Posterior and Computational Uncertainty in Gaussian Processes

Anonymous Author(s)
Affiliation
Address
email

Abstract

Gaussian processes scale prohibitively with the size of the dataset. In response, many approximation methods have been developed, which inevitably introduce approximation error. This additional source of uncertainty, due to limited computation, is entirely ignored when using the approximate posterior. Therefore in practice, GP models are often as much about the approximation method as they are about the data. Here, we develop a new class of methods that provides consistent estimation of the combined uncertainty arising from *both* the finite number of data observed *and* the finite amount of computation expended. The most common GP approximations map to an instance in this class, such as methods based on the Cholesky factorization, conjugate gradients, and inducing points. For any method in this class, we prove (i) convergence of its posterior mean in the associated RKHS, (ii) decomposability of its combined posterior covariance into mathematical and computational covariances, and (iii) that the combined variance is a tight worst-case bound for the squared error between the method's posterior mean and the latent function. Finally, we empirically demonstrate the consequences of ignoring computational uncertainty and show how it improves generalization performance on benchmark datasets.

1 Introduction

Gaussian processes (GPs) are a principled and expressive probabilistic model class, but their prohibitive scaling necessitates approximation. A range of GP approximations based on kernel [1–9] or precision matrix [10–13] estimates, inducing point methods [14–21], and iterative solvers [22–28] have been proposed. These methods all use an affordable amount of computation to obtain an approximation of the *mathematical* posterior, which exists theoretically but cannot be accessed given the limited computational resources. The approximate posterior is then used as a direct replacement of the mathematical posterior in downstream applications. Doing so, however, completely ignores the fact that we only expended a limited amount of computation. By analogy to the typical GP operation – where *limited data* induces modeling error that is captured by *mathematical uncertainty* – our work is motivated by the fact that *limited computation* induces approximation error that must be captured by *computational uncertainty*.

Here, we introduce IterGP, a class of methods which return a *combined uncertainty* that is the sum of mathematical and computational uncertainty. Figure 1 illustrates the difference between ignoring computational uncertainty and explicitly modeling it. We perform GP regression using a Matérn($\frac{3}{2}$) kernel on a toy dataset and compare SVGP (🔴) [21] to its analog, IterGP-PI, in our framework (🟢 + 🔵) for a fixed set of inducing points. The computational shortcuts of inducing point methods lead to unavoidable biases in their posterior mean and covariance [29]. As Figure 1 illustrates, SVGP may underestimate the marginal variance where inducing points do not coincide with datapoints. In contrast, IterGP is guaranteed to overestimate the mathematical uncertainty – with the difference

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

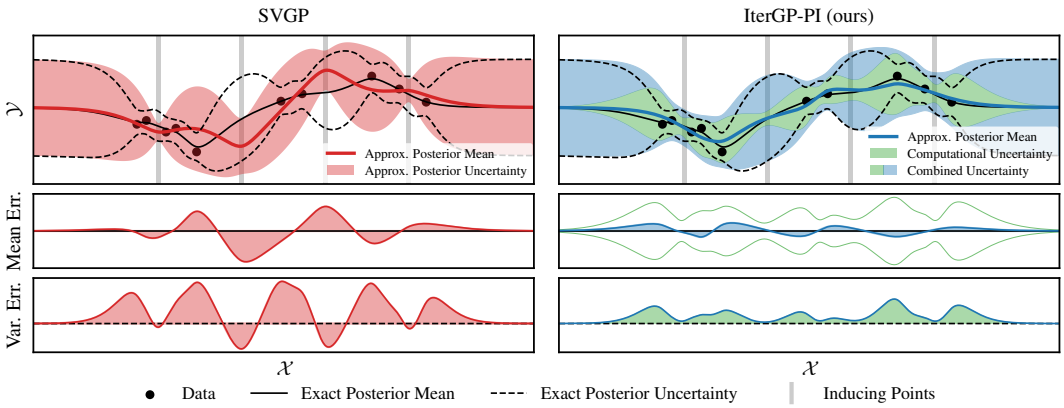


Figure 1: *Modeling computational uncertainty improves GP approximation.*

precisely given by the computational uncertainty (●). Additionally, the computational uncertainty is a worst-case bound (—) on the error of the approximate posterior mean.

To be clear, this overestimation is desirable: IterGP is not a typical approximation in the sense that its combined posterior attempts to approximate the mathematical posterior. Rather, IterGP recognizes that the mathematical posterior exists, but we do not have access to it, given computational constraints. Indeed, finite computation is as true a source of posterior uncertainty as finite data. Taking this view seriously, the true goal of GPs in the limited compute regime should in fact be to track combined uncertainty. This intuition motivates IterGP and is formally a feature of our results. We show that, if you update your GP via computation, specifically matrix-vector multiplication, then the combined uncertainty of the IterGP algorithm is precisely the correct object to capture your belief (Theorem 2).

Formally, IterGP treats the (unknown) representer weights as a latent variable with a prior belief that, when marginalized out, corresponds to a GP prior conditioned on no data. We then use a computational primitive (matrix-vector multiplication) that corresponds to tractable Bayesian updates on the representer weight distribution; that is, conditioning on computations performed on the data. The resulting belief can then be marginalized out to obtain a closed form, tractable expression for the combined – mathematical plus computational – uncertainty. This uncertainty quantification can be done *exactly* in quadratic time and linear space complexity.

Our framework admits three key theoretical properties. First, common GP approximations such as the partial Cholesky, the method of conjugate gradients and inducing point methods (SoR, DTC / Nystrom, SVGP) map to a corresponding IterGP instance. Therefore, pleasingly, these approaches can either be directly extended or modified to properly account for computational uncertainty. Second, the approximate posterior mean computed by any method in our proposed class satisfies a general convergence result in the associated reproducing kernel Hilbert space (RKHS), which only depends on how information about the kernel matrix is obtained in each iteration (Theorem 1). Third, the combined uncertainty is a tight worst case bound on the relative error between the approximate posterior mean and the latent function (Theorem 2). To the best of our knowledge no existing GP approximation has this last property; an analoguous guarantee only holds for exact GPs [30].

Contribution In summary, this work introduces IterGP, which includes:

(a) *a new class of GP approximations* that accounts for computational uncertainty. Some IterGP instances extend classic methods with improved uncertainty estimation.

(b) *a general convergence result for the approximate posterior mean* (Theorem 1)

(c) *an interpretation of the combined uncertainty as a tight worst-case bound* on the relative distance to the latent function (Theorem 2).

(d) *empirical quantification of computational uncertainty, improving generalization* or saving computation on benchmark datasets.

2 Computation-Aware Gaussian Process Inference

We aim to learn a latent function $h : \mathcal{X} \to \mathcal{Y}$ from $\mathcal{X} \subseteq \mathbb{R}^d$ to $\mathcal{Y} \subseteq \mathbb{R}$ given a training dataset $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{R}^{n \times d}$ of n inputs $\boldsymbol{x}_j \in \mathbb{R}^d$ and corresponding outputs $\boldsymbol{y} = (y_1, \ldots, y_n)^\intercal \in \mathbb{R}^n$.

Gaussian Processes A stochastic process $f \sim \mathcal{GP}(\mu, k)$ with mean function $\mu : \mathbb{R}^d \to \mathbb{R}$ and kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a *Gaussian process* (GP) if any collection of function values $\mathbf{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$ is jointly Gaussian with $\boldsymbol{\mu}_j = \mu(\boldsymbol{x}_j)$ and $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Assuming observation noise $\boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 \boldsymbol{I})$, the posterior distribution at test inputs $\boldsymbol{x}_\star$ is given by $\mathbf{f}_\star \sim \mathcal{N}(\mu_*(\boldsymbol{x}_\star), k_*(\boldsymbol{x}_\star, \boldsymbol{x}_\star))$ where the posterior mean and covariance functions are given by

$$\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X}) \overbrace{\hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})}^{\boldsymbol{v}_*}, \quad \text{and} \quad k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \hat{\boldsymbol{K}}^{-1} k(\boldsymbol{X}, \cdot) \tag{1}$$

where $\hat{\boldsymbol{K}} \coloneqq \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$. Computing the *representer weights* $\boldsymbol{v}_* = \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$ exactly (as well as the posterior variance) is prohibitive given our limited computational budget.

Learning Representer Weights Consider the conditional distribution of the latent GP given its representer weights:

$$p(\mathbf{f}_\star \mid \boldsymbol{v}_*) = \mathcal{N}(k(\boldsymbol{x}_\star, \boldsymbol{X}) \boldsymbol{v}_*, \;\; k_*(\boldsymbol{x}_\star, \boldsymbol{x}_\star)). \tag{2}$$

When $\boldsymbol{v}_*$ is known exactly, we recover eq. (1). However, if we instead treat $\boldsymbol{v}_*$ as a random variable with the prior $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{0}, \hat{\boldsymbol{K}}^{-1})$, then the resulting marginal $\int p(\mathbf{f}_\star \mid \boldsymbol{v}_*) p(\boldsymbol{v}_*) \, d\boldsymbol{v}_*$ recovers the GP prior $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$. Our goal is to update this prior by iteratively applying the tractable computational primitive (i.e. matrix-vector multiplies). More specifically, each iteration conditions the current belief distribution $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_{i-1}, \boldsymbol{\Sigma}_{i-1})$ on a one-dimensional projection of the current *residual* $\boldsymbol{r}_i = (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}} \boldsymbol{v}_{i-1}$, where the projection is defined by an arbitrary vector $\boldsymbol{s}_i$:

$$\alpha_i \coloneqq \boldsymbol{s}_i^\intercal \boldsymbol{r}_{i-1} = \boldsymbol{s}_i^\intercal ((\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}} \boldsymbol{v}_{i-1}) = \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} (\boldsymbol{v}_* - \boldsymbol{v}_{i-1}). \tag{3}$$

The choice of *actions* $\boldsymbol{s}_i$, which intuitively weight the approximation error of selected datapoints, defines different instances of our IterGP framework. Computing eq. (3) requires a single matrix-vector multiplication. After computing α_i, we can perform an exact Bayesian update of $p(\boldsymbol{v}_*)$ via linear Gaussian identities. The updated $p(\boldsymbol{v}_*)$ (conditioned on α_i) is $\mathcal{N}(\boldsymbol{v}_* \mid \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$, with

$$\boldsymbol{v}_i = \boldsymbol{v}_{i-1} + \underbrace{\boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i}_{=: \boldsymbol{d}_i} \underbrace{(\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i)}_{=: \eta_i}{}^{-1} \underbrace{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} (\boldsymbol{v}_* - \boldsymbol{v}_{i-1})}_{= \alpha_i} = \boldsymbol{C}_i (\boldsymbol{y} - \boldsymbol{\mu}) \tag{4}$$

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_{i-1} - \underbrace{\boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i}_{= \boldsymbol{d}_i} \underbrace{(\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i)}_{= \eta_i}{}^{-1} \underbrace{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1}}_{= \boldsymbol{d}_i^\intercal} = \hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i. \tag{5}$$

where $\boldsymbol{C}_i \coloneqq \sum_{j=1}^{i} \frac{1}{\eta_j} \boldsymbol{d}_j \boldsymbol{d}_j^\intercal = \boldsymbol{S}_i (\boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{S}_i)^{-1} \boldsymbol{S}_i^\intercal$ is a rank-i matrix (see Proposition S3 for details). With each computation, the uncertainty about the representer weights contracts as $\boldsymbol{C}_i \to \hat{\boldsymbol{K}}^{-1} = \boldsymbol{\Sigma}_0$ as $i \to n$. After n iterations, $\boldsymbol{C}_n = \hat{\boldsymbol{K}}^{-1}$, meaning we fully recovered the representer weights with zero uncertainty. The consistent estimate for the representer weights is consequently $\boldsymbol{v}_i = \boldsymbol{C}_i (\boldsymbol{y} - \boldsymbol{\mu})$.

Combining Mathematical and Computational Uncertainty We now have a belief $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$ over the representer weights which reflects the computation expended. To account for this computational uncertainty, we can treat the representer weights as a latent variable of the mathematical posterior by reparameterizing, i.e. $p(\mathbf{f}_\star \mid \boldsymbol{y}) = p(\mathbf{f}_\star \mid \boldsymbol{v}_*)$, and then marginalizing. The resulting marginal considers all possible representer weights which would have resulted in the same computational observations and therefore *implicitly* adds the uncertainty coming from *the computation itself*. Since the posterior mean of a GP is a linear function of the representer weights, the marginal distribution $p(\mathbf{f}_\star) = \int p(\mathbf{f}_\star \mid \boldsymbol{v}_*) p(\boldsymbol{v}_*) \, d\boldsymbol{v}_* = \mathcal{N}(\mathbf{f}_\star; \boldsymbol{\mu}_\star, \boldsymbol{\Sigma}_\star)$ is given by

$$\begin{aligned} \boldsymbol{\mu}_\star &= \mu(\boldsymbol{x}_\star) + k(\boldsymbol{x}_\star, \boldsymbol{X}) \boldsymbol{v}_i \\ \boldsymbol{\Sigma}_\star &= \underbrace{k(\boldsymbol{x}_\star, \boldsymbol{x}_\star) - k(\boldsymbol{x}_\star, \boldsymbol{X}) \hat{\boldsymbol{K}}^{-1} k(\boldsymbol{X}, \boldsymbol{x}_\star)}_{\text{mathematical uncertainty}} + \underbrace{k(\boldsymbol{x}_\star, \boldsymbol{X}) \boldsymbol{\Sigma}_i k(\boldsymbol{X}, \boldsymbol{x}_\star)}_{\text{computational uncertainty}} \\ &= \underbrace{k(\boldsymbol{x}_\star, \boldsymbol{x}_\star) - k(\boldsymbol{x}_\star, \boldsymbol{X}) \boldsymbol{C}_i k(\boldsymbol{X}, \boldsymbol{x}_\star)}_{\text{combined uncertainty}} \end{aligned} \tag{6}$$

since $\boldsymbol{\Sigma}_i = \hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i$. As we perform more computation, the computational uncertainty reduces and we approach the mathematical uncertainty. We note that, while the *individual* terms are computationally prohibitive, the *combined* uncertainty can be evaluated cheaply since the approximate

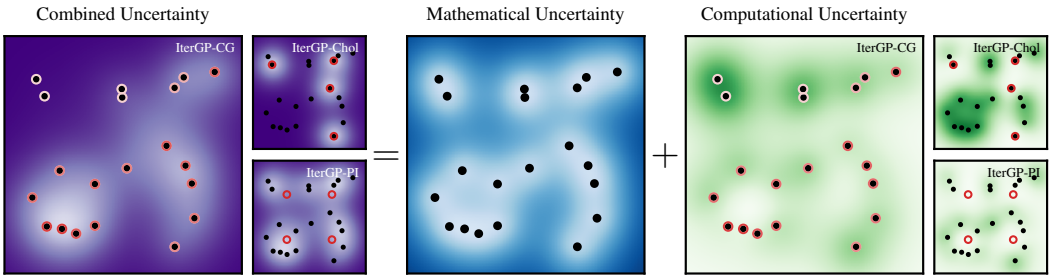


Figure 2: *Decomposition of the combined uncertainty.* The combined uncertainty (●) output by IterGP decomposes into the mathematical uncertainty (●) and computational uncertainty (●). After $i = 4$ iterations of Algorithm 1 computational uncertainty is small in parts of the input space where there either is no data (●) or computation was "targeted" (○). Which datapoints are targeted in each iteration and to what degree is defined by the magnitude of the action vector elements $(\boldsymbol{s}_i)_j$. Different instances of IterGP either reduce computational uncertainty locally (e.g. IterGP-Chol, IterGP-PI) or globally (e.g. IterGP-CG) by . After n iterations the mathematical uncertainty is recovered.

precision matrix $\boldsymbol{C}_i$ is of low rank. Figure 2 illustrates that computational uncertainty is large where there are data and we have not targeted computation yet. Different methods from our proposed class target computation in different parts of the input space. Where there is no data the prior is a good approximation of the posterior and therefore computational uncertainty is low.

Algorithm 1 computes an estimate of the representer weights $\boldsymbol{v}_i$ and the rank-i precision matrix approximation $\boldsymbol{C}_i$. A specific instance of IterGP is defined by a sequence of actions $\boldsymbol{s}_i$. To gain an intuition for how Algorithm 1 operates, it helps to interpret it as targeting a given computational budget towards certain datapoints defined by $\boldsymbol{s}_i$. Near datapoints $\boldsymbol{x}_j$ that are not targeted, i.e. $(\boldsymbol{s}_i)_j = 0$, computational uncertainty remains unchanged. In fact, datapoints $(\boldsymbol{x}_j, y_j)$ that are never targeted up to iteration i are not needed to compute $\mathcal{GP}(\mu_i, k_i)$, meaning that Algorithm 1 is *inherently online* and we can *observe data sequentially* without having to restart the algorithm.

Algorithm 1: A Class of Probabilistic Iterative Methods for GP Approximation

Input: prior mean function μ, prior covariance function / kernel k, training inputs $\boldsymbol{X}$, labels $\boldsymbol{y}$
Output: GP posterior $\mathcal{GP}(\mu_i, k_i)$

procedure ITERGP($\mu, k, \boldsymbol{X}, \boldsymbol{y}$)
  $\boldsymbol{\mu} \leftarrow \mu(\boldsymbol{X})$ ▷ Prior predictive mean.
  $\hat{\boldsymbol{K}} \leftarrow k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}$ ▷ Prior predictive kernel matrix.
  while not STOPPINGCRITERION() **do** ▷ Stopping criterion.
    $\boldsymbol{s}_i \leftarrow$ POLICY() ▷ Select action via policy (see Table 1 for examples).
    $\boldsymbol{r}_{i-1} \leftarrow (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}} \boldsymbol{v}_{i-1}$ ▷ Predictive residual.
    $\alpha_i \leftarrow \boldsymbol{s}_i^\intercal \boldsymbol{r}_{i-1}$ ▷ Observation via information operator.
    $\boldsymbol{d}_i \leftarrow \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i = (\boldsymbol{I} - \boldsymbol{C}_{i-1} \hat{\boldsymbol{K}}) \boldsymbol{s}_i$ ▷ Search direction.
    $\eta_i \leftarrow \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i = \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_i$ ▷ Normalization constant.
    $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_{i-1} + \frac{1}{\eta_i} \boldsymbol{d}_i \boldsymbol{d}_i^\intercal$ ▷ Precision matrix approximation $\boldsymbol{C}_i \approx \hat{\boldsymbol{K}}^{-1}$.
    $\boldsymbol{Q}_i \leftarrow \boldsymbol{Q}_{i-1} + \frac{1}{\eta_i} \hat{\boldsymbol{K}} \boldsymbol{d}_i \boldsymbol{d}_i^\intercal \hat{\boldsymbol{K}}$ ▷ Kernel matrix approximation $\boldsymbol{Q}_i \approx \hat{\boldsymbol{K}}$.
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_{i-1} + \frac{\alpha_i}{\eta_i} \boldsymbol{d}_i$ ▷ Representer weights estimate.
    $\boldsymbol{\Sigma}_i \leftarrow \boldsymbol{\Sigma}_0 - \boldsymbol{C}_i$ ▷ Computational representer weights uncertainty.
  $p(\boldsymbol{v}_*) \leftarrow \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$ ▷ Belief about representer weights.
  $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, \boldsymbol{X}) \boldsymbol{v}_i$ ▷ Approximate posterior mean function.
  $k_i(\cdot, \cdot) \leftarrow k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \boldsymbol{C}_i k(\boldsymbol{X}, \cdot)$ ▷ Combined uncertainty.
  return $\mu_i(\cdot), k_i(\cdot, \cdot), p(\boldsymbol{v}_*)$

2.1 Connection to Other GP Approximation Methods

IterGP extends the most commonly used GP approximations to include computational uncertainty, with at most quadratic cost (see Table 1 for a summary and Figure 2, Figure S2 for illustration).

Table 1: *Instances of Algorithm 1, which map to commonly used GP approximations.*

Method	Actions $\boldsymbol{s}_i$	Classic Analog	Reference
IterGP-Chol	$\boldsymbol{e}_i$	(partial) Cholesky	Theorem S3
IterGP-PBR	$\mathrm{ev}_i(\hat{\boldsymbol{K}})$	(partial) EVD / SVD	Theorem S4
IterGP-CG	$\boldsymbol{s}_i^{\mathrm{PCG}}$ or $\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i$	(preconditioned) CG	Theorem S5 and Corollary S2
IterGP-PI	$k(\boldsymbol{X}, \boldsymbol{x}_i)$	$\approx$ SoR, DTC / Nyström, SVGP	Section 2.1 and Theorem S6

Cholesky Decomposition The (partial) Cholesky decomposition iteratively chooses datapoints or pivots $\boldsymbol{x}_i$ based on a given ordering. The resulting Cholesky factor is lower triangular and increases in rank each iteration, and a well-chosen ordering achieves fast convergence (cf. [31, Thm. 2.3]). If one chooses standard unit vectors $\boldsymbol{e}_i$ as actions (the selected datapoint per iteration), then Algorithm 1 recovers partial Cholesky exactly (Theorem S3).

Conjugate Gradients CG [32] with preconditioning for GP inference has become increasingly popular [23–28, 33, 34]. Algorithm 1 recovers preconditioned CG exactly if we choose either preconditioned conjugate gradients or residuals as actions (see Theorem S5 and Corollary S2). In fact, Algorithm 1 can even construct its own diagonal-plus-low-rank preconditioner by first running a few iterations with an arbitrary policy and then using the byproducts of these iterations for the preconditioner. For example, if we run IterGP-Chol initially, we can construct an incomplete Cholesky preconditioner for subsequent CG iterations.

Inducing Point Methods Inducing point methods such as SoR [14, 35], DTC [17, 36], i.e. the Nyström approximation and SVGP [21] share a posterior mean, which by Theorem S6 takes the form

$$\mu(\cdot) = q(\cdot, \boldsymbol{X})\boldsymbol{K}_{\boldsymbol{XZ}}(\boldsymbol{K}_{\boldsymbol{ZX}}(q(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K}_{\boldsymbol{XZ}})^{-1}\boldsymbol{K}_{\boldsymbol{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}) \tag{7}$$

where $\boldsymbol{Z} \in \mathbb{R}^{n \times i}$ is a set of inducing points and $q(\cdot, \cdot) = k(\cdot, \boldsymbol{Z})\boldsymbol{K}_{\boldsymbol{ZZ}}^{-1}k(\boldsymbol{Z}, \cdot)$. These approximations also have very closely related posterior covariance functions [19, 37]. If we choose actions $\boldsymbol{s}_i = k(\boldsymbol{X}, \boldsymbol{z}_i)$, by Proposition S3, Algorithm 1 returns a posterior mean given by

$$\mu_i(\cdot) = k(\cdot, \boldsymbol{X})\boldsymbol{K}_{\boldsymbol{XZ}}(\boldsymbol{K}_{\boldsymbol{ZX}}(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K}_{\boldsymbol{XZ}})^{-1}\boldsymbol{K}_{\boldsymbol{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}). \tag{8}$$

Choosing such actions reduces computational uncertainty in regions close to inducing points (see IterGP-PI in Figure 2). Recognizing that $q(\boldsymbol{X}, \boldsymbol{X}) \preceq k(\boldsymbol{X}, \boldsymbol{X})$ and interpreting (7) and (8) as (approximate) Bayesian updates on the representer weights explains why SVGP is overconfident in its posterior mean. Roughly speaking, it does not update its representer weight estimates cautiously enough, in favor of linear time complexity. As the inducing points approach the data points, the two posterior mean expressions become closer, and are equivalent if they equal the entire training data.

2.2 The Cost of Computational Uncertainty

Quantifying combined uncertainty has greater cost than linear time GP approximations such as inducing point methods, due to its use of matrix-vector multiplication as the computational operation to condition on the data. Algorithm 1 in its most general form performs three matrix-vector products per iteration resulting in a quadratic time complexity $\mathcal{O}(n^2 i)$ overall for i iterations. In this sense, Algorithm 1 represents a middle ground between the mathematical posterior—which incurs a cubic time complexity—and $\mathcal{O}(ni^2)$ approximations—which can only estimate their computational error through potentially loose theoretical bounds which may [e.g. 20, 21, 38] or may not be computable in less than $\mathcal{O}(n^3)$ [3, 31]. At any point during a run of Algorithm 1, computing the predictive mean on $n_\star$ new data points has cost $\mathcal{O}(n_\star n)$, while the marginal predictive (co-)variance can be evaluated in $\mathcal{O}(n_\star n i)$ since $\boldsymbol{C}_i$ is of rank i. Additionally, using Matheron's rule [39–41], sampling from the approximate posterior at $n_\star$ evaluation points also only requires $\mathcal{O}(n_\star n i)$ computation (assuming we can sample from the prior—see Section S2.5). The objects required to make predictions and draw samples are the vector $\boldsymbol{v}_i$ and low rank matrix $\boldsymbol{C}_i$ which both require $\mathcal{O}(ni)$ memory. Finally, the memory cost of Algorithm 1 is only linear in n, since matrix multiplication $\boldsymbol{v} \mapsto \hat{\boldsymbol{K}}\boldsymbol{v}$ can be computed without explicitly forming $\hat{\boldsymbol{K}}$ [42].

2.3 Related Work

GP inference based on matrix-vector multiplies, particularly CG [32], has become popular recently [4, 23–28, 33]. Advances in specialized hardware has boosted their scalability without excessive memory footprint [26, 42]. Such iterative methods typically rely on preconditioning, which has been shown to significantly improve their performance [24, 25, 28]. Our method generalizes CG in this setting and thus retains the same benefits. At its core Algorithm 1 employs a (Bayesian) probabilistic numerical method [43–45], more specifically a probabilistic linear solver (PLS) [46–51] applied to the linear system $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$. The fact that a PLS using CG actions can recover CG in its posterior mean was observed previously [46, 48, 50]. Here, we extend this result to residual actions and preconditioning. Further, we also demonstrate the connection to the Cholesky and singular value decompositions. For randomized actions, the PLS as part of Algorithm 1 also recovers the randomized Kaczmarz method in its posterior mean [52–55]. Employing a PLS for GP approximation by updating beliefs over the kernel and precision matrix was suggested previously [50, 56]. Our work differs in that it updates a belief over the representer weights, as opposed to the kernel function or matrix, considers more general projections than just conjugate residuals, and, most importantly, provides a theoretically motivated combined posterior which can be computed exactly.

3 Theoretical Analysis

The main goal of our theoretical analysis will be to establish a convergence result for the posterior mean approximation (Theorem 1) and to prove that the combined uncertainty is a tight worst-case bound on the distance to all potential latent functions consistent with our (computational) observations (Theorem 2). At the heart of Algorithm 1 is a probabilistic linear solver [46–48, 50] iteratively updating a belief about the representer weights. It constructs an expanding subspace $\operatorname{span}\{\boldsymbol{s}_1, \dots, \boldsymbol{s}_i\} = \operatorname{span}\{\boldsymbol{d}_1, \dots, \boldsymbol{d}_i\}$ spanned by the actions in which the inverse $\hat{\boldsymbol{K}}^{-1}$ is perfectly identified. Each step $\boldsymbol{d}_i$ expanding this explored subspace is $\hat{\boldsymbol{K}}$-orthogonal to the previous ones.

Proposition 1 (Conjugate Direction Method)
Let the actions $\boldsymbol{s}_i$ of Algorithm 1 be linearly independent. Then Algorithm 1 is a conjugate direction method, i.e. it holds that $\boldsymbol{d}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_j = 0$ for all $i \neq j$.

Proof. Without loss of generality assume $i > j$. Then the result follows directly from Lemma S1. □

Geometrically, Algorithm 1 iteratively projects the representer weights onto the expanding subspace $\operatorname{span}\{\boldsymbol{S}_i\}$ with respect to $\langle \cdot, \cdot \rangle_{\hat{\boldsymbol{K}}}$. We can use this intuition to understand the convergence of the representer weights estimate. The relative error $\rho(i)$ at iteration i is given by how "small" the "angle" between this subspace and the representer weights vector is.

Proposition 2 (Relative Error Bound for the Representer Weights)
For any choice of actions a relative error bound $\rho(i)$, s.t. $\|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\hat{\boldsymbol{K}}} \leq \rho(i) \|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}$ is given by

$$\rho(i) = (\bar{\boldsymbol{v}}_*^\intercal \underbrace{(\boldsymbol{I} - \boldsymbol{C}_i \hat{\boldsymbol{K}})}_{\textit{projection onto } \operatorname{span}\{\boldsymbol{S}_i\}^{\perp_{\hat{\boldsymbol{K}}}}} \bar{\boldsymbol{v}}_*)^{\frac{1}{2}} \leq \lambda_{\max}(\boldsymbol{I} - \boldsymbol{C}_i \hat{\boldsymbol{K}}) \leq 1 \tag{9}$$

where $\bar{\boldsymbol{v}}_* = \boldsymbol{v}_* / \|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}$. If the actions $\{\boldsymbol{s}_i\}_{i=1}^n$ are linearly independent, then $\rho(i) \leq \delta_{n=i}$.

Proof. See Section S2.2. □

Proposition 2 guarantees convergence in at most n steps, if the actions are chosen to be linearly independent, since $\boldsymbol{C}_i \hat{\boldsymbol{K}}$ is a $\hat{\boldsymbol{K}}$-orthogonal projection onto $\operatorname{span}\{\boldsymbol{S}_i\}$ (see Lemma S1). From a modeling perspective, if our computational budget is large enough, we eventually recover the mathematical posterior. This is reflected by the contraction of the posterior over the representer weights (see Proposition S4). The bound in Proposition 2 is tight if no further assumption on the actions is made. This is because we can choose an adverserial sequence of actions such that the first $(n-1)$ are in $\operatorname{span}\{\boldsymbol{v}_*\}^{\perp_{\hat{\boldsymbol{K}}}}$. Then the inverse is perfectly identified in that subspace, but $\boldsymbol{v}_i = \boldsymbol{C}_i \boldsymbol{y} = \boldsymbol{C}_i \hat{\boldsymbol{K}} \boldsymbol{v}_* = \boldsymbol{0}$. In practice, one can derive tighter convergence bounds for specific sequences of actions. For example, for randomized actions the bound depends on their distribution [53, 54]. If residuals $\boldsymbol{r}_i$ are chosen as actions, we obtain the convergence rates

$$\rho(i) = 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \text{ or } \rho(i) = \left(\frac{\lambda_{n-i} - \lambda_1}{\lambda_{n-i} + \lambda_1} \right) \tag{10}$$

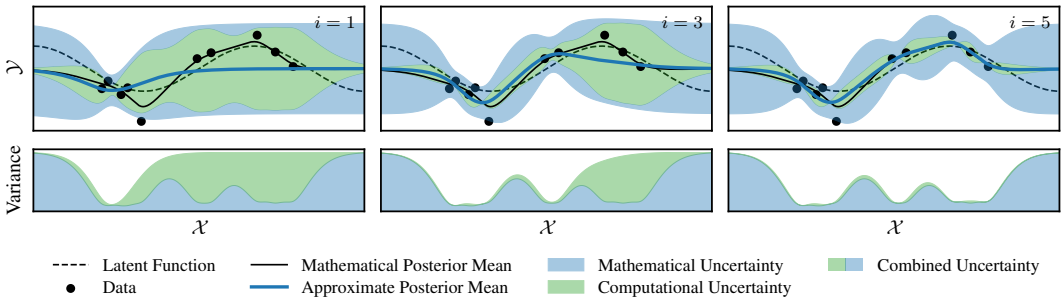


Figure 3: *Computational and combined uncertainty of IterGP as worst-case bounds.*[1]

since then Algorithm 1's estimate of the representer weights equals that of CG (Corollary S2). Here κ is the condition number and λ_j the eigenvalues of either (i) the kernel matrix $\hat{\boldsymbol{K}}$ if $\boldsymbol{s}_i = \boldsymbol{r}_i$, or (ii) the preconditioned kernel matrix $\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\frac{\intercal}{2}}$ if $\boldsymbol{s}_i = \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i$. Now that we have established a convergence result for the representer weights estimate, we can lift this bound to the RKHS to obtain convergence in norm of the posterior mean of IterGP at the same rate.

Theorem 1 (Convergence of the Posterior Mean in RKHS norm)
Let $\mathcal{H}_k$ be the RKHS associated with kernel $k(\cdot,\cdot)$, $\sigma^2 > 0$ and let $\mu_* - \mu_0 \in \mathcal{H}_k$ be the unique solution to the regularized empirical risk minimization problem

$$\arg\min_{f \in \mathcal{H}_k} \tfrac{1}{n}\big(\textstyle\sum_{j=1}^{n}(f(\boldsymbol{x}_j) - (y_j - \mu_j))^2 + \sigma^2\|f\|^2_{\mathcal{H}_k}\big) \tag{11}$$

Then for $i \in \{0, \dots, n\}$ the posterior mean $\mu_i(\cdot)$ computed by Algorithm 1 satisfies

$$\boxed{\|\mu_* - \mu_i\|_{\mathcal{H}_k} \leq \rho(i)c(\sigma^2)\|\mu_* - \mu_0\|_{\mathcal{H}_k}} \tag{12}$$

where the constant $c(\sigma^2) = \sqrt{1 + \frac{\sigma^2}{\lambda_{\min}(\boldsymbol{K})}} \to 1$ as $\sigma^2 \to 0$.

Proof. See Section S2.3. □

While Theorem 1 proves convergence in norm for the posterior mean of IterGP (as μ_* is equivalent to the Equation (11) solution; see [e.g. 57, Sec. 6.2]), the convergence rate may contain expressions which cannot be evaluated at runtime with the limited computation at our disposal (cf. eq. (10)). However, the combined uncertainty of IterGP is a tight bound on the *pointwise* relative error to all possible latent functions which would have resulted in the same computations.

Theorem 2 (Combined Uncertainty as Worst Case Error)
Let $\sigma^2 \geq 0$ and let $k_i(\cdot,\cdot) = k_*(\cdot,\cdot) + k_i^{\text{comp}}(\cdot,\cdot)$ be the combined uncertainty computed by Algorithm 1. Then, for any $\boldsymbol{x} \in \mathcal{X}$ (assuming $\boldsymbol{x} \notin \boldsymbol{X}$ if $\sigma^2 > 0$) we have

$$\sqrt{k_i(\boldsymbol{x},\boldsymbol{x}) + \sigma^2} = \sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \Big(\overbrace{\underbrace{g(\boldsymbol{x}) - \mu_*^g(\boldsymbol{x})}_{\text{error of math. post. mean}} + \underbrace{\mu_*^g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x})}_{\text{computational error}}}^{\text{error of approx. post. mean}}\Big), \quad \textit{and} \tag{13}$$

$$\sqrt{k_i^{\text{comp}}(\boldsymbol{x},\boldsymbol{x})} = \sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \underbrace{(\mu_*^g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x}))}_{\text{computational error}} \tag{14}$$

where $\mu_*^g(\cdot) = k(\cdot,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}g(\boldsymbol{X})$ is the mathematical and $\mu_i^g(\cdot) = k(\cdot,\boldsymbol{X})\boldsymbol{C}_i g(\boldsymbol{X})$ IterGP's posterior mean for the latent function $g \in \mathcal{H}_k$. If $\sigma^2 = 0$, then the above also holds for $\boldsymbol{x} = \boldsymbol{x}_j$.

Proof. See Section S2.4. □

Theorem 2 rigorously explains why the combined (mathematical + computational) uncertainty k_i is the correct object characterizing our belief about the latent function h, given that we are in the limited

[1] The combined (co-)variance decomposes into mathematical and computational covariances, as opposed to the combined standard deviation since $\sqrt{\bigcirc + \bigcirc} \neq \sqrt{\bigcirc} + \sqrt{\bigcirc}$. The bottom panel thus illustrates the variance decomposition. However to illustrate Theorem 2, in the upper panel we plot the combined standard deviation $\sqrt{\bigcirc + \bigcirc}$ and the computational standard deviation $\sqrt{\bigcirc}$ within it.

compute regime. In the same way that the mathematical uncertainty is a tight bound on the distance to all functions g which could have produced the data (see [30, Prop. 3.8]), the combined uncertainty is a tight bound on all functions g which would have produced the same computations. In particular as Figure 3 illustrates and Corollary 1 formalizes, the computational uncertainty (🟢) is a pointwise bound on the relative distance to the mathematical posterior mean and *the combined uncertainty (🟢 + 🔵) is a pointwise bound on the relative distance to the true latent function*.

Corollary 1 (Pointwise Convergence of the Posterior Mean)
Assume the conditions of Theorem 2 hold and assume the latent function $h \in \mathcal{H}_{k^\sigma}$, then it holds that

$$|h(\boldsymbol{x}) - \mu_i(\boldsymbol{x})| \leq \|h\|_{\mathcal{H}_{k^\sigma}} \sqrt{k_i(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2}, \quad \textit{and} \tag{15}$$

$$|\mu_*(\boldsymbol{x}) - \mu_i(\boldsymbol{x})| \leq \|h\|_{\mathcal{H}_{k^\sigma}} \sqrt{k_i^{\text{comp}}(\boldsymbol{x}, \boldsymbol{x})}. \tag{16}$$

Proof. This follows immediately from Theorem 2 by recognizing that $h/\|h\|_{\mathcal{H}_{k^\sigma}}$ has unit norm. □

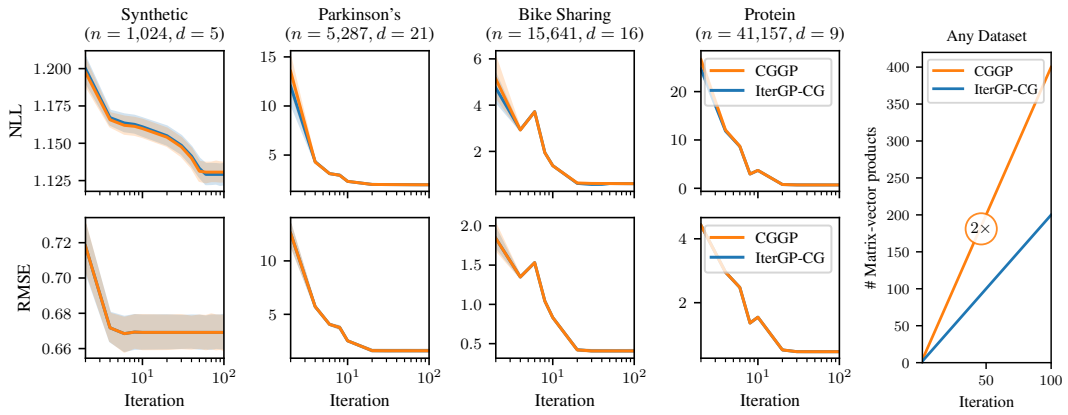
4 Experiments

To demonstrate the effects of quantifying computational uncertainty we perform GP regression on synthetic and benchmark datasets for the two most common GP approximations in the large-scale setting, SVGP [21] and CGGP [25], and their direct analogs from our class of methods. An implementation of Algorithm 1 based on KeOps [42] and ProbNum [58] is available at `<anonymized>`.

Experimental Setup We consider a synthetic dataset of iid uniformly sampled inputs $\boldsymbol{x}_j \in [-1, 1]^d$ with $y(\boldsymbol{x}) = \sin(\pi \boldsymbol{x}^\intercal \mathbf{1}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, as well as a range of UCI datasets [59] with training set sizes $n = 5,287$ to $57,247$, dimensions $d = 9$ to 26 and standardized features. All experiments were run on an NVIDIA GeForce RTX 2080 Ti graphics card. We perform GP regression using a zero mean prior and a Matérn($\frac{1}{2}$) kernel (for other kernels see Section S3). All experiments were run 10 times with randomly sampled training and test splits of $90/10$ and we report average metrics with 95% confidence intervals.

IterGP reduces the necessary computations for CG-based GP inference. We compare IterGP to the CG-based GP inference used in the GPyTorch library [25]. For all datasets, we select hyperparameters using the training procedure of Wenger et al. [28]. As we show in Theorem S5, the posterior mean of IterGP with (conjugate) residual actions is exactly equivalent to performing CG to compute the representer weights. Therefore, both methods produce the exact same posterior mean estimate and thus achieve the same RMSE as a function of CG iterations (Figure 4, bottom). The primary difference between the two methods is in the posterior variance. The combined variance estimate of IterGP is essentially "free" in the sense that it reuses terms from the posterior mean calculation. In contrast, computing the posterior variance with CG requires $n_\star$ additional linear solves $(\hat{\boldsymbol{K}}^{-1}\boldsymbol{x}_{\star 1}, \ldots, \hat{\boldsymbol{K}}^{-1}\boldsymbol{x}_{\star n_\star})$. GPyTorch relies on the Lanczos Variance Estimate technique [60] which essentially warm-starts each of these solves by reusing quantities from the linear solve $\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \boldsymbol{x}_{\star 1})$. While this approach produces reliable variance estimates that converge to the true posterior variance, it requires additional computation: at least one set of additional CG iterations to compute $\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \boldsymbol{x}_{\star 1})$. In Figure 4(a) (top), we see that IterGP and GPyTorch's CGGP achieve nearly identical NLL, suggesting that both methods produce variances that yield similar generalization. The key difference between the methods is that 1) unlike CGGP, IterGP's variances exactly capture both mathematical and computational uncertainty, and 2) IterGP's variances require no additional solves, resulting in half as much computation as GPyTorch's CGGP implementation (see Figure 4(b)).

Quantifying computational uncertainty improves generalization of inducing point methods. To understand the benefits of quantifying computational uncertainty, we compare the linear-time SVGP method (which does not quantify computational uncertainty) with the closest (quadratic-time) inducing point analog from our proposed IterGP framework (see Section 2.1). While the IterGP method is inherently more expensive than SVGP, our goal is simply to demonstrate that inducing points can yield far more accuracy if one has the budget to account for computational uncertainty. To that end, we compare SVGP against IterGP using the same set of randomly-placed inducing points. We identify a set of kernel hyperparameters by optimizing the ELBO of SVGP on the training data, using these for both SVGP and IterGP. As Figure 5 shows, we find that across all datasets that IterGP offers better RMSE and NLL than SVGP, despite the fact that the hyperparameters are chosen to



(a) Generalization on synthetic and UCI benchmark datasets. (b) Comp. Cost

Figure 4: *Generalization of CGGP and its closest IterGP analog.* (a) GP regression using a Matérn($\frac{1}{2}$) kernel on UCI datasets. The plot shows the average generalization error in terms of NLL and RMSE for an increasing number of solver iterations. The posterior mean of IterGP-CG and CGGP is identical, which explains the identical RMSE. However, CGGP performs additional computation for the posterior covariance as (b) illustrates, which is not needed as the identical NLL of IterGP-CG.

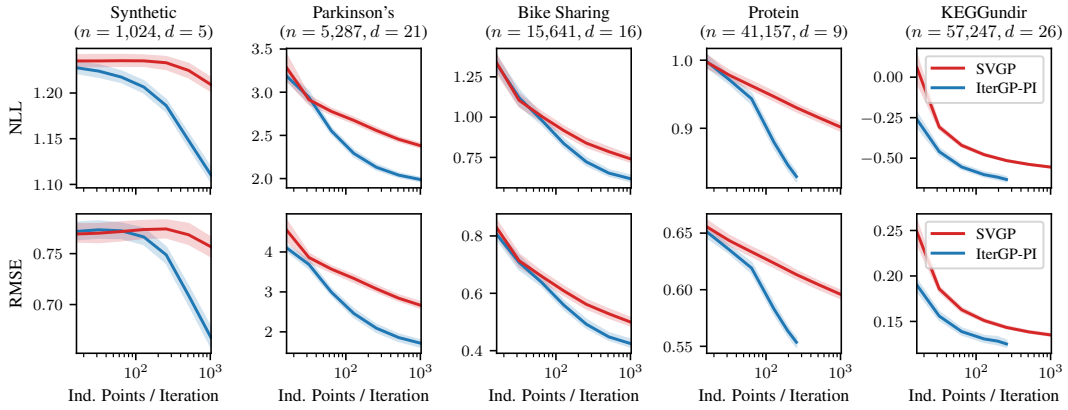


Figure 5: *Generalization of SVGP and its closest IterGP analog.* GP regression using a Matérn($\frac{1}{2}$) kernel on UCI datasets. The plot shows the average generalization error in terms of NLL and RMSE for an increasing number of identical inducing points. After a small number of inducing points relative to the size of the training data, IterGP has significantly lower generalization error than SVGP.

favor SVGP. This suggests that the extra computation needed to quantify computational uncertainty can more "effectively" utilize a set of inducing points for predictive models.

5 Conclusion

Scalable GP approximations inevitably introduce error, leading to a worse model for the latent function in question. This work demonstrates that it is possible to account for *both* uncertainty arising from limited data *and* uncertainty arising from limited computation *exactly* – which as we show improves model performance. IterGP methods return this combined uncertainty which crucially represents a dataset-specific, pointwise worst-case bound on the error to the true latent function. At its core, IterGP performs repeated matrix-vector multiplication resulting in quadratic complexity. Since modern computing architectures (i.e. GPUs) have been specifically designed for this operation at scale, iterative approaches for GP approximation are competitive with theoretically cheaper approximations, like inducing point methods [25, 26]. Finally, in addition to the general utility of IterGP, we expect this class of methods to be particularly useful in applications where accurate uncertainty quantification is important or, due to its inherently online nature, where data is acquired sequentially such as in active learning and Bayesian optimization.

References

[1] Huaiyu Zhu, Christopher KI Williams, Richard Rohwer, and Michal Morciniec. Gaussian regression and optimal finite dimensional linear models. In *Neural Networks and Machine Learning*, 1997.

[2] Giancarlo Ferrari Trecate, Christopher KI Williams, and Manfred Opper. Finite-dimensional approximation of Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 218–224, 1999.

[3] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

[4] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[5] Andrew Gordon Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on massively scalable Gaussian processes. *arXiv pre-print*, 2015. URL http://arxiv.org/abs/1511.01870.

[6] Zichao Yang, Andrew Wilson, Alex Smola, and Le Song. A la carte–learning fast kernels. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1098–1106, 2015.

[7] Pavel Izmailov, Alexander Novikov, and Dmitry Kropotov. Scalable Gaussian processes with billions of inducing inputs via tensor train decomposition. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

[8] Trefor Evans and Prasanth Nair. Scalable Gaussian processes with grid-structured eigenfunctions (GP-GRIEF). In *International Conference on Machine Learning (ICML)*, 2018.

[9] Amir Zandieh, Navid Nouri, Ameya Velingker, Michael Kapralov, and Ilya Razenshteyn. Scaling up kernel ridge regression via locality sensitive hashing. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[10] Aldo V Vecchia. Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):297–312, 1988.

[11] Abhirup Datta, Sudipto Banerjee, Andrew O Finley, and Alan E Gelfand. Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111(514):800–812, 2016.

[12] Matthias Katzfuss and Joseph Guinness. A General Framework for Vecchia Approximations of Gaussian Processes. *Statistical Science*, 36(1):124 – 141, 2021.

[13] Florian Schäfer, Matthias Katzfuss, and Houman Owhadi. Sparse Cholesky factorization by Kullback-Leibler minimization. *SIAM Journal on Scientific Computing*, 43(3):A2019–A2046, 2021.

[14] Alex J Smola and Peter Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 598–604, 2000.

[15] Christopher Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2001.

[16] Petros Drineas and Michael W Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

[17] Matthias W. Seeger, C. K. Williams, and N. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

[18] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems (NeurIPS)*, 18:1257–1264, 2005.

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

[20] Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 567–574. PMLR, 2009.

[21] James Hensman, Nicolò Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 282–290, 2013.

[22] Mark Gibbs. Bayesian Gaussian processes for classification and regression. *University of Cambridge, Cambridge*, 1997.

[23] Iain Murray. Gaussian processes and fast matrix-vector multiplies. In *Numerical Mathematics in Machine Learning Workshop (ICML)*, 2009.

[24] Kurt Cutajar, Michael Osborne, John Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *International Conference on Machine Learning (ICML)*, 2016.

[25] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018:7576–7586, 2018.

[26] Ke Alexander Wang, Geoff Pleiss, Jacob R Gardner, Stephen Tyree, Kilian Q Weinberger, and Andrew Gordon Wilson. Exact Gaussian processes on a million data points. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

[27] Artem Artemev, David R Burt, and Mark van der Wilk. Tighter bounds on the log marginal likelihood of Gaussian process regression using conjugate gradients. In *International Conference on Machine Learning (ICML)*, 2021.

[28] Jonathan Wenger, Geoff Pleiss, Philipp Hennig, John P. Cunningham, and Jacob R. Gardner. Preconditioning for scalable Gaussian process hyperparameter optimization. In *International Conference on Machine Learning (ICML)*, 2022.

[29] Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding probabilistic sparse Gaussian process approximations. *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 2016.

[30] Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences. *arXiv preprint*, 2018. URL http://arxiv.org/abs/1807.02582.

[31] Florian Schäfer, TJ Sullivan, and Houman Owhadi. Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity. *Multiscale Modeling and Simulation*, 19 (2):688–730, 2021.

[32] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49, 1952.

[33] John P Cunningham, Krishna V Shenoy, and Maneesh Sahani. Fast Gaussian process methods for point process intensity estimation. In *International Conference on Machine Learning (ICML)*, 2008.

[34] Andres Potapczynski, Luhuan Wu, Dan Biderman, Geoff Pleiss, and John P Cunningham. Bias-free scalable gaussian processes via randomized truncations. In *International Conference on Machine Learning (ICML)*, 2021.

[35] Bernhard W Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society: Series B (Methodological)*, 47(1):1–21, 1985.

[36] Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural computation*, 14(3):641–668, 2002.

[37] Veit Wild, Motonobu Kanagawa, and Dino Sejdinovic. Connections and equivalences between the Nyström method and sparse variational Gaussian processes. *arXiv pre-print*, 2021. URL http://arxiv.org/abs/2106.01121.

[38] David R. Burt, Carl Edward Rasmussen, and Mark Van Der Wilk. Rates of convergence for sparse variational Gaussian process regression. In *International Conference on Machine Learning (ICML)*, 2019.

[39] Andre G Journel and Charles J Huijbregts. *Mining geostatistics*. Academic Press London, 1976.

[40] James T Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Deisenroth. Efficiently sampling functions from Gaussian process posteriors. In *International Conference on Machine Learning (ICML)*, pages 10292–10302. PMLR, 2020.

[41] James T Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Peter Deisenroth. Pathwise conditioning of Gaussian processes. *arXiv pre-print*, 2020. URL http://arxiv.org/abs/2011.04026.

[42] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. Kernel operations on the GPU, with autodiff, without memory overflows. *Journal of Machine Learning Research (JMLR)*, 22(74):1–6, 2021. URL http://jmlr.org/papers/v22/20-275.html.

[43] Philipp Hennig, Mike A. Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 471(2179), 2015.

[44] Jon Cockayne, Chris Oates, TJ Sullivan, and Mark Girolami. Bayesian probabilistic numerical methods. *SIAM Review*, 61(4):756–789, 2019.

[45] Chris Oates and TJ Sullivan. A modern retrospective on probabilistic numerics. *Statistics and Computing*, 10 2019.

[46] Philipp Hennig. Probabilistic interpretation of linear solvers. *SIAM Journal on Optimization*, 25(1): 234–260, 2015.

[47] Simon Bartels, Jon Cockayne, Ilse CF Ipsen, and Philipp Hennig. Probabilistic linear solvers: A unifying view. *Statistics and Computing*, 29(6):1249–1263, 2019.

[48] Jon Cockayne, Chris Oates, Ilse C. Ipsen, and Mark Girolami. A Bayesian conjugate gradient method. *Bayesian Analysis*, 14(3):937–1012, 2019.

[49] Jon Cockayne, Ilse C. F. Ipsen, Chris J. Oates, and Tim W. Reid. Probabilistic iterative methods for linear systems. *arXiv pre-print*, 2020. URL http://arxiv.org/abs/2012.12615.

[50] Jonathan Wenger and Philipp Hennig. Probabilistic linear solvers for machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[51] Tim W. Reid, Ilse C. F. Ipsen, Jon Cockayne, and Chris J. Oates. BayesCG as an uncertainty aware version of CG, 2022. URL http://arxiv.org/abs/2008.03225.

[52] Stefan Kaczmarz. Angenäherte Auflösung von Systemen linearer Gleichungen. *Bulletin International de l'Académie Polonaise des Sciences et des Lettres. Classe des Sciences Mathématiques et Naturelles. Série A, Sciences Mathématiques*, pages 355–357, 1937.

[53] Thomas Strohmer and Roman Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278, 2009.

[54] Robert M Gower and Peter Richtárik. Randomized iterative methods for linear systems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1660–1690, 2015.

[55] Robert M Gower. Sketch and project: Randomized iterative methods for linear systems and inverting matrices. *arXiv preprint*, 2016. URL http://arxiv.org/abs/1612.06013.

[56] Simon Bartels and Philipp Hennig. Conjugate gradients for kernel machines. *Journal of Machine Learning Research*, 21(55):1–42, 2020.

[57] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[58] Jonathan Wenger, Nicholas Krämer, Nathanael Bosch, Marvin Pförtner, Jonathan Schmidt, Nina Effenberger, Maren Mahsereci, Johannes Zenn, Thomas Gläßle, Alexandra Gessner, Toni Karvonen, Philipp Hennig, and ProbNum Contributors. ProbNum: Probabilistic Numerics in Python, 2021. URL https://github.com/probabilistic-numerics/probnum.

[59] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[60] Geoff Pleiss, Jacob Gardner, Kilian Weinberger, and Andrew Gordon Wilson. Constant-time predictive distributions for Gaussian processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 4114–4123, 2018.

[61] Paz Fink Shustin and Haim Avron. Gauss-Legendre features for Gaussian process regression. *arXiv preprint*, 2021. URL http://arxiv.org/abs/2101.01137.

[62] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. JHU Press, fourth edition, 2013.

[63] Yousef Saad, Manshung Yeung, Jocelyne Erhel, and Frédéric Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing*, 21(5):1909–1926, 2000.

[64] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[65] Jason Frank and Cornelis Vuik. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23(2):442–462, 2001.

[66] Jean-Paul Chiles and Pierre Delfiner. *Geostatistics: modeling spatial uncertainty*, volume 497. John Wiley & Sons, 2009.

[67] Arnaud Doucet. A note on efficient conditional simulation of Gaussian distributions. Technical report, Departments of Computer Science and Statistics, University of British Columbia, 2010.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Proofs are included in the supplementary material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources 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...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]