Supplementary Material for Large Linear Multi-output Gaussian Process Learning

Anonymous Author
Anonymous Institution

1 Implementation Details

LLGP was implemented in Python 3 from Anaconda, which offered an Intel MKL-linked scipy (Jones et al., since 2001). The code made heavy use of other packages, namely climin (Bayer et al., 2016), GPy (GPy, since 2012), and paramz (Zwiessele, 2017). Code and benchmarks are available at `<anonymous repository>`.

Application of our approach to all replication studies was carried out on a large server in a multiprogramming environment: Ubuntu 16.04.3 LTS with 48 Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz. All experiments in the main report were allotted up to 16 threads (both for COGP MATLAB computation threads and parallel Monte Carlo in the case of LLGP). All experiments in the supplement (below, Sec. 2) were allotted only 1 thread each.

2 Representation evaluation

We evaluated the performance of the different kernel representations over various rank and parameterization settings. In particular, we have the following parameters: n total sample size across all outputs, D number of outputs, Q number of kernels k_q, R average added rank, ϵ mean noise, and `ktype` kernel type. Kernel type is one of `mat, periodic, rbf, mix` corresponding to Matérn-3/2, standard periodic[1], and radial basis functions. `mix` refers to a mix of all three kernels.

For each setting, we randomly sample entries for each $A_q, \boldsymbol{\kappa}_q$, and $\boldsymbol{\epsilon}$ and the inverse length scale γ for each kernel. Then, we investigate the average gradient construction accuracy and speed of LLGP for different settings of Q, ϵ, and `ktype`.Each kernel's inverse length scales and periods were selected by sampling uniformly in log space from 1 to 10 with Q samples. Next, we construct a random LMC kernel by sampling entries of each A_q from a standard normal distribution truncated to the unit interval, $\boldsymbol{\kappa}_q$ from an inverse gamma with unit shape and scale, and independent noise $\boldsymbol{\epsilon}$ for every output from an inverse gamma with unit scale and mean ϵ. Inputs and outputs were independently generated from Unif$[0, 1]$ for representation evaluation benchmarking.

[1]We define the periodic kernel as $k(r) = \exp\left(\frac{-\gamma}{2}\sin^2\frac{\pi r}{T}\right)$.

As expected from their asymptotic runtime, SUM, BT, and SLFM representations are complimentary in MVM speed for different configurations of D, R, Q—this results in sparse inversion computation that consistently outperforms Cholesky decomposition in runtime (Tab. 1). For inverting systems, all computations were carried out until the residual ℓ_2 norm was at most 10^{-4}. Altogether, this validates a rule based on asymptotic complexity, where SUM is chosen for $Q = 1$, BT when $QR < D^2$, and SLFM otherwise.

We next evaluated the accuracy of the gradients for $N = 10$ trace samples. Fixing $R = 3, n = 5000, D = 10$, we quantified the accuracy LLGP's $\nabla\mathcal{L}$ by comparing against the exact Cholesky approach. The relative error in the gradients is generally low for smooth kernels that induce diagonally dominant covariance matrices (Fig. 1). Kernels such as the single Matérn or periodic kernel with noise on the order of 10^{-4} lead to less accurate gradients, owing to poor MINRES convergence in the inversions (Fig. 2). We showed in the main paper that the stochastic gradients suffice for optimization in practical examples.

References

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, since 2001. URL `http://www.scipy.org/`. [Online; accessed 2017-02-06].

J. Bayer, C. Osendorfer, S. Diot-Girard, T. Rückstiess, and Sebastian Urban. climin - a pythonic framework for gradient-based function optimization. `http://github.com/BRML/climin`, 2016.

GPy. GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

Table 1: The runtime in seconds for solving $K\mathbf{x} = \mathbf{y}$ for a random kernel K constructed as in Sec. 2 using MINRES for each of the kernel representations. For comparison, the CHOL representation is wallclock time to compute the Cholesky decomposition of the matrix, which must be constructed, and use this decomposition to invert the system. We averaged over five runs. In every run, we use $n = 5000$ simulated data points, `mix` kernels, and $\epsilon = 0.1$.

D	R	Q	CHOLESKY	SUM	BT	SLFM
2	2	10	43.62	3.72	**0.89**	3.71
10	1	10	45.02	11.61	11.79	**2.78**
10	10	1	14.25	**0.24**	1.74	0.42

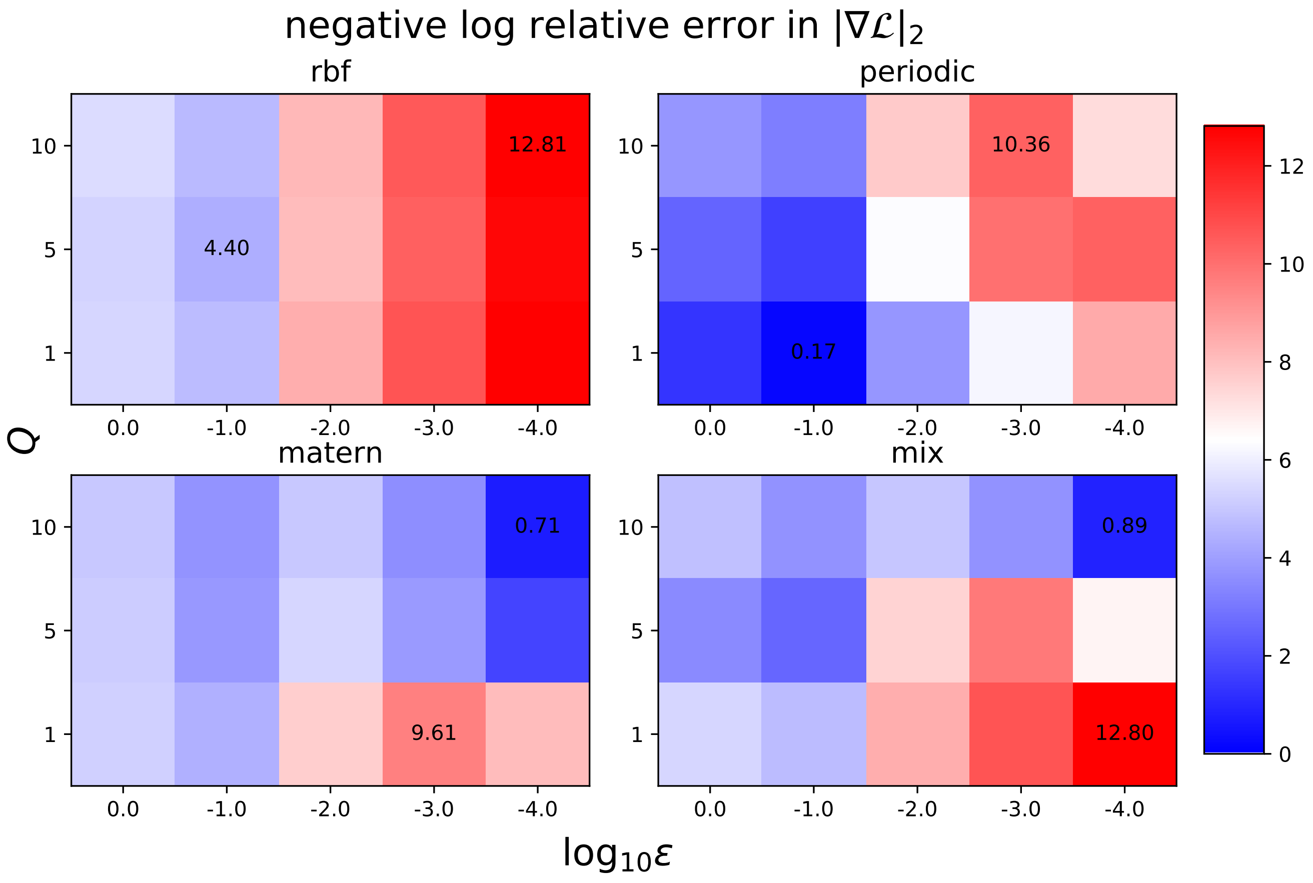

Figure 1: Negative logarithm of the relative error from the LLGP gradient construction to the exact log likelihood gradient. In each case, higher is better, and extremal values are noted. For each data point, the average was taken over five runs. Reducing average noise, corresponding to increases in $-\log_{10}\varepsilon$, and increases in Q, generally make gradient reconstruction more difficult by making K more ill-conditioned, reducing the accuracy of MINRES.

Max Zwiessele. paramz. `https://github.com/sods/paramz`, 2017.

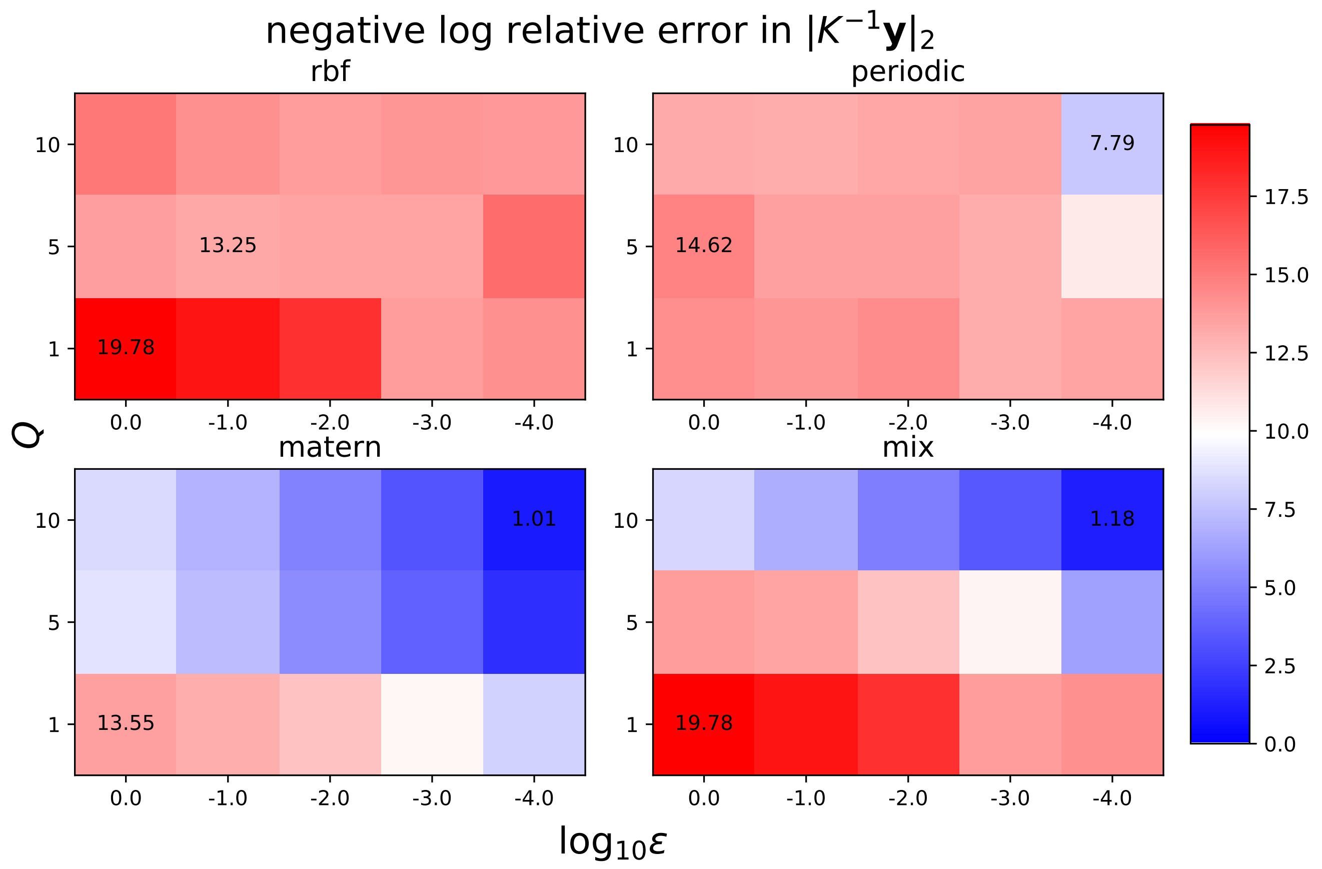

Figure 2: Negative logarithm of the relative error in $K^{-1}\mathbf{y}$, using MINRES compared to the Cholesky solution. Higher is better, and extremal values are noted. An average was taken over five runs.