation model—i.e. $\mathbf{y} = f(\mathbf{X})$.[1] Given a finite test dataset $\mathbf{x}_1^*, \ldots, \mathbf{x}_M^*$, we can obtain a posterior distribution over $f(\mathbf{x}_1^*), \ldots, f(\mathbf{x}_M^*)$ using standard Gaussian conditioning rules with the posterior mean and covariance:

$$\mathbb{E}[\mathbf{f}^* \mid \mathbf{f}] = \boldsymbol{\mu} + \mathbf{K}_{*\mathbf{f}}\mathbf{K}^{-1}(\mathbf{y} - \boldsymbol{\mu}),$$
$$\mathbb{D}[\mathbf{f}^* \mid \mathbf{f}] = \mathbf{K}_{**} - \mathbf{K}_{*\mathbf{f}}\mathbf{K}^{-1}\mathbf{K}_{\mathbf{f}*}.$$

We refer the reader to Rasmussen and Williams (2006, Ch. 2) for more details.

Hyperparameter Training. The hyperparameters $\boldsymbol{\theta}$ of the GP are learned by minimizing the negative marginal log likelihood (MLL) $\ell(\boldsymbol{\theta}) := -\log p(\mathbf{y}; \boldsymbol{\theta})$. With a Gaussian process prior on f, we have $p(\mathbf{y}; \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \mathbf{K}_{\boldsymbol{\theta}})$, yielding the following minimization:

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\, \ell(\boldsymbol{\theta}) \overset{c}{=} \tfrac{1}{2}\left(\mathbf{y}^\top \mathbf{K}_{\boldsymbol{\theta}}^{-1}\mathbf{y} + \log\det(\mathbf{K}_{\boldsymbol{\theta}})\right) \qquad (1)$$

[1] Note that we can easily recover an observational noise model by setting $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = k_{\text{base}}(\mathbf{x}, \mathbf{x}') + \sigma^2 \mathbb{1}[\mathbf{x} = \mathbf{x}']$ for some k_{base} and $\sigma > 0$, where $\mathbb{1}$ is the indicator function.

Equation (1) is commonly optimized with first-order methods, which require an (unbiased) estimate of $\frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta}$. Unfortunately, as (1) cannot be written in the usual $\sum_{i=1}^{n} \ell(\mathbf{x}_i, y_i)$ form common to many machine learning algorithms, standard minibatching strategies are not readily applicable. Following prior work (e.g. Cutajar et al., 2016; Gardner et al., 2018; Wenger et al., 2022a), we use the following unbiased estimate:

$$-\tfrac{1}{2}\mathbf{y}^\top \mathbf{K}_{\boldsymbol{\theta}}^{-1} \tfrac{\partial \mathbf{K}_{\boldsymbol{\theta}}}{\partial \theta} \mathbf{K}_{\boldsymbol{\theta}}^{-1}\mathbf{y} + \tfrac{1}{2l}\sum_{i=1}^{l}\left(\mathbf{z}_i^\top \mathbf{K}_{\boldsymbol{\theta}}^{-1}\right)\tfrac{\partial \mathbf{K}_{\boldsymbol{\theta}}}{\partial \theta}\mathbf{z}_i, \quad (2)$$

where $\mathbf{z}_i$ are *i.i.d.* stochastic trace samples with zero mean $\mathbb{E}[\mathbf{z}_i] = \mathbf{0}$ and identity covariance $\mathbb{E}[\mathbf{z}_i \mathbf{z}_i^\top] = \mathbf{E}$. Note that the second term is an unbiased stochastic approximation of $\operatorname{tr}\big(\mathbf{K}_{\boldsymbol{\theta}}^{-1}\frac{\partial \mathbf{K}_{\boldsymbol{\theta}}}{\partial \theta}\big)$. Crucially, computing (2) primarily involves computing linear solves with $\mathbf{K}_{\boldsymbol{\theta}}$.

Linear Solves with Iterative Methods. When the size of $\mathbf{K}$ is large, direct methods solving $\mathbf{K}\mathbf{w} = \mathbf{b}$ are prohibitively slow. Iterative methods, such as conjugate gradients (CG), offer reduced asymptotic complexity (Cutajar et al., 2016), significant GPU acceleration (Gardner et al., 2018), and memory savings if the kernel matrix $\mathbf{K}$ is accessed in a map-reduce fashion (Wang et al., 2019; Charlier et al., 2021).

CG minimizes the quadratic objective $\frac{1}{2}\mathbf{w}^\top \mathbf{K}\mathbf{w} - \mathbf{b}^\top \mathbf{w}$ by iteratively searching along conjugated directions. Each iteration requires a $\mathcal{O}(n^2)$ matrix-vector multiplication with $\mathbf{K}$. In exact arithmetic, CG returns an exact solution after n iterations. In practice for ill-conditioned problems, CG is terminated once the residual $\mathbf{r} = \mathbf{b} - \mathbf{K}\mathbf{w}$ is small enough, e.g., $\|\mathbf{r}\| \leq \epsilon \|\mathbf{b}\|$ for some predefined tolerance parameter ϵ.

For GP hyperparameter learning often large values of the tolerance ϵ are used despite the potential for overfitting (Potapczynski et al., 2021). For instance, $\epsilon = 1$ is used in practice (Wang et al., 2019; Maddox et al., 2022) and has been the default tolerance of CG during training in popular GP software packages, including GPyTorch[2] and GPflow[3].

For hyperparameter training, each MLL derivative evaluation requires a batched linear solve $\mathbf{K}\mathbf{W} = \mathbf{B}$, where $\mathbf{B} = (\mathbf{y} \quad \mathbf{z}_1 \quad \dots \quad \mathbf{z}_l)$ with $\mathbf{z}_i$ are random samples for stochastic MLL derivative estimation in (2).

RKHS. Every kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ induces a function space $\mathcal{H} = \overline{\operatorname{span}}\{k(\mathbf{x}, \cdot) : \mathbf{x} \in \mathcal{X}\} \subseteq \mathbb{R}^{\mathcal{X}}$, known as a reproducing kernel Hilbert space (RKHS) where its inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ satisfies $\langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot)\rangle_{\mathcal{H}} = k(\mathbf{x}, \mathbf{x}')$ for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$.

[2]GPyTorch setting `https://rb.gy/qi8er`

[3]GPflow setting `https://rb.gy/mozif`

RKHS Projection. Given a set of indices $I \subseteq [n]$, define the finite dimensional linear subspaces of $\mathcal{H}$:

$$\begin{aligned} V_{[n]} &:= \operatorname{span}\{k(\mathbf{x}_i, \cdot) : i = 1, 2, \cdots, n\} \subseteq \mathcal{H}, \\ V_I &:= \operatorname{span}\{k(\mathbf{x}_i, \cdot) : i \in I\} \subseteq V_{[n]}, \end{aligned} \quad (3)$$

By definition these subspaces contain functions of the form $f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot)$ and $f(\cdot) = \sum_{i \in I} \alpha_i k(\mathbf{x}_i, \cdot)$ respectively. We can map any $f \in \mathcal{H}$ onto these subspaces using the projection operator.

Definition 1 (Projection Operator)**.** *Let $V \subseteq \mathcal{H}$ be a closed linear subspace. The projection of any $f \in \mathcal{H}$ onto V is given by the* projection operator

$$\operatorname{proj}_V(f) = \operatorname*{argmin}_{g \in V} \quad \tfrac{1}{2}\|f - g\|_{\mathcal{H}}^2,$$

which is well-defined by the Hilbert space projection theorem, i.e., the unique minimizer exists.

Intuitively, the projection operator finds the best approximation of f inside V, where the approximation error is measured by the norm $\|\cdot\|_{\mathcal{H}}$. For $V = V_{[n]}$ and $V = V_I$, the projection operator has a simple form:

$$\begin{aligned} \operatorname{proj}_{V_{[n]}}(f) &= f(\mathbf{X})^\top \mathbf{K}^{-1} k(\mathbf{X}, \cdot), \\ \operatorname{proj}_{V_I}(f) &= f(\mathbf{X})^\top \mathbf{E}_I^\top \mathbf{K}_{I,I}^{-1} \mathbf{E}_I k(\mathbf{X}, \cdot). \end{aligned} \quad (4)$$

Importantly, these projections only evaluate f and the kernel k on the training data $\mathbf{X}$ (or subset $\mathbf{X}_I$). In other words, it is unnecessary to evaluate f or k outside of $\mathbf{X}$ (or $\mathbf{X}_I$). The complexity of computing the projection $\operatorname{proj}_V(f)$ depends on the dimension of the subspace V. A projection to $V_{[n]}$ takes $\mathcal{O}(n^3)$ time and a projection to V_I takes $\mathcal{O}(|I|^3)$ time.

3 METHOD

In this section, we develop an iterative method for computing solves $\mathbf{K}^{-1}\mathbf{b}$ by alternating projection. The method supports batch linear solves with multiple right-hand sides, as required by estimating the marginal log-likelihood (MLL) derivative (2), and is amenable to GPU parallelism. We cast the linear solve as a projection in the RHKS $\mathcal{H}$ and decompose the projection into a sequence of small-scale subproblems. Each subproblem is solved in $\mathcal{O}(n)$ time, allowing frequent updates. An appealing feature of alternating projection, as we will see later later, is that it typically makes rapid progress in the early stage and finds a medium-precision solution quickly, which are already good enough for GP training and predictions.

High Level Approach. Let k be strictly positive definite and assume there is no duplicate data. Then there exists $g \in \mathcal{H}$ interpolating $\mathbf{b}$, i.e., $g(\mathbf{X}) = \mathbf{b}$. The

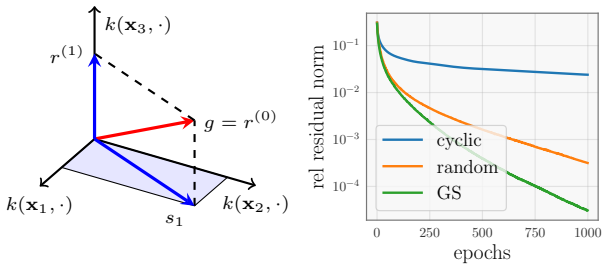


Figure 2: Left: Illustration of alternating projection. $s^{(1)}$ is the projection of $g = r^{(0)}$ onto the subspace spanned by $k(\mathbf{x}_1, \cdot)$ and $k(\mathbf{x}_2, \cdot)$. The residual $r^{(1)} = g - s^{(1)}$ will be projected to other coordinates in the next iteration. **Right:** Gauss-Southwell block selection rule results in faster convergence than random and cyclic selection rules.

exact form of g is not important (or unique for that matter); rather, we are interested in its projection onto the subspace $V_{[n]}$, which by (4) is

$$\mathrm{proj}_{V_{[n]}}(g) = \mathbf{b}^\top \mathbf{K}^{-1} k(\mathbf{X}, \cdot).$$

Thus, the solution $\mathbf{K}^{-1}\mathbf{b}$ can be obtained from the coefficients of the projection $\mathrm{proj}_{V_{[n]}}(g)$. At a first glance, it is not even clear how to come up with this function g, let alone computing its projection. As we will see soon, we do not need an explicit representation of g.

Directly projecting g onto $V_{[n]}$ is computationally infeasible, as the time complexity is cubic in n. Instead, we partition $[n]$ into subsets $\mathcal{P} = \{I_1, I_2, \cdots, I_m\}$. For each subset $I \in \mathcal{P}$, the projection to the linear subspace $V_I \subseteq V_{[n]}$ is cheap, provided that $|I|$ is small. Thus, we construct the (full) projection $\mathrm{proj}_{V_{[n]}}(g)$ by iteratively computing the projection onto the linear subspaces V_I where $I \in \mathcal{P}$.

Starting from $r^{(0)} = g$ and $s^{(0)} = 0$, the j-th iteration selects an index set $I \subseteq [n]$ and updates as follows

$$s^{(j+1)} = s^{(j)} + \mathrm{proj}_{V_I}\big(r^{(j)}\big) \tag{5}$$
$$r^{(j+1)} = r^{(j)} - \mathrm{proj}_{V_I}\big(r^{(j)}\big) \tag{6}$$

Intuitively, $s^{(j)}$ progressively approximates the true projection $\mathrm{proj}_{V_{[n]}}(g)$, since (5) iteratively adds the projection onto subspaces V_I to the current approximation $s^{(j)}$. Meanwhile, (6) accordingly updates the residual—the difference bewteen g and $s^{(j)}$. As $j \to \infty$, $s^{(j)}$ converges to the true projection $\mathbf{b}^\top \mathbf{K}^{-1} k(\mathbf{X}, \cdot)$ (Wendland, 2004). See Figure 2 (left panel) for an illustration of alternating projection.

Store $r^{(j)}$ Implicitly. Crucially, in the updates (5) and (6), the function $r^{(j)}$ is only ever accessed through its evaluation on the training data $\mathbf{X}$ (recall the projection formula (4)). Therefore, we only need to maintain the residual vector $\mathbf{r}^{(j)} = r^{(j)}(\mathbf{X}) \in \mathbb{R}^n$ instead of the entire function. The update (6) thus reduces to

$$\begin{aligned}\mathbf{r}^{(j+1)} &= \mathbf{r}^{(j)} - \mathrm{proj}_{V_I}\big(r^{(j)}\big)(\mathbf{X}) \\ &= \mathbf{r}^{(j)} - \mathbf{K}\mathbf{E}_I^\top \mathbf{K}_{I,I}^{-1} \mathbf{E}_I \mathbf{r}^{(j)} \\ &= \mathbf{r}^{(j)} - \mathbf{K}_{:,I} \mathbf{K}_{I,I}^{-1} \mathbf{r}_I^{(j)},\end{aligned} \tag{7}$$

where we recall that $\mathbf{E}_I$ denotes the rows of the identity matrix indexed by I.

Store $s^{(j)}$ by RKHS Bases. We prove by induction that $s^{(j)} \in V_{[n]}$ for every j and thus can be written as a linear combination $\sum_{i=1}^n w_i^{(j)} k(\mathbf{x}_i, \cdot)$ for some weight $\mathbf{w}^{(j)} \in \mathbb{R}^n$. At the 0-th iteration, we see that $s^{(0)}$ is the zero function with the weight vector $\mathbf{w}^{(0)} = \mathbf{0}$. Let $I \subseteq [n]$ be the indices selected in the j-th iteration. By the induction hypothesis, we have

$$\begin{aligned}s^{(j+1)} &= s^{(j)} + \mathrm{proj}_{V_I}\big(r^{(j)}\big) \\ &= \sum_{i=1}^n w_i^{(j)} k(\mathbf{x}_i, \cdot) + r^{(j)}(\mathbf{X})^\top \mathbf{E}_I^\top \mathbf{K}_{I,I}^{-1} \mathbf{E}_I k(\mathbf{X}, \cdot),\end{aligned}$$

where the last line gives an explicit update on $\mathbf{w}$:

$$\mathbf{w}^{(j+1)} = \mathbf{w}^{(j)} + \mathbf{E}_I^\top \mathbf{K}_{I,I}^{-1} \mathbf{r}_I^{(j)}.$$

Recalling the property of left multiplication with $\mathbf{E}_I^\top$, only entries in $\mathbf{w}$ indexed by I need to be updated, while keeping the entries outside I unchanged:

$$\begin{aligned}\mathbf{w}_I^{(j+1)} &= \mathbf{w}_I^{(j)} + \mathbf{K}_{I,I}^{-1} \mathbf{r}_I^{(j)}, \\ \mathbf{w}_{[n]\setminus I}^{(j+1)} &= \mathbf{w}_{[n]\setminus I}^{(j)}.\end{aligned} \tag{8}$$

Summary. The updates (7) and (8) yield iterations on the residual $\mathbf{r}^{(j)}$ and weight vector $\mathbf{w}^{(j)}$ by simple matrix operations. As $j \to \infty$, we have $s^{(j)}$ converges to $\mathrm{proj}_{V_{[n]}}(g)$. As a result, the residual vector $\mathbf{r}^{(j)}$ converges to zero and the weight vector $\mathbf{w}^{(j)} \to \mathbf{K}^{-1}\mathbf{b}$. We summarize this approach in Algorithm 1. Note that the algorithm can be adapted easily to perform multiple right-hand solves in parallel by replacing vectors $\mathbf{w}, \mathbf{r}, \mathbf{b}$ with matrices $\mathbf{W}, \mathbf{R}, \mathbf{B}$.

Block Selection. Selecting which block to update is crucial for fast convergence. The simplest block selection rules are random selection (sample I uniformly from $\mathcal{P}$) and cyclic selection (the j-th iteration selects the $(j \mod m)$-th block), which usually converge slowly (see Figure 2). A more sensible choice is selecting the block I with the largest residual norm

$$I = \operatorname*{argmax}_{I \in \mathcal{P}} \|\mathbf{R}_{I,:}\|_{\mathrm{F}}^2. \tag{9}$$

In the special case when $\mathbf{R}$ is an $n \times 1$ vector, the selection rule (9) reduces to the Gauss-Southwell (GS)

Algorithm 1: Alternating Projection

Input: A batched linear system $\mathbf{KW} = \mathbf{B}$
Output: The solution $\mathbf{W}^* = \mathbf{K}^{-1}\mathbf{B}$

Initialize $\mathbf{W} = \mathbf{O}$ and $\mathbf{R} = \mathbf{B}$
for $t = 1, 2, \cdots$ **do** // epoch
  for $j = 1, 2, \cdots, m$ **do** // mini-batch
    select a block $I \in \mathcal{P}$ from the partition
    $\mathbf{W}_I = \mathbf{W}_I + \mathbf{K}_{I,I}^{-1}\mathbf{R}_I$
    $\mathbf{R} = \mathbf{R} - \mathbf{K}_{:,I}\mathbf{K}_{I,I}^{-1}\mathbf{R}_I$
  end
  if $\|\mathbf{R}\| \leq \epsilon\|\mathbf{B}\|$ **then**
    return $\mathbf{W}$
end

rule (Nutini et al., 2015). When $\mathbf{R}$ is a matrix, however, the selection rule (9) is not the same as applying the GS rule independently in each column, which may select different blocks for different columns. Thus, the convergence behavior of Algorithm 1 on multiple right hand sides is not exactly the same as running the algorithm on each right hand side independently.

Cached Cholesky Factors. Updating $\mathbf{W}$ and $\mathbf{R}$ requires solving a linear system with the submatrix $\mathbf{K}_{I,I}$. To avoid repeatedly inverting the same matrices, we compute and cache the Cholesky factors of all principal submatrices $\{\mathbf{K}_{I,I} : I \in \mathcal{P}\}$ once at the beginning of Algorithm 1. Namely, we cache the Cholesky factors whenever the GP hyperparameters are updated, i.e., once per gradient computation. To facilitate parallelism, we partition the blocks evenly so that every block has the same size $|I| = b$ (except for the last block) and factorize all submatrices in a single batch Cholesky call. Caching Cholesky factors costs $\mathcal{O}(nb^2)$ time and $\mathcal{O}(nb)$ memory.

Complexity. The block selection takes $\mathcal{O}(nb)$ time. With the cached Cholesky factors available, updating the weights $\mathbf{W}$ takes $\mathcal{O}(b^2)$ time and updating the residual $\mathbf{R}$ takes $\mathcal{O}(nb)$ time. Each epoch runs $m = n/b$ inner iterations and thus takes $\mathcal{O}(nb + n^2)$ time in total—each epoch has the same complexity as a single CG iteration. A more fine-grained analysis in §F shows that each epoch has $(2 + \frac{3}{b})n^2 + (2b + 1)n$ floating point operations (FLOPs). Thus, for typical batch sizes $1 \ll b \ll n$, each epoch requires roughly the same $2n^2$ FLOPs as a single CG iteration. In the upcoming sections, we will compare the total number of CG iterations and the total number of alternating projection epochs, as a proxy of comparing FLOPs. We note that every inner iteration in Algorithm 1 has linear (in terms of n) time and memory complexity. In particular, the peak memory complexity is $\mathcal{O}(nb)$.

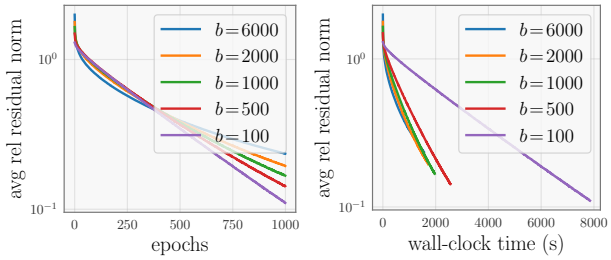


Figure 3: Convergence of alternating projection with different batch sizes b on 3droad. **Left:** Smaller batch sizes converge faster within the same epochs. **Right:** However, smaller batch sizes result in more sequential updates on the GPU and thus longer wall-clock time.

Connection with Coordinate Descent. It can be shown that Algorithm 1 produces iterates equivalent to block coordinate descent on the quadratic form (§B). We will exploit this connection to give a convergence rate. While block coordinate descent is arguably more intuitive, we introduce this method as alternating projection for two reasons. First, unlike in coordinate descent, the update rules based on alternating projection maintain the residual $\mathbf{R}$ incrementally, which enables efficient block selection rules like (9) without re-evaluating the residual. Ultimately, block coordinate descent has to be implemented as Algorithm 1 for efficiency. Second, alternating projection can be easily adapted to new settings. For instance, a parallel coordinate descent algorithm was discovered via the connection with (Dykstra's) alternating projection (Boyle and Dykstra, 1986; Tibshirani, 2017) in the setting of regularized least-squares, which hints that Algorithm 1 may be distributed.

4 CONVERGENCE

Let $\lambda_{\max}$ and $\lambda_{\min}$ be the largest and smallest eigenvalues of $\mathbf{K}$, $\kappa = \lambda_{\max}/\lambda_{\min}$ its condition number, and define $\lambda'_{\max} = \max_{I \in \mathcal{P}} \lambda_{\max}(\mathbf{K}_{I,I})$ as the maximum of the largest eigenvalues of the principal submatrices $\{K_{I,I} : I \in \mathcal{P}\}$. By leveraging the connection with coordinate descent (Nutini et al., 2022), we can prove an explicit convergence rate for Algorithm 1 when applied to a linear system with multiple right-hand sides.

Theorem 1. *Let $\mathbf{W}^*$ be the unique solution of the linear system $\mathbf{KW} = \mathbf{B}$ and $\mathbf{W}^{(t)}$ its approximation after t epochs of Algorithm 1 using the modified GS rule (9). Then it holds that*

$$\|\mathbf{W}^{(t)} - \mathbf{W}^*\|_{\mathbf{K}}^2 \leq \exp\big(-t/\kappa'\big)\|\mathbf{W}^{(0)} - \mathbf{W}^*\|_{\mathbf{K}}^2,$$

where $\|\mathbf{W} - \mathbf{W}^*\|_{\mathbf{K}}^2 = \mathrm{tr}\big((\mathbf{W} - \mathbf{W}^*)^\top \mathbf{K}(\mathbf{W} - \mathbf{W}^*)\big)$ and $\kappa' = \lambda'_{\max}/\lambda_{\min} \leq \kappa$.

The rate in Theorem 1 improves over gradient descent despite only needing submatrices, for which the above holds with $\exp(-t/\kappa)$, since generally $\kappa' \leq \kappa$. For comparison, the convergence rate of (batched) CG is $4\big((\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)\big)^{2t} \approx 4\exp\big(-4t/\sqrt{\kappa}\big)$ for a sufficiently large condition number $\kappa \gg 1$. The convergence rate of alternating projection is asymptotically faster than that of CG if $\kappa' \leq \frac{1}{4}\sqrt{\kappa}$. In general, we do not expect this condition to hold. However, alternating projection has practical advantages despite a slower asymptotic convergence rate. First, alternating projection has n/b times more updates than CG with the same number of floating point operations. More frequent updates may leads to more progress especially in the beginning of the optimization. Second, alternating projection generally decreases the residual in every epoch, whereas CG residual is well-known to be non-monotonic. Empirically, CG often increases the residual dramatically in the early stage and it takes time for CG to enter the "linear convergence phase".

Figure 1 demonstrates the above two points. This figure is plotted using two checkpoints at the 50-th epoch of GP training on the 3droad and house electric datasets respectively. The batched linear system $\mathbf{KW} = \mathbf{B}$ has 16 right-hand sides, where $\mathbf{b}_0 = \mathbf{y} - \boldsymbol{\mu}$ is the difference between the training labels and prior mean and $\{\mathbf{b}_i\}_{i=1}^{15}$ are *i.i.d.* stochastic trace samples.

Figure 2 right panel compares the convergence rates of different block selection rules. We can show that the random selection rule achieves a similar rate as Theorem 1, but only in expectation (Nesterov, 2012). In practice, the GS rule almost always converges faster than random selection.

The batch size b affects the rate in Theorem 1 through the ratio $\kappa' = \lambda'_{\max}/\lambda_{\min}$. Note that the largest eigenvalue of the principal submatrix is bounded by its trace $\lambda_{\max}(\mathbf{K}_{I,I}) \leq \mathrm{tr}(\mathbf{K}_{I,I})$, where the trace grows linearly in $|I|$. A small batch size $b = |I|$ is likely to have a small eigenvalue $\lambda'_{\max}$ and thus a faster convergence rate (at least according to Theorem 1). Indeed, as shown in Figure 3, we compare convergence rates of different batch sizes in practice. Although small batch sizes lead to faster convergence rates, they generally have a longer running time due to more sequential updates. Therefore, we recommend using the largest batch size possible subject to memory constraints. In addition, we note that the convergence rate in Theorem 1 is loose for large batch sizes b. In the extreme case where $b = n$, Algorithm 1 is equivalent to the Cholesky decomposition on the full kernel matrix $\mathbf{K}$ and thus converges to the exact solution in one update. However, Theorem 1 does not reflect that. Hence, the convergence rate in practice may be faster than the theory predicts.

5 EXPERIMENTS

We evaluate the efficacy of the alternating projections solver in a GP regression task. Our evaluation includes a training dataset of $n = 4M$, which, to the best of our knowledge, is considerably larger than any other dataset where a GP has been applied without inducing points or employing modeling approximations. Our implementation is available at `https://github.com/kayween/alternating-projection-for-gp`.

Experiments are performed on a single 24 GB NVIDIA RTX A5000 GPUs with single precision floating point arithmetic. All numerical algorithms and GP models are implemented in PyTorch and GPyTorch (Gardner et al., 2018). We use the KeOps library (Charlier et al., 2021) to implement all matrix-free numerical methods in a map-reduce fashion, thus eliminating the need to store large $n \times n$ kernel matrices in memory.

5.1 Main Result: GP Regression

We first evaluate our method on large-scale Gaussian process training tasks. We compare against GPs trained with CG, which is the predominant matrix-free GP training approach (Gardner et al., 2018; Wang et al., 2019; Maddox et al., 2022).

Metrics. Our primary desiderata for GPs are 1) low computational costs for training and 2) generalization. Hence, we report the following metrics for each training method: 1) the wall-clock **training time**, and 2/3) the trained model's **RMSE** and **NLL** measured on the test set. Additionally, for CG-trained and alternating projection-trained GPs, we report the total number of CG iterations and alternating projection epochs.

Datasets and Models. We conduct experiments on UCI regression datasets, whose statistics are shown in Table 5. Each dataset is split into 80% training and 20% test. The labels are normalized so that they have zero mean and unit variance. Almost all experiments are averaged over 5 runs. Because of resource constraints, we limit the two largest datasets—house electric and gas sensors—to 3 and 1 run respectively.

We train GPs with $\nu = 2.5$ Matérn kernels and a constant prior mean. We optimize the following hyperparameters: a scalar constant for the prior mean, a d-dimensional kernel lengthscale, a scalar outputscale, and a scalar observational noise σ^2. Experiments with $\nu = 1.5$ Matérn kernels are deferred to §E.

MLL Optimization. To compute the stochastic MLL gradient (2), we use $l = 15$ stochastic trace samples $\mathbf{z}_i$. Thus, all matrix-free methods solve a batched linear system with 16 right-hand sides with $\mathbf{b}_0 = \mathbf{y} - \boldsymbol{\mu}$

Table 1: Gaussian process training on UCI benchmark datasets. Metrics are computed across multiple runs and reported with ± one standard deviation.

Dataset	Method	RMSE	NLL	CG iters/AP epochs	Training time	Speed up
sgemm	CG	0.048 ± 0.000	$\mathbf{-1.037 \pm 0.001}$	551 ± 1	9.1m ±0.0	
$n = 241,600$	Alt. Proj.	$\mathbf{0.046 \pm 0.000}$	-0.999 ± 0.001	550 ± 0	12.2m ±0.2	0.7×
$d = 14$	SVGP	0.086 ± 0.000	-0.934 ± 0.003	NA	14.8m ±0.1	
air quality	CG	$\mathbf{0.261 \pm 0.001}$	0.143 ± 0.004	2965 ± 19	33.5m ±1.5	
$n = 382,168$	Alt. Proj.	$\mathbf{0.262 \pm 0.001}$	$\mathbf{0.137 \pm 0.003}$	550 ± 0	16.9m ±0.5	2.0×
$d = 13$	SVGP	0.363 ± 0.003	0.399 ± 0.006	NA	23.4m ±0.1	
3droad	CG	$\mathbf{0.069 \pm 0.000}$	1.324 ± 0.002	5128 ± 114	53.2m ±2.8	
$n = 434,874$	Alt. Proj.	0.076 ± 0.000	1.203 ± 0.001	676 ± 1	21.1m ±0.5	2.5×
$d = 3$	SVGP	0.327 ± 0.002	$\mathbf{0.320 \pm 0.005}$	NA	26.1m ±0.1	
song	CG	$\mathbf{0.747 \pm 0.002}$	1.140 ± 0.003	4431 ± 110	13.8h ±0.8	
$n = 515,345$	Alt. Proj.	$\mathbf{0.749 \pm 0.002}$	$\mathbf{1.132 \pm 0.002}$	550 ± 0	2.7h ±0.1	5.1×
$d = 90$	SVGP	0.790 ± 0.002	1.184 ± 0.002	NA	0.5h ±0.0	
buzz	CG	$0.321^{*} \pm 0.144$	$0.669^{*} \pm 1.152$	16726 ± 2724	31.1h ±5.4	
$n = 583,250$	Alt. Proj.	$\mathbf{0.239 \pm 0.001}$	$\mathbf{0.018 \pm 0.003}$	550 ± 0	2.0h ±0.1	15.6×
$d = 77$	SVGP	0.259 ± 0.002	0.066 ± 0.006	NA	0.6h ±0.0	
house electric	CG	-	-	$\geqslant 50441$	$\geqslant$ 11d	
$n = 2,049,280$	Alt. Proj.	$\mathbf{0.030 \pm 0.000}$	-1.148 ± 0.001	1100 ± 0	9.8h ±0.4	$\geqslant$ 26.9×
$d = 11$	SVGP	0.050 ± 0.000	$\mathbf{-1.549 \pm 0.001}$	NA	2.1h ±0.0	
gas sensors	CG	-	-	-	-	
$n = 4,178,504$	Alt. Proj.	**0.203**	$\mathbf{0.070}^{\dagger}$	1100	84.5h	
$d = 17$	SVGP	0.330 ± 0.001	0.339 ± 0.003	NA	8.7h ±0.03	

*: At test time, CG does not reach the tolerance $\epsilon = 0.01$ after 4000 iterations on some checkpoints.

-: CG does not finish GP training.

†: This predictive variance is calculated using only 500 Lanczos iterations to save time and avoid numerical instability.

and $\mathbf{b}_i = \mathbf{z}_i$ for $1 \leq i \leq 15$ in each training iteration. On the first five datasets, the GPs are trained by 50 iterations of Adam with a step size 0.1. On house electric and gas sensors, the GPs are trained by 100 iterations of Adam with a step size 0.1.

Alternating Projection Details. As discussed in §4, a large batch size is preferred empirically. We use the largest batch size that we can fit on a 24 GB GPU. The batch sizes b are set as: 6000 on sgemm, air quality and 3droad; 4000 on song and buzz; 1000 on house electric; 500 on gas sensors. We use the sequential partition $\mathcal{P}$: the data points from $(j-1)b+1$ to jb belong to the j-th block I_j for $j = 1, 2, \cdots n/b$.

The maximum CG iterations and the maximum number of alternating projection epoch is set to 1000. Following GPyTorch's CG stopping criteria, we terminate the alternating projection solves after (a) the average relative residual norm is strictly smaller than the tolerance $\epsilon = 1$ or (b) 1000 total epochs, whichever comes first. In addition, we ensure that at least 11 epochs of alternating projections have been run before termination (again following GPyTorch). We define the average relative residual norm as $\frac{1}{l+1}\sum_{i=0}^{l}\|\mathbf{r}_i\|/\|\mathbf{b}_i\|$ when there are $l+1$ right hand sides $(\mathbf{b}_0 \quad \mathbf{b}_1 \quad \cdots \quad \mathbf{b}_l)$.

CG Details. We use GPyTorch's implementation of CG, which uses the same stopping criteria as our alternating projection implementation. Following Wang et al. (2019); Wenger et al. (2022a), we use a pivoted Cholesky preconditioner of size 500 on all datasets except: house electric uses a size 300 and gas sensors uses a size 150 due to GPU memory overflow.

Prediction. At test time, the predictive mean is computed by the same iterative method used for training, i.e., CG for the CG-trained GP, alternating projection for the AP-trained GP. A limitation of our method is that it does not easily result in a cache for predictive variances. Therefore, we use 1000 Lanczos iterations as in Pleiss et al. (2018); Wang et al. (2019).

Results on $10^5 < n < 10^6$ Datasets. In Table 5, we compare the predictive performance and the training speed of CG-based versus alternating projection-based GPs. Both training procedures produce GPs with similar RMSE and NLL. We conjecture that this similarity occurs because both approaches solve linear systems up to the same tolerance, and thus find similar hyperparameters. One exception is the buzz dataset: CG struggles to converge during training, resulting in considerably worse RMSE and NLL.

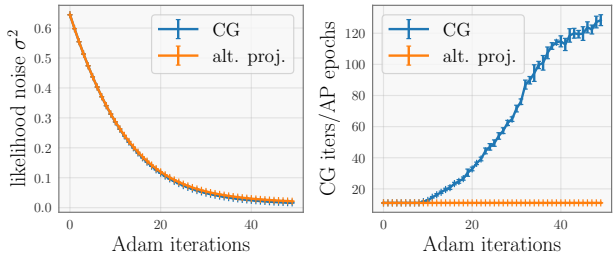
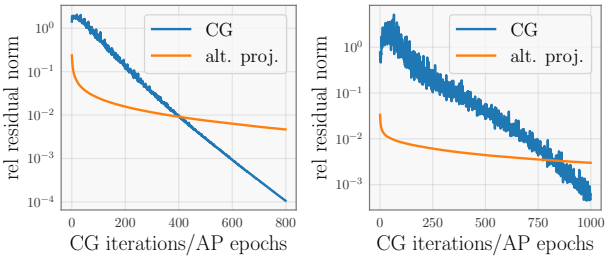


Figure 4: GP training with Adam on air quality. **Left:** As the likelihood noise σ^2 decreases in training, the kernel matrix $\mathbf{K}$ gets more ill-conditioned. **Right:** The y-axis is the number of iterations for CG and the number of epochs for alternating projection. CG is sensitive to this increased ill-conditioning, while alternating projections is robust.



(a) air quality **(b)** buzz

Figure 5: Running CG and alternating projection on test-time solves $\mathbf{K}^{-1}(\mathbf{y}-\boldsymbol{\mu})$. The x-axis is the number of iterations for CG and the number of epochs for alternating projection. **Left:** CG has a faster asymptotic convergence rate, but CG does not reach the test-time tolerance $\epsilon = 0.01$ much faster. **Right:** Alternating projection reaches the tolerance $\epsilon = 0.01$ faster despite its slower asymptotic convergence rate.

The primary difference between the two methods is training time. Alternating projection-based training is up to 27× faster than CG. The only exception is sgemm, which seems to be a well-conditioned dataset since CG converges quickly.

For reference, we also report the training/test performance of SVGPs with 1024 inducing points (see §E for experimental design details). GPs trained by alternating projection achieve substantially lower RMSE and comparable NLL compared with SVGP. We do note that SVGPs have lower NLL on 3droad and house electric, which we suspect is a limitation of the Lanczos predictive variance estimates. SVGP's predictive variances can be computed exactly and do not make use of the Lanczos estimator, while the predictive variances of CG/AP-trained GPs are approximated by 1000 Lanczos iterations. Indeed, in §E we find that the NLL gap shrinks as we increase the rank of the Lanczos variance estimator, suggesting that this gap is not a fundamental limitation of the alternating projections training methodology.

Results on $n \geq 10^6$ Datasets. Previous attempts to train GPs using iterative methods on datasets with $n \geq 10^6$ examples have used a large noise constraint $\sigma^2 \geq 0.1$ to improve the conditioning of the kernel matrix (e.g., Wang et al., 2019; Maddox et al., 2022). Since alternating projection is much less conditioning-sensitive than CG (as we will see soon in §5.2), for the first time, we are able to train the GP with a much smaller noise constraint $\sigma^2 \geq 10^{-4}$, the default in GPyTorch for the Gaussian likelihood.[4] Removing the large noise constraint in hyperparameter optimization yields much better predictive performance: the RMSE 0.030 is significantly lower than what can be achieved with high-noise constraint models (cf. §E).

[4]GPyTorch likelihood setting `https://rb.gy/fv41w`

We additionally train a GP on the gas sensors dataset with 4 million data points. To the best of our knowledge, this is the largest dataset trained on using GPs without the use of inducing points or other modeling approximations. CG training appears to be intractable on such a large dataset, requiring over a month. In contrast, alternating projection finishes training in 84.5 hours.

5.2 Effect of Kernel Matrix Conditioning

We observe empirically that alternating projection is less sensitive to ill-conditioning than CG. Figure 4 shows this phenomenon, which depicts training on the $n \approx 400K$ air quality dataset. Over the course of training, the observation noise parameter σ^2 decreases for both methods, resulting in increasingly ill-conditioned kernel matrices (as $\lambda_{\min}(\mathbf{K}) \approx \sigma^2$). At the end of training, when $\sigma^2 \approx 0.01$, CG requires over 120 iterations to converge—10× as many iterations as the beginning of training. In contrast, alternating projection consistently converges in 11 iterations despite the decreasing noise and increasing condition number. See more datasets in §E.

5.3 Alternating Projection at Test Time

Any linear solver can be used to compute the posterior mean on the test data, by solving the linear system $\mathbf{K}^{-1}(\mathbf{y}-\boldsymbol{\mu})$. We explore alternating projection at test time, shown in Table 2. With a test-time tolerance $\epsilon = 0.01$, the posterior mean computed by alternating projection is practically the same as CG: the RMSE computed by both methods are the same up to the 3rd digit after the decimal point in most cases. While alternating projection is slightly slower on medium-size

Table 2: Compute the predictive mean of the *same* GP using CG and alternating projection.

Dataset	RMSE		CG iterations/AP epochs		Time		Speed up
	CG	Alt. Proj.	CG	Alt. Proj.	CG	Alt. Proj.	
sgemm	0.046 ± 0.000	0.046 ± 0.000	95 ± 2	17 ± 0	35.0s ±1.1	13.5s ±0.3	0.4×
air quality	0.256 ± 0.001	0.256 ± 0.001	374 ± 33	388 ± 85	2.8s ±0.3	3.6s ±0.8	0.7×
3droad	0.076 ± 0.000	0.076 ± 0.000	1586 ± 31	1720 ± 79	5.8m ±0.4	9.6m ±0.6	0.6×
song	0.749 ± 0.002	0.749 ± 0.001	211 ± 7	86 ± 6	38.1m ±0.7	16.4m ±1.0	2.3×
buzz	0.241 ± 0.001	0.239 ± 0.001	579 ± 72	41 ± 10	1.2h ±0.6	4.4m ±1.2	17.2×
house electric	0.032 ± 0.000	0.030 ± 0.000	2111 ± 375	24 ± 0	5.6h ±0.6	4.7m ±0.2	72.3×
gas sensors	0.203	0.203	560	13	16.1h	27.7m	34.9×

datasets such air quality and 3droad, we observe strong speed up on larger datasets. In particular, alternating projection computes the posterior mean 17.2× faster in wall-clock time than CG on buzz, and computes the posterior mean on house electric in 5 min—72× faster than CG.

Figure 5 plots the convergence CG and alternating projection at test time. Even though CG has faster asymptotic convergence rates, alternating projection reaches the test-time tolerance $\epsilon = 0.01$ faster. Note that CG does find high precision solutions quicker, e.g., $\epsilon = 10^{-4}$, but they are seldom necessary for GP predictions (Wang et al., 2019; Maddox et al., 2022).

6 RELATED WORK

The early usage of conjugate gradients (CG) in GP training and inference dates back at least to Gibbs and MacKay (1997). Later, Yang et al. (2004); Shen et al. (2005) proposed methods speeding up CG by approximate matrix-vector multiplications. More recently, CG has been revisited by Davies (2015); Cutajar et al. (2016) on larger datasets with various preconditioners. Then, a series of work (Pleiss et al., 2018; Gardner et al., 2018; Wang et al., 2019; Artemev et al., 2021) and software packages such as GPyTorch (Gardner et al., 2018) and GPflow (Matthews et al., 2017) have popularized CG for GP training and inference.

Alternating projection (Von Neumann, 1949) is a general algorithm finding a point in the intersection of convex sets. The method presented in §3 is a special case in the reproducing kernel Hilbert space, and has been applied to radial basis function interpolation (Beatson et al., 2001; Wendland, 2004). The method turns our to be equivalent to block coordinate descent and we provide a self-contained explanation in §B. An early work applying coordinate descent to GPs with greedy block selection is done by Bo and Sminchisescu (2008). However, the greedy block selection rule is not parallelizable on modern hardware like GPUs due to the inherent sequential nature of greedy selection.

Lin et al. (2023) have recently proposed an approximate GP posterior sampling method. Their method uses stochastic gradient descent (SGD) to minimize an approximate objective based on random Fourier features and inducing points approximation. SGD generally converges sublinearly due to stochastic noise, and its step size requires manual tuning. Though, SGD could have cheaper per iteration cost independent of the data size n. In contrast, alternating projection enjoys linear convergence with no parameters to tune, and thus may be easier to use in practice. It would be interesting to apply our method in sampling as well to compare with Lin et al. (2023).

7 CONCLUSION

In this work, we propose an alternating projection method with a linear convergence rate for solving dense kernel linear systems and applied it to GP training and inference. The method quickly reaches commonly used tolerances faster than CG, requires only linear time per iteration, and is more robust to ill-conditioning. Experiments on several large-scale benchmark datasets show that the method achieves a speed-up of up to 27× over CG-based training and of up to 72× over CG-based inference. We are able to train and evaluate GPs on millions of data points without artificially inflating the observation noise for stability, leading to increased predictive performance. In particular, this includes a dataset with 4 million data points, to the best of our knowledge, the largest dataset reported in the literature so far without inducing point approximation.

Acknowledgements

The authors thank the anonymous reviewers for their helpful comments. KW, HJ and JRG are supported by NSF award IIS-2145644. JW was supported by the Gatsby Charitable Foundation (GAT3708), the Simons Foundation (542963), the NSF AI Institute for Artificial and Natural Intelligence (ARNI: NSF DBI 2229929) and the Kavli Foundation.

References

Artemev, A., Burt, D. R., and van der Wilk, M. (2021). Tighter bounds on the log marginal likelihood of Gaussian process regression using conjugate gradients. In *International Conference on Machine Learning (ICML)*, volume 139, pages 362–372.

Beatson, R. K., Light, W. A., and Billings, S. (2001). Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM Journal on Scientific Computing*, 22(5):1717–1740.

Bertin-Mahieux, T. (2011). YearPredictionMSD. UCI Machine Learning Repository.

Bo, L. and Sminchisescu, C. (2008). Greedy block coordinate descent for large scale Gaussian process regression. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Boyle, J. P. and Dykstra, R. L. (1986). A method for finding projections onto the intersection of convex sets in Hilbert spaces. In *Advances in Order Restricted Statistical Inference: Proceedings of the Symposium on Order Restricted Statistical Inference*, pages 28–47.

Charlier, B., Feydy, J., Glaunès, J. A., Collin, F.-D., and Durif, G. (2021). Kernel operations on the GPU, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6.

Chen, S. (2019). Beijing Multi-Site Air-Quality Data. UCI Machine Learning Repository.

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28.

Cutajar, K., Osborne, M., Cunningham, J., and Filippone, M. (2016). Preconditioning kernel matrices. In *International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 2529–2538.

Davies, A. J. (2015). *Effective implementation of Gaussian process regression for machine learning*. PhD thesis, University of Cambridge.

Fonollosa, J. (2015). Gas sensor array under dynamic gas mixtures. UCI Machine Learning Repository.

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

Gibbs, M. N. and MacKay, D. J. C. (1997). Efficient implementation of Gaussian processes. Technical report, Department of Physics, Cavendish Laboratory, Cambridge University.

Hebrail, G. and Berard, A. (2012). Individual household electric power consumption. UCI Machine Learning Repository.

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

Hensman, J., Matthews, A., and Ghahramani, Z. (2015). Scalable variational Gaussian process classification. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 38, pages 351–360.

Jankowiak, M., Pleiss, G., and Gardner, J. (2020). Parametric Gaussian process regressors. In *International Conference on Machine Learning (ICML)*, pages 4702–4712.

Katzfuss, M. and Guinness, J. (2021). A general framework for Vecchia approximations of Gaussian processes. *Statistical Science*, 36(1):124–141.

Kaul, M. (2013). 3D Road Network (North Jutland, Denmark). UCI Machine Learning Repository.

Kelly, M., Longjohn, R., and Nottingham, K. (2023). The UCI machine learning repository.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Lin, J. A., Antorán, J., Padhy, S., Janz, D., Hernández-Lobato, J. M., and Terenin, A. (2023). Sampling from Gaussian process posteriors using stochastic gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Maddox, W. J., Potapcynski, A., and Wilson, A. G. (2022). Low-precision arithmetic for fast Gaussian processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 180, pages 1306–1316.

Matthews, A. G. d. 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. *Journal of Machine Learning Research*, 18(40):1–6.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. (2018). Mixed precision training. In *International Conference on Learning Representations (ICLR)*.

Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.

Nutini, J., Laradji, I., and Schmidt, M. (2022). Let's make block coordinate descent converge faster: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence. *Journal of Machine Learning Research*, 23(131):1–74.

Nutini, J., Schmidt, M., Laradji, I., Friedlander, M., and Koepke, H. (2015). Coordinate descent converges faster with the Gauss-Southwell rule than random selection. In *International Conference on Machine Learning (ICML)*, volume 37, pages 1632–1641.

Paredes, E. and Ballester-Ripoll, R. (2018). SGEMM GPU kernel performance. UCI Machine Learning Repository.

Pleiss, G., Gardner, J., Weinberger, K., and Wilson, A. G. (2018). Constant-time predictive distributions for Gaussian processes. In *International Conference on Machine Learning (ICML)*, volume 80, pages 4114–4123.

Pleiss, G., Jankowiak, M., Eriksson, D., Damle, A., and Gardner, J. (2020). Fast matrix square roots with applications to Gaussian processes and Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 22268–22281.

Potapczynski, A., Wu, L., Biderman, D., Pleiss, G., and Cunningham, J. P. (2021). Bias-free scalable Gaussian processes via randomized truncations. In *International Conference on Machine Learning (ICML)*, volume 139, pages 8609–8619.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*. MIT Press.

Salimbeni, H., Cheng, C.-A., Boots, B., and Deisenroth, M. (2018). Orthogonally decoupled variational Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31.

Shen, Y., Seeger, M., and Ng, A. (2005). Fast Gaussian process regression using kd-trees. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 18.

Snelson, E. and Ghahramani, Z. (2005). Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 18.

Tibshirani, R. J. (2017). Dykstra's algorithm, admm, and coordinate descent: Connections, insights, and extensions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30.

Titsias, M. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 567–574.

Von Neumann, J. (1949). On rings of operators. reduction theory. *Annals of Mathematics*, pages 401–485.

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 (NeurIPS)*, volume 32.

Wendland, H. (2004). *Scattered Data Approximation*, volume 17. Cambridge University Press.

Wenger, J., Pleiss, G., Hennig, P., Cunningham, J., and Gardner, J. (2022a). Preconditioning for scalable Gaussian process hyperparameter optimization. In *International Conference on Machine Learning (ICML)*, volume 162, pages 23751–23780.

Wenger, J., Pleiss, G., Pförtner, M., Hennig, P., and Cunningham, J. P. (2022b). Posterior and computational uncertainty in Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 10876–10890.

Yang, C., Duraiswami, R., and Davis, L. S. (2004). Efficient kernel machines using the improved fast Gauss transform. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 17.

Yang, Z., Wilson, A., Smola, A., and Song, L. (2015). A la Carte – Learning Fast Kernels. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38, pages 1098–1106.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model.
   Yes.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm.
   Yes.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries.
   Yes.
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results.
   Yes.
   (b) Complete proofs of all theoretical results.
   Yes.
   (c) Clear explanations of any assumptions.
   Yes.
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL).
   Yes.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen).
   Yes.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times).
   Yes.
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider).
   Yes.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator if your work uses existing assets.
   Yes.
   (b) The license information of the assets, if applicable.
   Not applicable.
   (c) New assets either in the supplemental material or as a URL, if applicable.
   Not applicable.
   (d) Information about consent from data providers/curators.
   Not applicable.
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content.
   Not applicable.
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots.
   Not applicable.
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable.
   Not applicable.
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation.
   Not applicable.

Large-Scale Gaussian Processes via Alternating Projection: Supplementary Material

A Von Neumann's Alternating Projection **14**

B Connection between Coordinate Descent and Alternating Projection **14**

C Proofs **15**

D Descriptions of the UCI Datasets in the Experiments **17**

E Additional Experiments **17**
E.1 Further Experimental Details 17
E.2 GP Training on House Electric with Large Noise Constraint $\sigma^2 \geq 0.1$ 18
E.3 CG Iterations During Training 18
E.4 Increasing Lanczos Iterations Improves NLL 18
E.5 Training Gaussian Processes with Matérn $\nu = 1.5$ 19

F FLOPs in Algorithm 1 **19**

G Additional Discussions **19**

A Von Neumann's Alternating Projection

This section shows that the method presented in §3 is indeed a special case of von Neumann's alternting projection (Von Neumann, 1949). Let $g \in \mathcal{H}$ be a function that interpolates the data, i.e., $g(\mathbf{X}) = \mathbf{b}$. Write g as an orthogonal decomposition

$$g = \mathrm{proj}_{V_{[n]}}(g) + \mathrm{proj}_{V_{[n]}^\perp}(g),$$

where $V_{[n]}^\perp$ is the orthogonal complement of $V_{[n]}$. Thus, computing $\mathrm{proj}_{V_{[n]}}(g)$ reduces to computing the projection to the orthogonal complement $V_{[n]}^\perp$. Write the orthogonal complement in the form

$$V_{[n]}^\perp = \bigcap_{I \in \mathcal{P}} V_I^\perp = \bigcap_{I \in \mathcal{P}} \{f \in \mathcal{H} : f(\mathbf{x}_i) = 0 \forall i \in I\},$$

which is an intersection of n convex sets. Starting from $f^{(0)} = g$, the j-th iteration of alternating projection selects a block I and computes a projection

$$f^{(j+1)} = \mathrm{proj}_{V_I^\perp}\big(f^{(j)}\big) \tag{10}$$

As $j \to \infty$, we have $f^{(j)} \to \mathrm{proj}_{V_{[n]}}^\perp(g)$ and $g - f^{(j)} \to \mathrm{proj}_{V_{[n]}}(g)$. Recall the identity $\mathrm{proj}_{V_I^\perp}(f) = f - \mathrm{proj}_{V_I}(f)$. Thus, (10) implies

$$f^{(j+1)} = f^{(j)} - \mathrm{proj}_{V_I}\big(f^{(j)}\big),$$

which is exactly the same as the update rule (6) on the residual $r^{(j)}$. As a result, $g - f^{(j)}$ is exactly the same as $s^{(j)}$ in the update (5).

B Connection between Coordinate Descent and Alternating Projection

This section presents the connection between Algorithm 1 and coordinate descent, as shown in Algorithm 2.

Algorithm 2: Block Coordinate Descent

Input: A kernel linear system $\mathbf{KW} = \mathbf{B}$
Output: The solution $\mathbf{K}^{-1}\mathbf{B}$

Initialize $\mathbf{W} = \mathbf{O}$
for $i = 1, 2, \cdots$ **do** // epoch
  for $j = 1, 2, \cdots, m$ **do** // mini-batch
    select a block $I \in \{I_1, I_2, \cdots, I_m\}$
    $\mathbf{W}_I = \mathbf{K}_{I,I}^{-1}\big(\mathbf{B}_I - \mathbf{K}_{I,\neg I}\mathbf{W}_{\neg I}\big)$
  end
  if *converged* **then**
    return $\mathbf{W}$
end

Observe that the minimizer of the quadratic objective

$$h(\mathbf{W}) = \frac{1}{2}\mathrm{tr}\big(\mathbf{W}^\top \mathbf{K}\mathbf{W}\big) - \mathrm{tr}\big(\mathbf{B}^\top \mathbf{W}\big) \tag{11}$$

is exactly the solution $\mathbf{K}^{-1}\mathbf{B}$ of the linear system $\mathbf{KW} = \mathbf{B}$.

Given a partition of indices $\{I_1, I_2, \cdots, I_m\}$ where $I_i \cap I_j = \emptyset$ for all $i \neq j$ and $\cup_{i=1}^m I_i = [n]$, coordinate descent minimizes (11) by minimizing over a subset of variables $\mathbf{W}_I = \mathbf{W}_{I,:}$ in each iteration. Taking the derivative *w.r.t.* the subblock $\mathbf{W}_I$, we have

$$\begin{aligned}[\nabla h(\mathbf{W})]_I &= \mathbf{K}_I\mathbf{W} - \mathbf{B}_I \\ &= \begin{pmatrix}\mathbf{K}_{I,I} & \mathbf{K}_{I,\neg I}\end{pmatrix}\begin{pmatrix}\mathbf{W}_I \\ \mathbf{W}_{\neg I}\end{pmatrix} - \mathbf{B}_I,\end{aligned}$$

where the second line splits $\mathbf{K}_I$ and $\mathbf{W}$ into two blocks. The index $\neg I = [n] \setminus I$ denotes the complement of I. Setting the derivative to zero gives the following update

$$\mathbf{W}_I^{(j+1)} = \mathbf{K}_{I,I}^{-1}\big(\mathbf{B} - \mathbf{K}_{I,\neg I}\mathbf{W}_{\neg I}^{(j)}\big)$$

which minimizes (11) over $\mathbf{W}_I$ exactly. The full algorithm of coordinate descent is shown in Algorithm 2.

The following lemma shows the $\mathbf{R}$ matrix in Algorithm 1 is indeed the residual of the linear system. This lemma will be useful in proving the equivalence between Algorithm 1 and Algorithm 2.

Lemma 1. *Let $\mathbf{R}^{(j)}$ and $\mathbf{W}^{(j)}$ be the residual and weight after j updates of Algorithm 1. Then, we have*

$$\mathbf{R}^{(j)} = \mathbf{B} - \mathbf{K}\mathbf{W}^{(j)}.$$

Proof. The proof is based on an induction on the number of updates j (the number of inner loops). At the initialization $j = 0$, the equality holds trivially. Suppose after the j-th update we have $\mathbf{R}^{(j)} = \mathbf{B} - \mathbf{K}\mathbf{W}^{(j)}$. All we need to do is to verify this equality in the case of $j + 1$ by direct calculation:

$$\begin{aligned}
\mathbf{B} - \mathbf{K}\mathbf{W}^{(j+1)} &= \mathbf{B} - \mathbf{K}\big(\mathbf{W}^{(j)} + \mathbf{E}_I^\top \mathbf{K}_{I,I}^{-1}\mathbf{E}_I\mathbf{R}^{(j)}\big) \\
&= \mathbf{R}^{(j)} - \mathbf{K}\mathbf{E}_I^\top \mathbf{K}_{I,I}^{-1}\mathbf{E}_I\mathbf{R}^{(j)} \\
&= \mathbf{R}^{(j+1)}
\end{aligned}$$

where the first line uses the update rule (8) of $\mathbf{W}^{(j)}$ and the last line uses the update rule (7) of $\mathbf{R}^{(j)}$. □

With Lemma 1, we are ready to show the equivalence between Algorithm 1 and Algorithm 2.

Lemma 2. *Let $\mathbf{W}^{(j)}$ be the weight produced by Algorithm 1 after j updates. Them, we have*

$$\begin{aligned}
\mathbf{W}_I^{(j+1)} &= \mathbf{K}_{I,I}^{-1}\big(\mathbf{B}_I - \mathbf{K}_{I,\neg I}\mathbf{W}_{\neg I}^{(j)}\big), \\
\mathbf{W}_{\neg I}^{(j+1)} &= \mathbf{W}_{\neg I}^{(j)},
\end{aligned}$$

where $\neg I = [n] \setminus I$. Thus, Algorithm 1 produces the same iterates as Algorithm 2.

Proof. Recalling the update rule (8), we have

$$\mathbf{W}^{(j+1)} = \mathbf{W}^{(j)} + \mathbf{E}_I^\top \mathbf{K}_{I,I}^{-1}\mathbf{E}_I\mathbf{R}^{(j)}.$$

Recalling the property of left multiplication with $\mathbf{E}_I^\top$, entries outside I are unchanged and thus $\mathbf{W}_{\neg I}^{(j+1)} = \mathbf{W}_{\neg I}^{(j)}$.

On the other hand, entries indexed by I satisfy $\mathbf{W}_I^{(j+1)} = \mathbf{W}_I^{(j+1)} + \mathbf{K}_{I,I}^{-1}\mathbf{E}_I\mathbf{R}^{(j)}$. Plug in $\mathbf{R}^{(j)} = \mathbf{B} - \mathbf{K}\mathbf{W}^{(j)}$ by Lemma 1 and thus we have

$$\begin{aligned}
\mathbf{W}_I^{(j+1)} &= \mathbf{W}_I^{(j)} + \mathbf{K}_{I,I}^{-1}\mathbf{E}_I\big(\mathbf{B} - \mathbf{K}\mathbf{W}^{(j)}\big) \\
&= \mathbf{W}_I^{(j)} + \mathbf{K}_{I,I}^{-1}\big(\mathbf{B}_I - \mathbf{K}_I\mathbf{W}^{(j)}\big) \\
&= \mathbf{W}_I^{(j)} + \mathbf{K}_{I,I}^{-1}\big(\mathbf{B}_I - \mathbf{K}_{I,I}\mathbf{W}_I^{(j)} - \mathbf{K}_{I,\neg I}\mathbf{W}_{\neg I}^{(j)}\big) \\
&= \mathbf{K}_{I,I}^{-1}\big(\mathbf{B}_I - \mathbf{K}_{I,\neg I}\mathbf{W}_{\neg I}^{(j)}\big)
\end{aligned}$$

where the second line uses the definition of $\mathbf{E}_I$; the third line split the matrix $\mathbf{K}_I$ into blocks $\mathbf{K}_I = \begin{pmatrix}\mathbf{K}_{I,I} & \mathbf{K}_{I,\neg I}\end{pmatrix}$; the last line is straightforward algebra. □

C Proofs

Lemma 3. *The quadratic objective function* (11) *satisfies the Polyak-Łojasiewicz (PL) inequality*

$$\frac{1}{2}\|\nabla h(\mathbf{W})\|_{\mathrm{F}}^2 \geq \lambda_{\min}(h(\mathbf{W}) - h(\mathbf{W}^*)),$$

where $\lambda_{\min} > 0$ is the smallest eigenvalue of $\mathbf{K}$.

Proof. If $\mathbf{W}$ has only a single column this follows directly from the strong convexity of the quadratic function. When $\mathbf{W}$ has multiple columns, h is a separable function across each column. Therefore, h is also $\lambda_{\min}$ strongly convex which implies the PL inequality. □

Lemma 4. *For $h(\mathbf{W})$ as in (11), it holds that $h(\mathbf{W}) - h(\mathbf{W}^*) = \frac{1}{2}\|\mathbf{W} - \mathbf{W}^*\|_{\mathbf{K}}^2$.*

Proof. Plugging $\mathbf{B} = \mathbf{K}\mathbf{W}^*$ into the expression of h, straightforward algebra gives

$$\begin{aligned} h(\mathbf{W}) - h(\mathbf{W}^*) &= \frac{1}{2}\langle \mathbf{W}, \mathbf{K}\mathbf{W}\rangle - \langle \mathbf{B}, \mathbf{W}\rangle - \frac{1}{2}\langle \mathbf{W}^*, \mathbf{K}\mathbf{W}^*\rangle + \langle \mathbf{B}, \mathbf{W}^*\rangle \\ &= \frac{1}{2}\langle \mathbf{W}, \mathbf{K}\mathbf{W}\rangle - \langle \mathbf{K}\mathbf{W}^*, \mathbf{W}\rangle - \frac{1}{2}\langle \mathbf{W}^*, \mathbf{K}\mathbf{W}^*\rangle + \langle \mathbf{K}\mathbf{W}^*, \mathbf{W}^*\rangle \\ &= \frac{1}{2}\langle \mathbf{W}, \mathbf{K}\mathbf{W}\rangle - \langle \mathbf{K}\mathbf{W}^*, \mathbf{W}\rangle + \frac{1}{2}\langle \mathbf{W}^*, \mathbf{K}\mathbf{W}^*\rangle \\ &= \frac{1}{2}\|\mathbf{W} - \mathbf{W}^*\|_{\mathbf{K}}^2. \end{aligned}$$

□

Theorem 1. *Let $\mathbf{W}^*$ be the unique solution of the linear system $\mathbf{K}\mathbf{W} = \mathbf{B}$ and $\mathbf{W}^{(t)}$ its approximation after t epochs of Algorithm 1 using the modified GS rule (9). Then it holds that*

$$\|\mathbf{W}^{(t)} - \mathbf{W}^*\|_{\mathbf{K}}^2 \leq \exp\big(-t/\kappa'\big)\|\mathbf{W}^{(0)} - \mathbf{W}^*\|_{\mathbf{K}}^2,$$

where $\|\mathbf{W} - \mathbf{W}^*\|_{\mathbf{K}}^2 = \mathrm{tr}\big((\mathbf{W} - \mathbf{W}^*)^\top \mathbf{K}(\mathbf{W} - \mathbf{W}^*)\big)$ and $\kappa' = \lambda'_{\max}/\lambda_{\min} \leq \kappa$.

Proof. By straightforward algebra, the improvement on the objective h as in (11) after the j-th update is

$$h(\mathbf{W}^{(j+1)}) - h(\mathbf{W}^{(j)}) = -\frac{1}{2}\big\|\mathbf{R}_{I,:}^{(j)}\big\|_{\mathbf{K}_{I,I}^{-1}}^2.$$

For any residual $\mathbf{R}$ matrix, note the following inequality

$$\|\mathbf{R}\|_{\mathrm{F}}^2 = \sum_{I\in\mathcal{P}} \|\mathbf{R}_{I,:}\|_{\mathrm{F}}^2 \leq |\mathcal{P}| \cdot \max_{I\in\mathcal{P}} \|\mathbf{R}_{I,:}\|_{\mathrm{F}}^2 = m \cdot \max_{I\in\mathcal{P}} \|\mathbf{R}_{I,:}\|_{\mathrm{F}}^2. \tag{12}$$

Thus, the improvement on the objective h is bounded by

$$\begin{aligned} h(\mathbf{W}^{(j+1)}) - h(\mathbf{W}^{(j)}) &\leq -\frac{1}{2\lambda'_{\max}}\|\mathbf{R}_{I,:}^{(j)}\|_{\mathrm{F}}^2 \\ &\leq -\frac{1}{2m\lambda'_{\max}}\|\mathbf{R}^{(j)}\|_{\mathrm{F}}^2 \end{aligned}$$

where the first inequality is because $\frac{1}{\lambda'_{\max}}$ is the smallest eigenvalue of $\mathbf{K}_{I,I}$; the second inequality is due to the Gauss-Southwell selection rule and (12). Subtract $h^* = h(\mathbf{W}^*)$ from both sides. Then, we have

$$\begin{aligned} h(\mathbf{W}^{(j+1)}) - h^* &= h(\mathbf{W}^{(j)}) - h^* - \frac{1}{2m\lambda'_{\max}}\|\mathbf{R}^{(j)}\|_{\mathrm{F}}^2 \\ &\leq \big(1 - \frac{\lambda_{\min}}{m\lambda'_{\max}}\big)\big(h(\mathbf{W}^{(j)}) - h^*\big) \\ &\leq \big(1 - \frac{1}{m\kappa'}\big)\big(h(\mathbf{W}^{(j)}) - h^*\big) \end{aligned}$$

where the second line uses $\mathbf{R}^{(j)} = \mathbf{B} - \mathbf{K}\mathbf{W}^{(j)} = -\nabla h(\mathbf{W}^{(j)})$ by Lemma 1 and the PL inequality by Lemma 3. Using the inequality $(1-x)^t \leq \exp(-tx)$, we obtain a convergence rate in the number of updates j:

$$h(\mathbf{W}^{(j+1)}) - h^* \leq \exp\Big(-\frac{j}{m\kappa'}\Big)\big(h(\mathbf{W}^{(0)}) - h^*\big).$$

Since each epoch has m updates, the convergence rate in the number of epochs t is

$$h(\mathbf{W}^{(t+1)}) - h^* \leq \exp\Big(-\frac{t}{\kappa'}\Big)\big(h(\mathbf{W}^{(0)}) - h^*\big).$$

By Lemma 4, the left and right hand sides can be written as $\|\mathbf{W}^{(t)} - \mathbf{W}^*\|_{\mathbf{K}}^2$ and $\|\mathbf{W}^{(0)} - \mathbf{W}^*\|_{\mathbf{K}}^2$ respectively, which concludes the proof.

□

D Descriptions of the UCI Datasets in the Experiments

This section lists the relevant information of the datasets with citations. The datasets used in the papers are sgemm GPU (Paredes and Ballester-Ripoll, 2018), air quality (Chen, 2019), 3droad (Kaul, 2013), song (Bertin-Mahieux, 2011), buzz (Yang et al., 2015), house electric (Hebrail and Berard, 2012), and gas sensors (Fonollosa, 2015). All of them are downloaded from the UCI machine learning repository (Kelly et al., 2023).

E Additional Experiments

This section presents more experimental details and additional experiments.

E.1 Further Experimental Details

GP Training. All Gaussian processes, including stochastic variational Gaussian processes, use an observation noise constraint $\sigma^2 \geq 10^{-4}$, which is the default in GPyTorch. For the stochastic trace estimation (2), we use $\ell = 15$ random probe vectors. For CG, the probe vectors are sampled from $\mathcal{N}(\mathbf{0}, \mathbf{P})$, where $\mathbf{P}$ is the pivoted Cholesky preconditioner. Again, these settings are default in GPyTorch. For alternating projection, the probe vectors are sampled from a Rademacher distribution.

Preconditioning. CG uses the pivoted Cholesky preconditioner both in training and test. During training, the preconditioner size is 500 on sgemm, air quality, 3droad, song and buzz; 300 on house electric; 150 on gas sensors. We decrease the preconditioner size on house electric and gas sensors due to GPU memory overflow. During test, the preconditioner size is 500 on sgemm, air quality, 3droad, song, buzz and house electric; 300 on gas sensors. Again, we decrese the preconditioner size on gas sensors due to GPU memory flow. See Table 3.

Alternating Projection Batch Size. The batch sizes during training and test are shown in Table 3.

SVGP Training. All SVGPs use 1024 inducing points and a batch size of 4096. On the first six datasets, SVGPs are trained with 50 epochs of Adam with a step size 0.01 and another 150 epochs of Adam with a step size 0.001. On gas sensors, we train the SVGP with 50 epochs of Adam with a step size 0.01 followed by 350 epochs of Adam with a step size 0.001.

Other Experimental Settings. The right panel of Figure 2 is produced on with an alternating projection-trained GP on air quality with batch size 1000. The linear system solved in the figure is $\mathbf{K}^{-1}\mathbf{y}$ (without subtracting the prior mean $\boldsymbol{\mu}$). Figure 3 is plotted with an alternating projection-trained GP on 3droad. The linear system in the figure is $\mathbf{K}^{-1}(\mathbf{y} \quad \mathbf{z}_1 \quad \mathbf{z}_2 \quad \cdots \quad \mathbf{z}_{15})$ where $\mathbf{z}_i$ are sampled from a standard Gaussian distribution.

Table 3: Preconditioner sizes and batch sizes during training and test.

method	train/test	sgemm	air quality	3droad	song	buzz	house electric	gas sensors
CG preconditioner size	train	500	500	500	500	500	300	150
	test	500	500	500	500	500	500	300
alt. proj. batch size	train	6000	6000	6000	4000	4000	1000	500
	test	6000	6000	6000	4000	4000	1000	500

E.2 GP Training on House Electric with Large Noise Constraint $\sigma^2 \geq 0.1$

We compare Gaussian processes on house electric trained with two different noise constraints $\sigma^2 \geq 0.1$ and $\sigma^2 \geq 10^{-4}$, as shown Table 4. We observe significant improvements on both RMSE and NLL when the noise is smaller. In particular, the GP trained with small noise constraint $\sigma^2 \geq 10^{-4}$ has 40% smaller RMSE and significantly smaller NLL. This indicates that artificially inflating the observation noise σ^2, while making the kernel matrix well-conditioned, ultimately hurts the predictive performance.

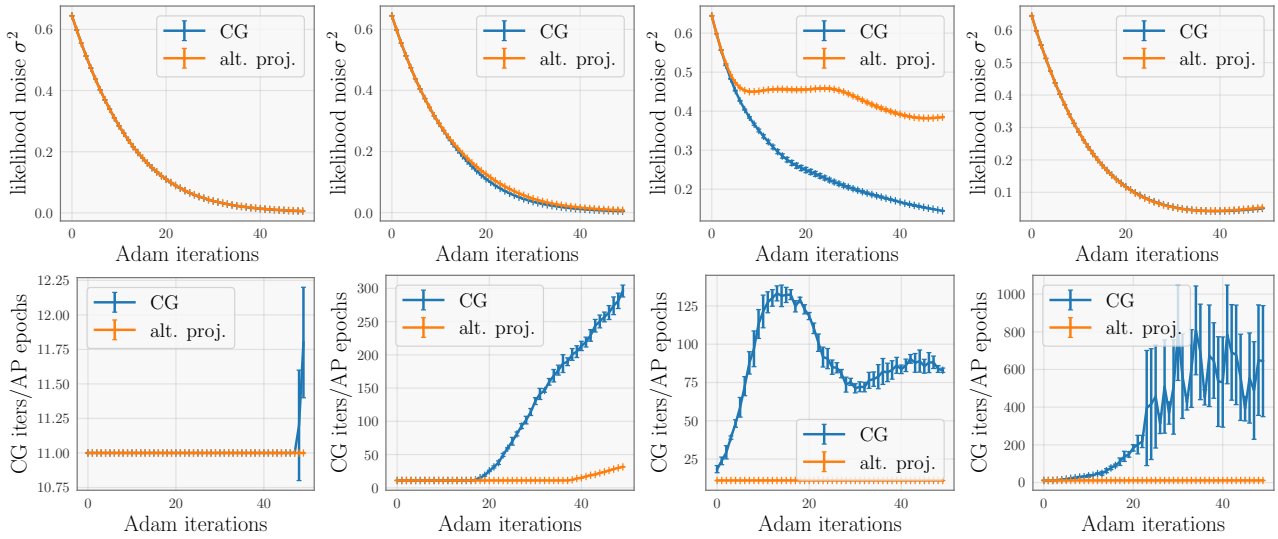
With alternating projection, training the GP with a small noise constraint $\sigma^2 \geq 10^{-4}$ is as fast as the GP with a large noise constraint $\sigma^2 \geq 10^{-1}$. The RMSE and NLL are computed with the same settings as the main paper.

Table 4: Comparison of GP training on the house electric dataset with large noise constraint $\sigma^2 \geq 0.1$ and small noise constraint $\sigma^2 \geq 10^{-4}$.

Dataset	Method	RMSE	NLL	CG iterations/AP epochs	Time
house electric	CG $(\sigma^2 \geq 10^{-1})$	0.050 ± 0.000	-0.196 ± 0.000	1200 ± 8	9.6h $\pm$0.6
$n = 2,049,280$	Alt. Proj. $(\sigma^2 \geq 10^{-1})$	0.053 ± 0.000	-0.197 ± 0.000	1100 ± 0	9.8h $\pm$0.4
$d = 11$	Alt. Proj. $(\sigma^2 \geq 10^{-4})$	$\mathbf{0.030 \pm 0.000}$	$\mathbf{-1.148 \pm 0.001}$	1100 ± 0	9.8h $\pm$0.4

E.3 CG Iterations During Training

Figure 4 in the main paper is produced on air quality. This section presents figures on more datasets, as shown in Figure 6. We observe a similar phenomenon: as the noise decreases during training, the number of CG iterations increases; in contrast, alternating projection converges steadily.

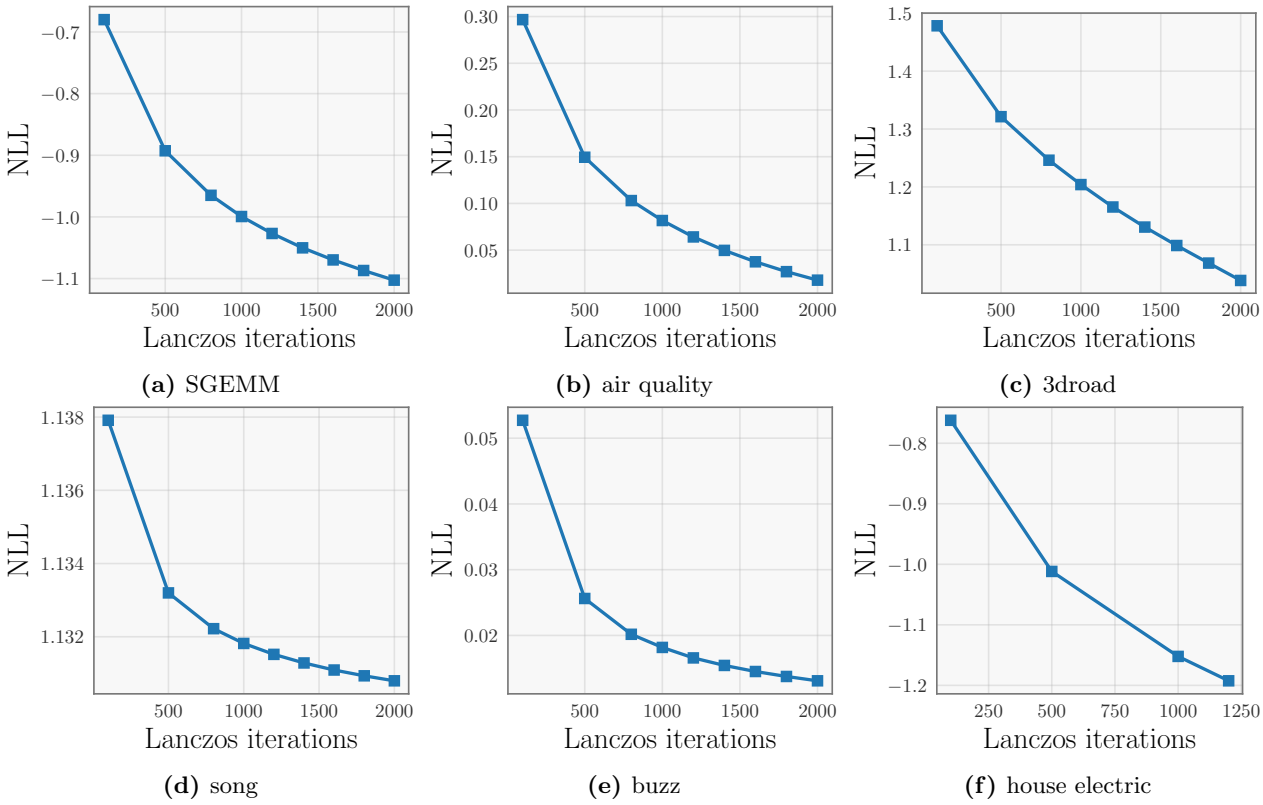


(a) sgemm GPU **(b)** 3droad **(c)** song **(d)** buzz

Figure 6: The observation noise σ^2 and the number of CG iterations/alternating projection epochs during training. **Top:** The observation noisea σ^2 decreases as the training goes. **Bottom:** CG takes more iterations to converge as the observation noise decreases during training. However, alternating projection is less sensitive to the decrease of observation noise.

E.4 Increasing Lanczos Iterations Improves NLL

In the experiments, we use 1000 Lanczos iterations to compute the predictive variance and the test negative log likelihood (NLL). This section investigates the relation between test NLL and the Lanczos iterations, as shown in Figure 7. We empirically observe that increasing the Lanczos iterations always decreases the test NLL. This suggests that the true NLL of the GPs may be even lower than what is reported in Table 5.



(a) SGEMM **(b)** air quality **(c)** 3droad

(d) song **(e)** buzz **(f)** house electric

Figure 7: Test negative log likelihood (NLL) *vs*. the number of Lanczos iterations. Empirically, the test NLL decreases as the number of Lanczos iterations increases on all datasets.

E.5 Training Gaussian Processes with Matérn $\nu = 1.5$

Lastly, we report results using Matérn $\nu = 1.5$. The experimental settings are exactly the same as Matérn $\nu = 2.5$ GPs. We observe a similar phenomenon: while CG-trained GPs and alternating projection-trained GPs have similar RMSE and NLL, alternating projection achieves 1.4× to 27.2× speed up against CG.

F FLOPs in Algorithm 1

The following table gives floating point operations (FLOPs) and memory complexity of Algorithm 1. There is no hidden constant in the leading term. Throughout, we assume $l \ll n$ and $1 \ll b \ll n$. Note that the peak memory consumption is $2nb$. We use this to estimate the largest batch b that fits in a GPU.

G Additional Discussions

The alternating projection method presented in this paper is not easy to parallel on multiple GPUs. Indeed, the update for each block is sequential. When multiple GPUs are available, CG might be more beneficial as explored by Wang et al. (2019). Another limitation of alternating projection is that it does not yield an estimate of the marginal log-likelihood (MLL). Therefore, one cannot monitor the convergence progress by plotting the MLL. A workaround is to instead monitor the observation noise σ^2. Typically, the observation noise σ^2 diminishes during training, and a small update in σ^2 is usually a good indication of convergence.

Artemev et al. (2021) utilize CG to construct a better variational lower bound for variational GPs (Titsias, 2009). Different from the stochastic variational GP (Hensman et al., 2013), this method cannot be trained by mini-batch stochastic optimization, since they plug in the closed-form solution of the variational distribution. Interestingly, they show that warming up CG for the linear solve $\mathbf{K}^{-1}\mathbf{y}$ yield a significant speed-up. This trick might be useful in CG-based and alternating projection-based training as well.

Table 5: Gaussian process training on UCI benchmark datasets with Matérn $\nu = 1.5$. Metrics are computed across multiple runs and reported with ± one standard deviation.

Dataset	Method	RMSE	NLL	CG iters/AP epochs	Training time	Speed up
SGEMM	CG	$\mathbf{0.048 \pm 0.000}$	$\mathbf{-1.071 \pm 0.001}$	550 ± 0	8.9m ±0.2	
$n = 241,600$	Alt. Proj.	$\mathbf{0.048 \pm 0.000}$	-1.060 ± 0.001	550 ± 0	12.1m ±0.2	0.7×
$d = 14$	SVGP	0.085 ± 0.000	-0.932 ± 0.001	NA	18.3m ±0.1	
air quality	CG	$\mathbf{0.227 \pm 0.002}$	0.131 ± 0.003	1825 ± 26	22.5m ±1.2	
$n = 382,168$	Alt. Proj.	0.253 ± 0.001	$\mathbf{0.033 \pm 0.002}$	550 ± 0	16.1m ±0.5	1.4×
$d = 13$	SVGP	0.358 ± 0.002	0.387 ± 0.005	NA	28.8m ±0.1	
3droad	CG	$\mathbf{0.065 \pm 0.001}$	1.062 ± 0.003	6086 ± 142	44.4m ±2.2	
$n = 434,874$	Alt. Proj.	0.069 ± 0.001	0.896 ± 0.002	572 ± 1	16.5m ±0.3	2.7×
$d = 3$	SVGP	0.319 ± 0.002	$\mathbf{0.294 \pm 0.007}$	NA	32.4m ±0.1	
song	CG	$\mathbf{0.743 \pm 0.001}$	1.135 ± 0.003	4393 ± 159	13.7h ±0.6	
$n = 515,345$	Alt. Proj.	$\mathbf{0.746 \pm 0.002}$	$\mathbf{1.129 \pm 0.002}$	550 ± 0	2.6h ±0.0	5.3×
$d = 90$	SVGP	0.790 ± 0.002	1.184 ± 0.002	NA	0.6h ±0.0	
buzz	CG	$\mathbf{0.238 \pm 0.000}$	0.027 ± 0.002	13608 ± 2299	25.4h ±4.7	
$n = 583,250$	Alt. Proj.	$\mathbf{0.238 \pm 0.001}$	$\mathbf{0.002 \pm 0.004}$	550 ± 0	1.9h ±0.1	13.4×
$d = 77$	SVGP	0.255 ± 0.002	0.049 ± 0.009	NA	0.7h ±0.0	
house electric	CG	-	-	-	$\geqslant$ 11d	
$n = 2,049,280$	Alt. Proj.	$\mathbf{0.029 \pm 0.000}$	-1.321 ± 0.000	1100 ± 0	9.7h ±0.1	$\geqslant$ 27.2×
$d = 11$	SVGP	0.048 ± 0.000	$\mathbf{-1.580 \pm 0.003}$	NA	2.6h ±0.0	
gas sensors	CG	-	-	-	-	
$n = 4,178,504$	Alt. Proj.	**0.201**	$\mathbf{0.245}^{\dagger}$	1100	42h*	
$d = 17$	SVGP	0.311 ± 0.002	0.286 ± 0.004	NA	10.6h ±0.1	

†: This predictive variance is calculated using only 500 Lanczos iterations to save time and avoid numerical instability.
*: Time measured on a A100 GPU.

Table 6: FLOP Counting in Algorithm 1.

Operation	FLOPs	Memory
Cache Cholesky decomposition of $\{\mathbf{K}_{I,I} : I \in \mathcal{P}\}$	$\frac{1}{3}nb^2$	nb
GS rule $I = \text{argmax}_{I \in \mathcal{P}} \ \|\mathbf{R}_{I,:}\|_{\text{F}}^2$	$2nl$	-
$\mathbf{W}_I = \mathbf{W}_I + \mathbf{K}_{I,I}^{-1}\mathbf{R}_I$	$(b^2 + b)l$	-
$\mathbf{R} = \mathbf{R} - \mathbf{K}_{:,I}\mathbf{K}_{I,I}^{-1}\mathbf{R}_I$	$(b^2 + 2nb + n)l$	nb
total FLOPs of a single epoch	$\big((2 + \frac{3}{b})n^2 + (2b + 1)n\big)l$	$2nb$