When are Iterative Gaussian Processes Reliably Accurate?

Wesley J. Maddox [1] **Sanyam Kapoor** [1] **Andrew Gordon Wilson** [1]

Abstract

While recent work on conjugate gradient methods and Lanczos decompositions have achieved scalable Gaussian process inference with highly accurate point predictions, in several implementations these iterative methods appear to struggle with numerical instabilities in learning kernel hyperparameters, and poor test likelihoods. By investigating CG tolerance, preconditioner rank, and Lanczos decomposition rank, we provide a particularly simple prescription to correct these issues: we recommend that one should use a small CG tolerance ($\epsilon \leq 0.01$) and a large root decomposition size ($r \geq 5000$). Moreover, we show that L-BFGS-B is a compelling optimizer for Iterative GPs, achieving convergence with fewer gradient updates.

1. Introduction

There are now many methods for scaling Gaussian processes (GPs), including finite basis expansions (Rahimi et al., 2007; Solin & Särkkä, 2020), inducing point methods (Snelson & Ghahramani, 2006; Titsias, 2009; Hensman et al., 2013), and iterative numerical methods (Gibbs & MacKay, 1996; Cutajar et al., 2016; Gardner et al., 2018). These methods all work in various ways to scale linear solves and log determinant computations involving covariance matrices. In some type of limit, these methods all generally converge to the "exact" Gaussian process regressor, but at a computational expense.

Iterative methods, such as the preconditioned conjugate gradient (CG) and Lanczos algorithms that form the backbone of the popular software library GPyTorch (Gardner et al., 2018), are rapidly growing in popularity, in part due to their accuracy and effective use of GPU parallelization. These methods are different from other scalable GP approximations in that the accuracy of their solves can be precisely controlled by hyperparameters such as tolerances. Thus these methods are considered "exact" in the same sense as mathematical optimization — accurate to arbitrary precision. In practice, these methods can be more precise than the "exact" Cholesky-based alternatives, due to round-off errors in floating point numbers (Gardner et al., 2018).

However, these iterative methods have demonstrated mysterious empirical behaviour. For example, in Jankowiak et al. (2020) the "exact" iterative GPs generally have impressive test RMSE, but surprisingly poor test likelihood. Similar results can be found for the CG-based SKI method in Sun et al. (2021). Moreover, we find there are sometimes numerical instabilities in training kernel hyperparameters with iterative methods, for instance leading to increasingly large kernel length-scales and increasingly poor test performance, as marginal likelihood optimization proceeds. Recent work has explicitly noted issues with iterative CG-based methods (Artemev et al., 2021; Potapczynski et al., 2021). For example, Potapczynski et al. (2021) show that CG methods can provide biased estimates, and propose random truncations as a remedy.

In this work, we study the empirical convergence of iterative Gaussian process models based on conjugate gradients and Lanczos decompositions. Our empirical study evaluates the hyperparameters of these methods, namely the size of the test time Lanczos decomposition r, the tolerance of conjugate gradients τ, and the rank of the preconditioner used for conjugate gradients solutions, w. The primary source of the mysterious empirical issues for iterative methods are overly relaxed defaults in GPyTorch.

We provide simple prescriptions for adjusting these defaults in order for practitioners to achieve reliably good performance with iterative methods. We also highlight the practical benefits of L-BFGS-B for marginal likelihood optimization, which is not presently a standard method for learning kernel hyperparameters.

2. Gaussian Process Regression

Gaussian processes (GPs) are a non-parametric method that model priors directly over functions, and provide well-calibrated uncertainties (Rasmussen & Williams, 2008). In this work, for simplicity, we focus on the regression set-

[1]New York University. Correspondence to: Wesley J. Maddox <wjm363@nyu.edu>.

Proceedings of the OPTML Workshop at International Conference on Machine Learning (ICML), 2021.

ting with isotropic noise. For a dataset $\mathcal{D}$ of size n, the relation between d-dimensional inputs $\mathbf{X} \in \mathbb{R}^{n\times d}$, and corresponding outputs $\mathbf{y} \in \mathbb{R}^n$, is modeled using a GP prior $f(\mathbf{X}) \sim \mathcal{GP}\left(\mu(\mathbf{X}), k(\mathbf{X},\mathbf{X})\right)$ and a Gaussian observation likelihood $\mathbf{y} \sim \mathcal{N}\left(f(\mathbf{X}), \sigma^2\mathbf{I}\right)$. The prior is fully specified by a mean function $\mu(\cdot)$, and a kernel function $k(\cdot,\cdot)$, both with parameters collectively denoted as θ. We take the mean function to be zero, as typical in literature. The kernel function induces a covariance matrix between two sets $A \in \mathbb{R}^{m\times d}$ and $B \in \mathbb{R}^{m'\times d}$, denoted as $K_{A,B} \in \mathbb{R}^{m\times m'}$. Therefore, the prior covariance matrix is denoted by $K_{\mathbf{X},\mathbf{X}} \in \mathbb{R}^{n\times n}$.

Gaussian process inference aims to find the posterior over the functions f for $n_\star$ novel inputs $\mathbf{X}_\star$, which is also a Gaussian and available in closed-form as,

$$
\begin{aligned}
p(f(\mathbf{X}_\star) \mid \mathbf{X}_\star, \mathcal{D}, \theta) &= \mathcal{N}(\mu(\mathbf{X}_\star), \Sigma(\mathbf{X}_\star)), \\
\mu(\mathbf{X}_\star) &= K_{\mathbf{X}_\star,\mathbf{X}} \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \mathbf{y} \,, \\
\Sigma(\mathbf{X}_\star) &= K_{\mathbf{X}_\star,\mathbf{X}_\star} - K_{\mathbf{X}_\star,\mathbf{X}} \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} K^\top_{\mathbf{X}_\star,\mathbf{X}} \,,
\end{aligned} \tag{1}
$$

where $\widehat{K}_{\mathbf{X},\mathbf{X}} = K_{\mathbf{X},\mathbf{X}} + \sigma^2\mathbf{I}$. The parameters θ are chosen by maximizing the marginal log-likelihood (MLL) L_θ,

$$
\underbrace{\log p(\mathbf{y} \mid \mathbf{X}, \theta)}_{L_\theta} \propto -\frac{1}{2}\mathbf{y}^\top \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \mathbf{y} - \frac{1}{2}\log\left|\widehat{K}_{XX}\right| \,, \tag{2}
$$

and its gradients are given by,

$$
\frac{\partial L_\theta}{\partial \theta} = \mathbf{y}^\top \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \frac{\partial \widehat{K}_{\mathbf{X},\mathbf{X}}}{\partial \theta} \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \mathbf{y} + \operatorname{Tr}\left(\widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \frac{\partial \widehat{K}_{\mathbf{X},\mathbf{X}}}{\partial \theta}\right) . \tag{3}
$$

Traditional methods to compute (1) to (3) use Cholesky factorization to solve the linear systems involving $\widehat{K}^{-1}_{\mathbf{X},\mathbf{X}}$, and the determinant computations involving $\widehat{K}_{\mathbf{X},\mathbf{X}}$ (Rasmussen & Williams, 2008). This incurs an expensive cost of $\mathcal{O}\left(n^3\right)$, making exact GPs feasible only for less than $n = 10,000$ data points and severely limiting scalability.

2.1. Iterative Gaussian Processes

Gardner et al. (2018) propose conjugate gradients (CG) to solve $\widehat{K}^{-1}_{\mathbf{X},\mathbf{X}}\mathbf{y}$, and stochastic trace estimator (Hutchinson, 1989) to compute the derivative of *log*-determinant $\operatorname{Tr}\left(\widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \frac{\partial \widehat{K}_{\mathbf{X},\mathbf{X}}}{\partial \theta}\right)$ in (3) as,

$$
\mathbb{E}_{p(z)}\left[z^\top \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \frac{\partial \widehat{K}_{\mathbf{X},\mathbf{X}}}{\partial \theta} z\right] \approx \frac{1}{B}\sum_{b=1}^{B}(z_b^\top \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \frac{\partial \widehat{K}_{\mathbf{X},\mathbf{X}}}{\partial \theta} z_b)
$$

where we use B probe vectors $\{z_b\}_{b=1}^B$. Notably, we can again use conjugate gradients to solve $\widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} z_b$.

Consequently, for kernel matrices $K_{\mathbf{X},\mathbf{X}}$, using $r \ll n$ iterations of conjugate gradients produces exact inference up to machine precision in $\mathcal{O}\left(rn^2\right)$ time, achieving significant computational gains. The MLL can be computed using $B+1$ CG solves and is easily parallelized. Exact GPs are recovered when $r = n$. A flurry of recent work (Wang et al., 2019; Potapczynski et al., 2021; Artemev et al., 2021; Kapoor et al., 2021) has shown the effectiveness of these methods for highly scalable Gaussian processes.

Further, Pleiss et al. (2018) propose to store the single solve of $\mathbf{m} = \widehat{K}^{-1}_{\mathbf{X},\mathbf{X}}\mathbf{y}$ at test time as the predictive mean cache, while similarly computing a rank k Lanczos decomposition of $\widehat{K}^{-1}_{\mathbf{X},\mathbf{X}} \approx RR^\top$ (the predictive covariance cache) and storing that for all new test points. The predictive mean and variance for a test point $\mathbf{x}_\star$ are then given by,

$$
\begin{aligned}
\mu(\mathbf{x}_\star) &= K_{\mathbf{x}_\star,\mathbf{X}}\mathbf{m} \,, \\
\Sigma(\mathbf{x}_\star) &= K_{\mathbf{x}_\star,\mathbf{x}_\star} - K_{\mathbf{x}_\star,\mathbf{X}} RR^\top K^\top_{\mathbf{x}_\star,\mathbf{X}} \,.
\end{aligned} \tag{4}
$$

Computing the Lanczos decomposition costs $\mathcal{O}(kn^2)$, so that computing the predictive mean and covariance after this pre-computation cost only $\mathcal{O}(knn_\star)$. In general, Lanczos will tend to find the largest and smallest eigenvalues of $\widehat{K}^{-1}_{\mathbf{X},\mathbf{X}}$, converging from the *outside in* so to speak (Demmel, 1997, Ch. 7). Collectively, we call inference involving all the above numerical methods as *Iterative Gaussian Processes* (Potapczynski et al., 2021).

3. Understanding Iterative GP Approximations

Understandably, the *speed* and *quality* of Iterative GPs are crucially reliant on conjugate gradients. (i) First, the number of CG iterations r depend on the condition number of $\widehat{K}_{\mathbf{X},\mathbf{X}}$, which grow somewhat with n. To alleviate this concern, Gardner et al. (2018) use a pivoted Cholesky preconditioner of rank w, noting that low ranks are sufficient. Alternatively, Artemev et al. (2021) propose an inducing point-based pre-conditioner. (ii) Second, in practice, we often use a pre-determined error tolerance ϵ to truncate CG iterations. A higher tolerance implies that the solve may use fewer iterations than r, while a lower tolerance may require more iterations than r. A high tolerance, however, has been noted to be detrimental to MLL (2) maximization (Artemev et al., 2021), and may lead to noisy training curves affecting final performance (Kapoor et al., 2021). (iii) Finally, CG truncations at a value $r \ll n$ introduces approximation bias, which is also detrimental to the maximization of MLL (Potapczynski et al., 2021).

Owing to the considerations above, and their emphasis in prior work (Gardner et al., 2018; Pleiss et al., 2018; 2020), we focus our study on analyzing *three* key hyperparameters — (i) the CG tolerance ϵ, (ii) rank of the pivoted Cholesky pre-conditioner w, and (iii) the rank of the Lanczos decomposition k at test time. In addition, we analyze the interac-

tion of these hyperparameters to both a first-order optimizer Adam (Kingma & Ba, 2014), and a second-order optimizer L-BFGS-B (Zhu et al., 1997).

Historically, (nearly) second order optimizers such as L-BFGS-B have been preferred for optimizing Gaussian process hyper-parameters (Rasmussen & Williams, 2008); however, they tend to perform poorly in the presence of noisy estimates of gradients (Gardner et al., 2018; Balandat et al., 2020). Thus, Gardner et al. (2018); Wang et al. (2019) tend to fit their GP models with first order optimizers such as Adam, as L-BFGS-B on all but small datasets tends to be infeasible. One of our goals in our study is to determine the hyper-parameters under which we can use L-BFGS-B to train hyper-parameters of Gaussian process models while still using iterative methods.

3.1. Lanczos Decomposition Rank and Posterior Likelihood

Proposition 1. *For a single test data point $\mathbf{x}_\star$, increasing the rank k of an approximate eigen-decomposition will decrease the posterior variance σ^2.*

The proof of Proposition 1 is in Appendix A, and proceeds algebraically. Proposition 1 implies that increasing the rank of a Lanczos decomposition at test reduces the posterior variance of the GP, making the GP more confident about its predictions. As we show next, this effect can have counterintuitive effects on the test time negative log likelihood (NLL). As a function of the variance σ^2, the NLL is,

$$\text{NLL}(\sigma^2) := \frac{1}{2}\log\sigma^2 + \frac{1}{2\sigma^2}(\mu - y)^2 \,, \tag{5}$$

where μ is the predictive mean. Differentiating and setting the derivative equal to zero finds that $\sigma^2 = (\mu - y)^2$ is a minimum (the global minimum) agrees with standard results where the maximum likelihood estimator of σ^2 in regression is the sum of squared errors of the predictor.

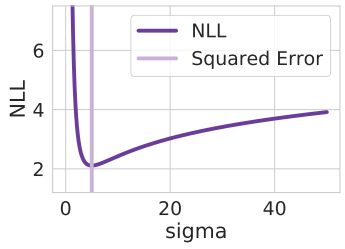

Figure 1. NLL (5) as a function of the posterior standard deviation σ. The NLL is minimized when $\sigma^2 = (y - \mu)^2$.

We show in Figure 1 that the NLL is minimized when σ^2 is set as the squared error of the prediction. Figure 1 also demonstrates that if we control the posterior variance (the square of σ), then the NLL is *not monotonic* as we decrease (or increase) the posterior variance. Thus, as we increase the rank of the Lanczos decomposition, we decrease the variance — beginning by decreasing the NLL towards the NLL's minimizer (which may or may not correspond to the full rank Cholesky solution). However, once we have reached the NLL's minimizer, continuing to decrease the posterior variance then increases the NLL again.

These two settings imply two different behaviors — if the NLL decreases as we increase the root decomposition, then the hyper-parameters of the GP model are under-fit as the squared residual error on the test set is less than the estimated posterior variance. If the opposite occurs (NLL increases as we increase the root decomposition), then the GP model itself is over-fit, the squared residual error on the test set is greater than the estimated posterior variance, e.g. we are over-confident about our predictions.

4. Results

Through our experiments, we establish that (i) increasing the Lanczos decomposition rank r fixes the discrepancy between exact GPs and Iterative GPs; (ii) decreasing the CG tolerance allows us to run L-BFGS-B[1] (Zhu et al., 1997), which has not been investigated in prior literature.

Experimental Setup: We consider four benchmark datasets from the UCI repository (Dua & Graff, 2017), elevators, protein, bike and poletele, standardized to zero sample mean and unit sample variance using the training set. We run the Cholesky and Iterative GPs over five random splits of the data, reporting the mean. As in Gardner et al. (2018), we use the Matérn-$5/2$ kernel with automatic relevance determination (ARD) and constant means. For optimization, we use a learning rate of 0.05 for 2000 steps or until convergence. As established in Section 3, the key hyperparameters of importance are the CG error tolerance ϵ, rank of pivoted Cholesky pre-conditioner w, and the rank of the Lanczos decomposition k at test time. In addition, we want to understand the interaction of these parameters with optimizers Adam and L-BFGS-B. As in Wang et al. (2019), which used L-BFGS-B only for pre-training initialization, we use the default setting of 10 memory vectors.

Effect of Tolerance and Pre-conditioner Size: We find that the the discrepancy between the NLL achieved by Cholesky-based inference and Iterative GPs can be attributed primarily to (i) too large of a CG tolerance, and (ii) too small of a test time root decomposition.

Varying the (test-time) root decomposition size on elevators for a large preconditioner, and use a smaller CG tolerance (0.01 or less), as in Figure 2, we are able to reach the

[1]We use the full-batch implementation available at https://github.com/hjmshi/PyTorch-LBFGS.

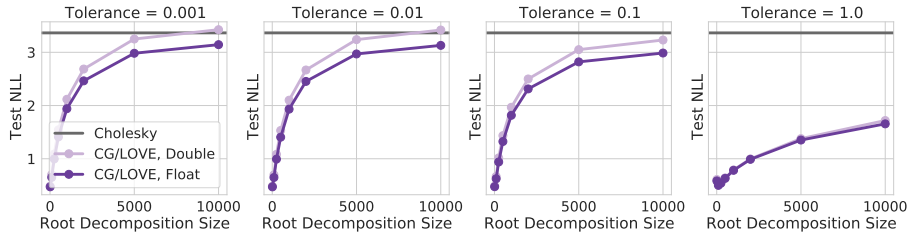

Figure 2. Test set NLL as a function of root decomposition size and CG tolerance on elevators. Decreasing the CG tolerance with a preconditioner of size 100 produces more accurate solves and thus more accurate NLL estimation for large root decomposition sizes.

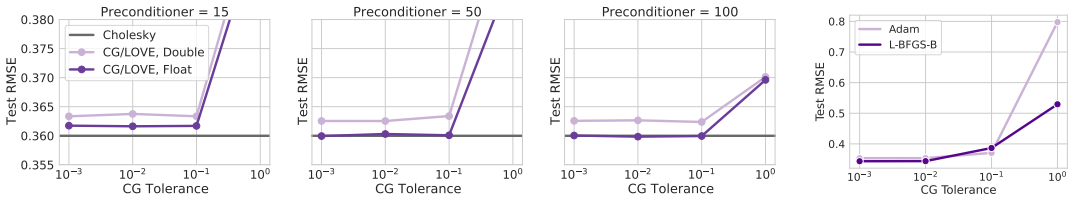

Figure 3. RMSEs as a function of preconditioner size on elevators. A small enough CG tolerance (0.1 or less) produces very similar results to the cholesky baseline even for small preconditioner sizes.

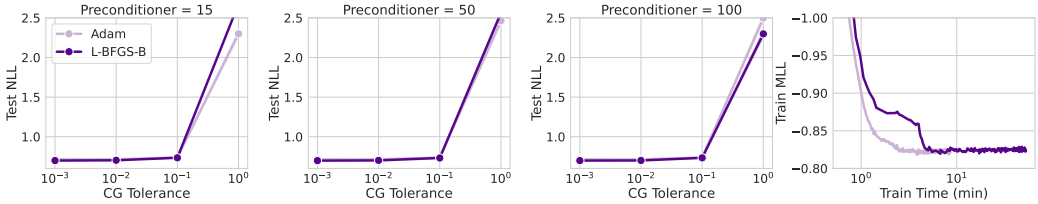

Figure 4. **Left Three Panels:** Test set NLL on protein for both Adam (with L-BFGS-B pre-training) and L-BFGS-B as a function of preconditioner size and CG tolerance. CG tolerance strongly matters for performance, even more so for L-BFGS-B. **Far Right Panel:** Training time on a single GPU; Adam is approximately twice as fast to reach the same training MLL (and ultimately test set NLL).

Cholesky baseline in terms of NLL for a root decomposition size of about $10,000$. By comparison, the RMSE converges much faster to the Cholesky baseline (shown in grey again) for preconditioner sizes of 50 and 100 on the same dataset (Figure 3 left three panels). In addition, we vary the CG tolerance in both double and float precision for three different pre-conditioner sizes, finding that the RMSE converges very quickly, and for CG tolerances less than 0.1, the test time RMSE differences are negligible.

Table 1. Recommended settings for stable training (and testing) with iterative GPs, especially when using L-BFGS-B.

HYPERPARAMETER	VALUE(S)
CG TOLERANCE ϵ	10^{-3}
CG PRECONDITIONER RANK w	50
LANCZOS DECOMPOSITION RANK k	≥ 5000

While the previous experiments compared a mix of first order and second order methods as in Wang et al. (2019), in Figure 4, we show CG tolerance is also important for the success of L-BFGS-B only. In Figure 3 far right, we use a tolerance of 1.0 at train time, varying the test time CG tolerance, finding that the performance of Adam and L-BFGS-B decay for high tolerances, with the Adam performance decaying more. Finally, in Figure 4, we vary preconditioner size and CG tolerance for fixed root decomposition, finding again that low test-time CG tolerance is imperative for good results with L-BFGS-B.

5. Recommendations

As shown in Table 1, we recommend that when evaluating GP negative log likelihoods, one should use a CG tolerance ≤ 0.01 and a large root decomposition size ($\sim \mathcal{O}(5000)$). Otherwise, the NLL estimates can be quite off. As of this writing, default GPyTorch settings[2] are a tolerance of 1.0 and a root decomposition size of 100. We hypothesize that NLL is not the only quantity that can be significantly off as a result of using too small a root decomposition, as sampling depends on similar quantities. In the future, we hope to expand our benchmarking to use Iterative GPs for large-scale data in Bayesian optimization (Balandat et al., 2020).

[2] https://docs.gpytorch.ai/en/stable/settings.html

Acknolwedgements

We thank Sam Stanton for helpful discussions.

References

Artemev, A., Burt, D. R., and van der Wilk, M. Tighter Bounds on the Log Marginal Likelihood of Gaussian Process Regression Using Conjugate Gradients. *arXiv:2102.08314 [cs, stat]*, 2021. arXiv: 2102.08314. 1, 2

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. Botorch: A framework for efficient monte-carlo bayesian optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 3, 4

Cutajar, K., Osborne, M., Cunningham, J., and Filippone, M. Preconditioning kernel matrices. In *International Conference on Machine Learning*, pp. 2529–2538. PMLR, 2016. 1

Demmel, J. W. *Applied numerical linear algebra*. SIAM, 1997. 2

Dua, D. and Graff, C. UCI machine learning repository, 2017. 3

Gardner, J. R., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, volume 31, 2018. 1, 2, 3

Gibbs, M. and MacKay, D. Efficient implementation of gaussian processes, 1996. 1

Hensman, J., Fusi, N., and Lawrence, N. D. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013. 1

Hutchinson, M. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 18:1059–1076, 1989. 2

Jankowiak, M., Pleiss, G., and Gardner, J. Parametric gaussian process regressors. In *International Conference on Machine Learning*, pp. 4702–4712, 2020. 1

Kapoor, S., Finzi, M., Wang, K. A., and Wilson, A. G. Skiing on simplices: Kernel interpolation on the permutohedral lattice for scalable gaussian processes. *arXiv preprint arXiv:2106.06695*, 2021. 2

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

Pleiss, G., Gardner, J. R., Weinberger, K. Q., and Wilson, A. G. Constant-time predictive distributions for gaussian processes. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, 2018. 2, 7

Pleiss, G., Jankowiak, M., Eriksson, D., Damle, A., and Gardner, J. Fast Matrix Square Roots with Applications to Gaussian Processes and Bayesian Optimization. In *Advances in Neural Information Processing Systems*, 2020. 2

Potapczynski, A., Wu, L., Biderman, D., Pleiss, G., and Cunningham, J. P. Bias-Free Scalable Gaussian Processes via Randomized Truncations. *arXiv:2102.06695 [cs, stat]*, 2021. arXiv: 2102.06695. 1, 2

Rahimi, A., Recht, B., et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, pp. 5. Citeseer, 2007. 1

Rasmussen, C. E. and Williams, C. K. I. *Gaussian processes for machine learning*. Adaptive computation and machine learning. 3. print edition, 2008. ISBN 978-0-262-18253-9. 1, 2, 3

Snelson, E. and Ghahramani, Z. Sparse gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, 18:1259–1266, 2006. 1

Solin, A. and Särkkä, S. Hilbert space methods for reduced-rank gaussian process regression. *Statistics and Computing*, 30(2):419–446, 2020. 1

Sun, S., Shi, J., Wilson, A. G., and Grosse, R. Scalable variational gaussian processes via harmonic kernel decomposition. *arXiv preprint arXiv:2106.05992*, 2021. 1

Titsias, M. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pp. 567–574. PMLR, 2009. 1

Wang, K. A., Pleiss, G., Gardner, J. R., Tyree, S., Weinberger, K. Q., and Wilson, A. G. Exact gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 2, 3, 4

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997. 3

Appendix for When is an Iterative Gaussian Process Numerically Exact?

Wesley J. Maddox [1] **Sanyam Kapoor** [1] **Andrew Gordon Wilson** [1]

A. Technical Appendix

We prove Proposition 1 here.

Proposition 1. *For a single test data point $\mathbf{x}_\star$, increasing the rank k of an approximate eigen-decomposition will decrease the posterior variance σ^2.*

Proof. For a given test data point $\mathbf{x}_\star$, we denote the posterior variance using rank k as $\sigma_k^2(\mathbf{x}_\star)$. We also note a full eigendecomposition $\widehat{K}_{\mathbf{X},\mathbf{X}} = Q\widehat{\Lambda}Q^\top$, where $\widehat{\Lambda} = \Lambda + \sigma^2\mathbf{I}$ such that the diagonal of matrix Λ contains the eigenvalues of the covariance matrix $K_{\mathbf{X},\mathbf{X}}$. Then, we aim to show,

$$\sigma_k^2(\mathbf{x}_\star) - \sigma_{k+1}^2(\mathbf{x}_\star) \geq 0 \,. \tag{6}$$

Denoting by $Q_{:k}$ and $\widehat{\Lambda}_{:k}$ a slice of the first k eigenvectors and eigenvalues respectively, we expand the posterior variances as in (1). Therefore, we need,

$$\begin{aligned}&\left(K_{\mathbf{x}_\star,\mathbf{x}_\star} - K_{\mathbf{x}_\star,\mathbf{X}}Q_{:k}\widehat{\Lambda}_{:k}^{-1}Q_{:k}^\top K_{\mathbf{x}_\star,\mathbf{X}}^\top\right) - \\ &\left(K_{\mathbf{x}_\star,\mathbf{x}_\star} - K_{\mathbf{x}_\star,\mathbf{X}}Q_{:k+1}\widehat{\Lambda}_{:k+1}^{-1}Q_{:k+1}^\top K_{\mathbf{x}_\star,\mathbf{X}}^\top\right) \geq 0 \,.\end{aligned} \tag{7}$$

By appending zeros to the rank-k diagonal matrix $\widehat{\Lambda}_{:k}^{-1}$, and factorizing, we have,

$$K_{\mathbf{x}_\star,\mathbf{X}}Q_{:k+1}\left(\widehat{\Lambda}_{:k+1}^{-1} - \left[\widehat{\Lambda}_{:k}^{-1};\mathbf{0}\right]\right)Q_{:k+1}^\top K_{\mathbf{x}_\star,\mathbf{X}}^\top \geq 0 \tag{8}$$

$$(1/\widehat{\lambda}_{k+1})\left(K_{\mathbf{x}_\star,\mathbf{X}}Q_{:k+1}\right)^\top\left(K_{\mathbf{x}_\star,\mathbf{X}}Q_{:k+1}\right) \geq 0 \,, \tag{9}$$

where $\widehat{\lambda}_k > 0$ denotes the k^{th} eigenvalue, and (9) holds by virtue of being an inner product. Hence, the posterior variance reduces as we increase r at test time. ■

B. Hyperparameters Matrix

We list all the hyperparameters studied in Table 2. For model fitting, we use Matern-$5/2$ kernels with constant means and automatic relevance determination[3] and use an exponential moving average stopping rule to monitor convergence[4]

Table 2. We document all the settings and hyperparameters involved.

HYPERPARAMETER	VALUE(S)
KERNEL FAMILY	Matérn-5/2
MAX. EPOCHS	2000
OPTIMIZER	{Adam, L-BFGS-B }
L-BFGS-B MEMORY VECTORS	10
LEARNING RATE	0.05
MAX. CG ITERATIONS	500
CG TOLERANCE ϵ	$\{10^{-3}, 10^{-2}, 10^{-1}, 1.\}$
CG PRECONDITIONER RANK w	{15, 50, 100}
LANCZOS DECOMPOSITION RANK k	{15, 100, 500, 1000, 2500, 5000, 10000}

C. Further Experimental Results

Default settings of GPyTorch In Figure 7, we display the standardized negative log likelihood on elevators, bike and poletele as a function of the root decomposition size for LOVE for the gpytorch default settings (CG tolerance of 1 and a rank 15 preconditioner). On elevators and bike, there are significant differences between the NLLs, even in double precision and for very large root decompositions. Furthermore, elevators shows evidence of over-fitting as the NLL increases when the root decomposition grows while bike (and poletele) show evidence of some under-fitting as the NLL decreases when the root decomposition is larger.

By comparison, the actual accuracy, solely determined by the accuracy of the CG solve $\widehat{K}_{\mathbf{X},\mathbf{X}}^{-1}\mathbf{y}$ is much closer for the default settings. We show these results in Figure 8, where intriguingly a less accurate solve can sometimes reduce the RMSE, perhaps due to increased regularization.

C.1. Comparing Adam and L-BFGS-B

In Figures 9 to 11, we plot the curves for train MLL (2), the test RMSE performance, and the runtime in minutes,

[3] https://botorch.org/api/models.html#module-botorch.models.gp_regression except that we place a softplus transform on the raw noise.

[4] From https://botorch.org/api/optim.html#botorch.optim.fit.fit_gpytorch_torch.

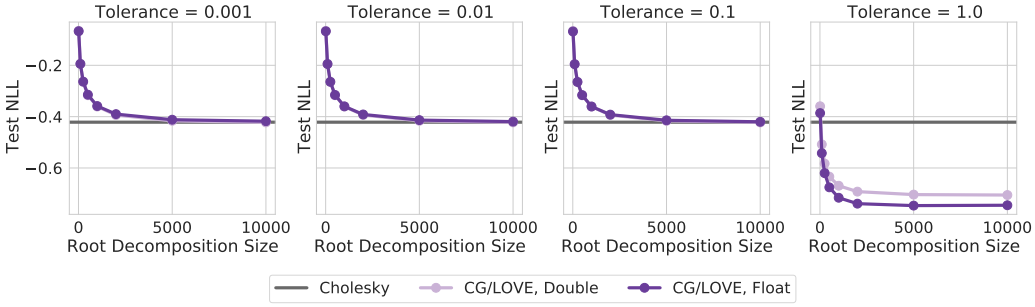

Figure 5. Test NLL as a function of root decomposition size and CG tolerance on `bike` dataset. Decreasing the CG tolerance with a preconditioner of size 15 produces more accurate solves and thus more accurate NLL estimation for large root decomposition sizes.

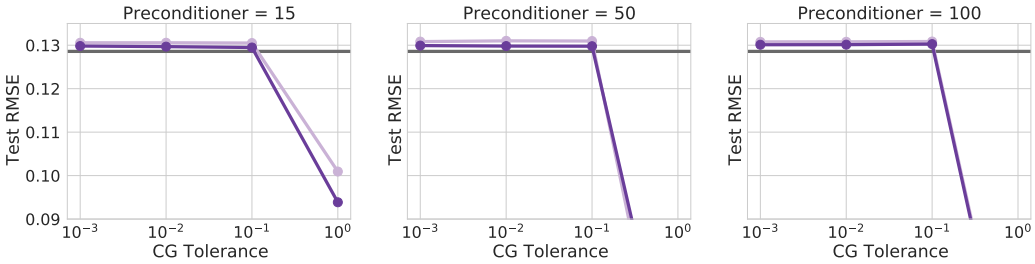

Figure 6. When plotting the test RMSE on `bike` dataset against the CG error tolerance, we find that the CG preconditioner rank does not significantly affect the performance.

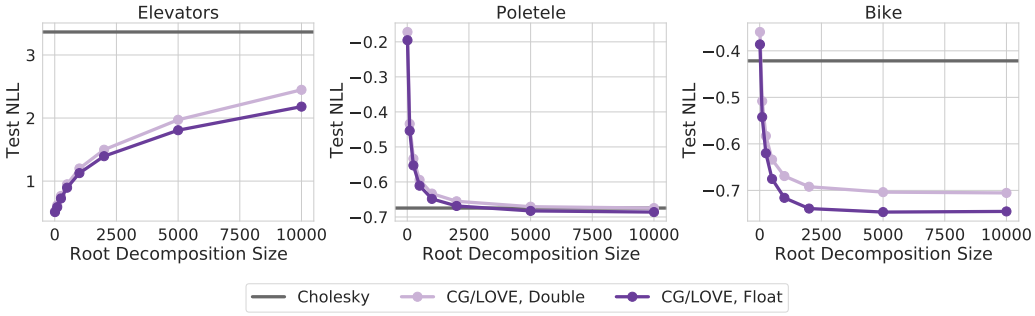

Figure 7. Test NLL (standardized) on `elevators`, `bike`, and `poletele` as a function of the root decomposition size for LOVE (Pleiss et al., 2018) using GPyTorch default settings for preconditioners and CG tolerance. Only on `poletele` does any size root decomposition reach the same NLL as the Cholesky implementation.

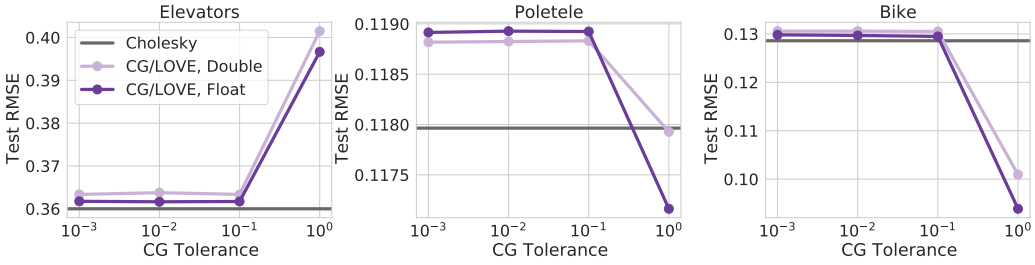

Figure 8. Test set root mean square error (RMSE, standardized) on elevators, bike, and poletele as a function of the CG tolerance for the default rank 15 preconditioner.

against the number of epochs.

Overall, we find that while both optimizers achieve the same train MLL and test RMSE, L-BFGS-B converges much faster than Adam. Unfortunately, the gains achieved here are defeated by the L-BFGS-B being much slower. This is a practical challenge to be resolved in future work.

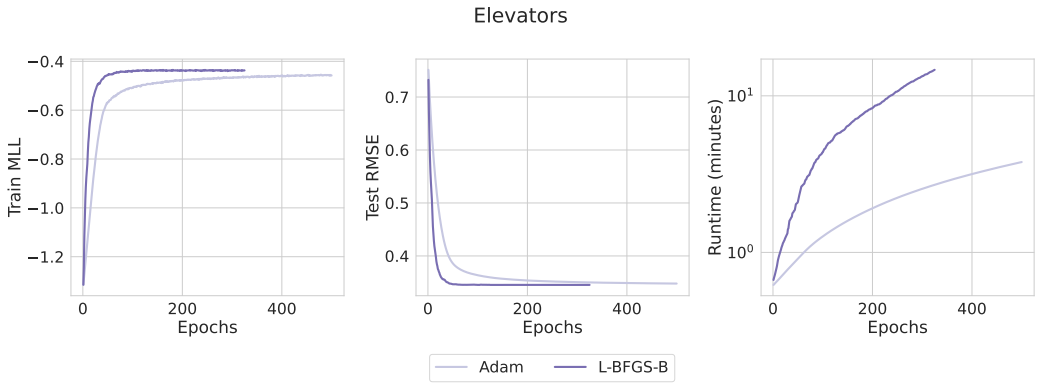

Figure 9. Comparing Adam and L-BFGS-B performance on `elevators` dataset.

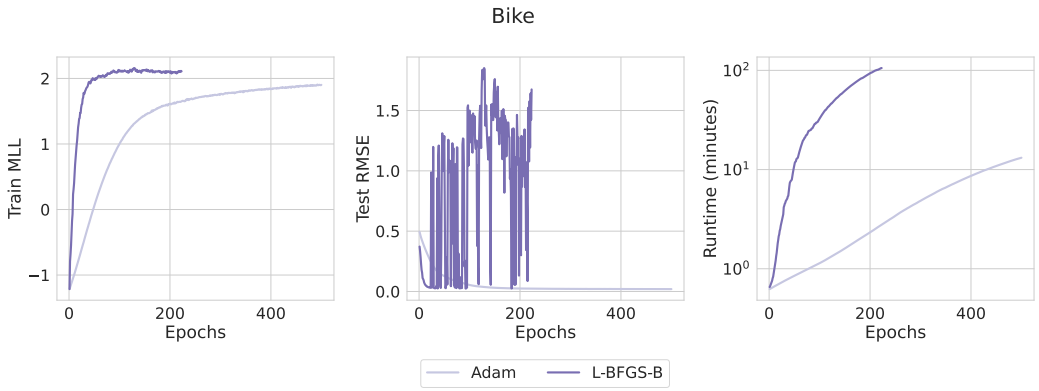

Figure 10. Comparing Adam and L-BFGS-B performance on `bike` dataset.

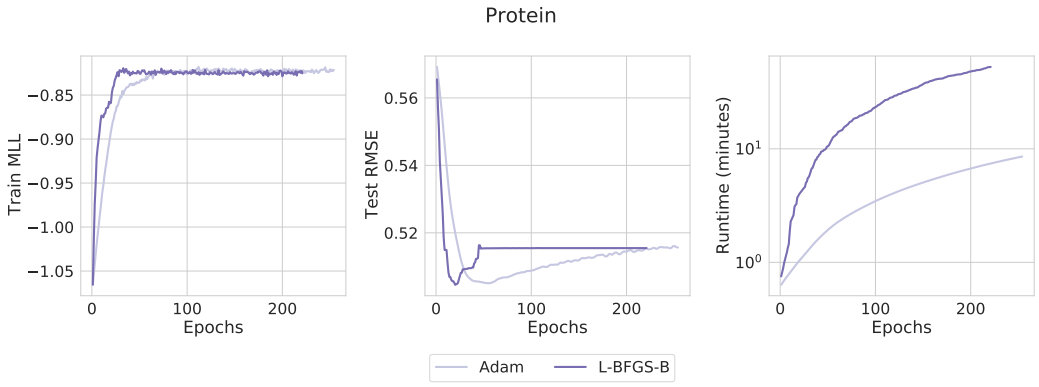

Figure 11. Comparing Adam and L-BFGS-B performance on `protein` dataset.