HAL
archives-ouvertes.fr

Numerical issues in maximum likelihood parameter estimation for Gaussian process interpolation

Subhasish Basak, Sébastien Petit, Julien Bect, Emmanuel Vazquez

To cite this version:

Subhasish Basak, Sébastien Petit, Julien Bect, Emmanuel Vazquez. Numerical issues in maximum likelihood parameter estimation for Gaussian process interpolation. 7th International Conference on machine Learning, Optimization and Data science (LOD 2021), Oct 2021, Grasmere, United Kingdom. hal-03119528v2

HAL Id: hal-03119528

https://hal-centralesupelec.archives-ouvertes.fr/hal-03119528v2

Submitted on 28 Jul 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives| 4.0 International License

Numerical issues in maximum likelihood parameter estimation for Gaussian process interpolation

Subhasish Basak[1], Sébastien Petit[1,2], Julien Bect[1], and Emmanuel Vazquez[1]

[1] Laboratoire des Signaux et Systèmes, CentraleSupélec, CNRS, Univ. Paris-Saclay, Gif-sur-Yvette, France. <firstname>.<lastname>@centralesupelec.fr
[2] Safran Aircraft Engines, Moissy-Cramayel, France

Abstract This article investigates the origin of numerical issues in maximum likelihood parameter estimation for Gaussian process (GP) interpolation and investigates simple but effective strategies for improving commonly used open-source software implementations. This work targets a basic problem but a host of studies, particularly in the literature of Bayesian optimization, rely on off-the-shelf GP implementations. For the conclusions of these studies to be reliable and reproducible, robust GP implementations are critical.

Keywords: Gaussian process · Maximum likelihood estimation · Optimization.

1 Introduction

Gaussian process (GP) regression and interpolation (see, e.g., Rasmussen and Williams, 2006), also known as kriging (see, e.g., Stein, 1999), has gained significant popularity in statistics and machine learning as a non-parametric Bayesian approach for the prediction of unknown functions. The need for function prediction arises not only in supervised learning tasks, but also for building fast surrogates of time-consuming computations, e.g., in the assessment of the performance of a learning algorithm as a function of tuning parameters or, more generally, in the design and analysis computer experiments (Santner et al., 2003). The interest for GPs has also risen considerably due to the development of Bayesian optimization (Mockus, 1975; Jones et al., 1998; Emmerich et al., 2006; Srinivas et al., 2010...).

This context has fostered the development of a fairly large number of open-source packages to facilitate the use of GPs. Some of the popular choices are the Python modules scikit-learn (Pedregosa et al., 2011), GPy (Sheffield machine learning group, 2020), GPflow (Matthews et al., 2017), GPyTorch (Gardner et al., 2018), OpenTURNS (Baudin et al., 2017); the R package DiceKriging (Roustant et al., 2012); and the Matlab/GNU Octave toolboxes GPML (Rasmussen and Nickisch, 2010), STK (Bect et al., 2021) and GPstuff (Vanhatalo et al., 2012).

In practice, all implementations require the user to specify the mean and covariance functions of a Gaussian process prior under a parameterized form. Out of the various methods available to estimate the model parameters, we can safely say that the most popular approach is the *maximum likelihood estimation* (MLE) method. However, a simple numerical experiment consisting in interpolating a function (see Table 1), as

Table 1: Inconsistencies in the results across different Python packages. The results were obtained by fitting a GP model, with constant mean and a Matérn kernel ($\nu = 5/2$), to the Branin function, using the default settings for each package. We used 50 training points and 500 test points sampled from a uniform distribution on $[-5, 10] \times [0, 15]$. The table reports the estimated values for the variance and length scale parameters of the kernel, the empirical root mean squared prediction error (ERMSPE) and the minimized negative log likelihood (NLL). The last row shows the improvement using the recommendations in this study.

LIBRARY	Version	Variance	Lengthscales	ERMSPE	NLL
SCIKIT-LEARN	0.24.2	$9.9 \cdot 10^4$	$(13, 43)$	1.482	132.4
GPY	1.9.9	$8.1 \cdot 10^8$	$(88, 484)$	0.259	113.7
GPYTORCH	1.4.1	$1.1 \cdot 10^1$	$(4, 1)$	12.867	200839.7
GPFLOW	1.5.1	$5.2 \cdot 10^8$	$(80, 433)$	0.274	114.0
OPENTURNS	1.16	$1.3 \cdot 10^4$	$(8, 19)$	3.301	163.1
GPY "IMPROVED"	1.9.9	$9.4 \cdot 10^{10}$	$(220, 1500)$	0.175	112.0

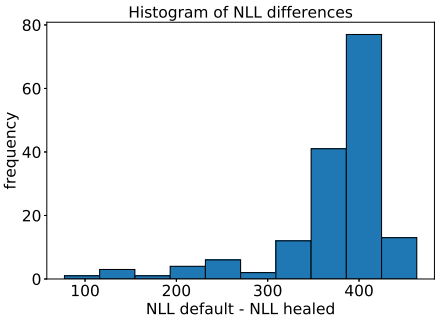
is usually done in Bayesian optimization, shows that different MLE implementations from different Python packages produce very dispersed numerical results when the default settings of each implementation are used. These significant differences were also noticed by Erickson et al. (2018) but the causes and possible mitigation were not investigated. Note that each package uses its own default algorithm for the optimization of the likelihood: GPyTorch uses ADAM (Kingma and Ba, 2015), OpenTURNS uses a truncated Newton method (Nash, 1984) and the others generally use L-BFGS-B (Byrd et al., 1995). It turns out that none of the default results in Table 1 are really satisfactory compared to the result obtained using the recommendations in this study[3].

Focusing on the case of GP interpolation (with Bayesian optimization as the main motivation), the first contribution of this article is to understand the origin of the inconsistencies across available implementations. The second contribution is to investigate simple but effective strategies for improving these implementations, using the well-established GPy package as a case study. We shall propose recommendations concerning several optimization settings: initialization and restart strategies, parameterization of the covariance, etc. By anticipation of our numerical results, the reader is invited to refer to Figure 1 and Table 2, which show that significant improvement in terms of estimated parameter values and prediction errors can be obtained over default settings using better optimization schemes.

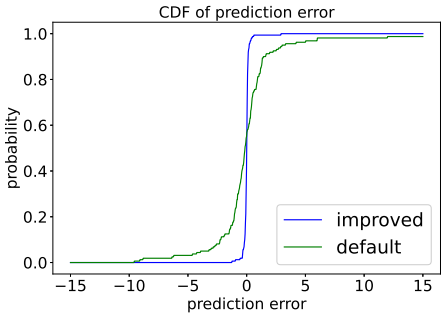
Even though this work targets a seemingly prosaic issue, and advocates somehow simple solutions, we feel that the contribution is nonetheless of significant value considering the widespread use of GP modeling. Indeed, a host of studies, particularly in the literature of Bayesian optimization, rely on off-the-shelf GP implementations: for their conclusions to be reliable and reproducible, robust implementations are critical.

The article is organized as follows. Section 2 provides a brief review of GP modeling and MLE. Section 3 describes some numerical aspects of the evaluation and optimization of the likelihood function, with a focus on GPy's implementation. Section 4 provides

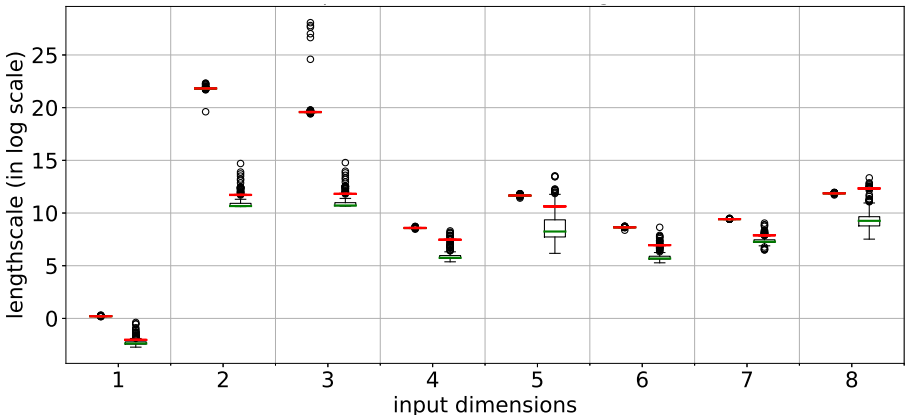
[3] Code available at https://github.com/saferGPMLE



(a) optimized NLL



(b) prediction error



(c) optimized lengthscales

Figure 1: *Improved* (cf. Section 6) vs *default* setups in GPy on the Borehole function with $n = 20d = 160$ random training points. We remove one point at a time to obtain (a) the distribution of the differences of negative log-likelihood (NLL) values between the two setups; (b) the empirical CDFs of the prediction error at the removed points; (c) pairs of box-plots for the estimated range parameters (for each dimension, indexed from 1 to 8 on the x-axis, the box-plot for *improved* setup is on the left and the box-plot for *default* setup is on the right; horizontal red lines correspond to the estimated values using the whole data set without leave-one-out). Notice that the parameter distributions of the *default* setup are more spread out.

Table 2: *Improved* (cf. Section 6) vs *default* setups in GPy for the interpolation of the Borehole function (input space dimension is $d = 8$) with $n \in \{3d, 5d\}$ random data points (see Section 5.3 for details). The experiment is repeated 50 times. The columns report the leave-one-out mean squared error (LOO-MSE) values (empirical mean over the repetitions, together with the standard deviation and the average proportion of the LOO-MSE to the total standard deviation of the data in parentheses).

METHOD	$n = 3d$	$n = 5d$
DEFAULT	17.559 (4.512, 0.387)	10.749 (2.862, 0.229)
IMPROVED	3.949 (1.447, 0.087)	1.577 (0.611, 0.034)

an analysis of factors influencing the accuracy of numerical MLE procedures. Finally, Section 5 assesses the effectiveness of our solutions through numerical experiments and Section 6 concludes the article.

2 Background

2.1 Gaussian processes

Let $Z \sim \mathrm{GP}(m, k)$ be a Gaussian process indexed by $\mathbb{R}^d$, $d \geq 1$, specified by a mean function $m : \mathbb{R}^d \to \mathbb{R}$ and a covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

The objective is to predict $Z(x)$ at a given location $x \in \mathbb{R}^d$, given a data set $D = \{(x_i, z_i) \in \mathbb{R}^d \times \mathbb{R}, 1 \leq i \leq n\}$, where the observations z_is are assumed to be the outcome of an additive-noise model: $Z_i = Z(x_i) + \varepsilon_i$, $1 \leq i \leq n$. In most applications, it is assumed that the ε_is are zero-mean Gaussian i.i.d. random variables with variance $\sigma_\varepsilon^2 \geq 0$, independent of Z. (In rarer cases, heteroscedasticity is assumed.)

Knowing m and k, recall (see, e.g. Rasmussen and Williams, 2006) that the posterior distribution of Z is such that $Z \mid Z_1, \ldots, Z_n, m, k \sim GP(\hat{Z}_n, k_n)$, where $\hat{Z}_n$ and k_n stand respectively for the posterior mean and covariance functions:

$$\begin{aligned} \hat{Z}_n(x) &= m(x) + \textstyle\sum_{i=1}^n w_i(x; \underline{\mathrm{x}}_n)\,(z_i - m(x_i))\,, \\ k_n(x,y) &= k(x,y) - \mathrm{w}(y; \underline{\mathrm{x}}_n)^\mathsf{T} \mathrm{K}(\underline{\mathrm{x}}_n, x)\,, \end{aligned}$$

where $\underline{\mathrm{x}}_n$ denotes observation points $(x_1, \ldots, x_n)$ and the weights $w_i(x; \underline{\mathrm{x}}_n)$ are solutions of the linear system:

$$(\mathrm{K}(\underline{\mathrm{x}}_n, \underline{\mathrm{x}}_n) + \sigma_\varepsilon^2 \mathrm{I}_n)\, \mathrm{w}(x; \underline{\mathrm{x}}_n) = \mathrm{K}(\underline{\mathrm{x}}_n, x)\,, \tag{1}$$

with $\mathrm{K}(\underline{\mathrm{x}}_n, \underline{\mathrm{x}}_n)$ the $n \times n$ covariance matrix with entries $k(x_i, x_j)$, I_n the identity matrix of size n, and $\mathrm{w}(x; \underline{\mathrm{x}}_n)$ (resp. $\mathrm{K}(\underline{\mathrm{x}}_n, x)$) the column vector with entries $w_i(x; \underline{\mathrm{x}}_n)$ (resp. $k(x_i, x)$), $1 \leq i \leq n$. The posterior covariance at $x, y \in \mathbb{R}^d$ may be written as

$$k_n(x,y) = k(x,y) - \mathrm{w}(y; \underline{\mathrm{x}}_n)^\mathsf{T} \mathrm{K}(\underline{\mathrm{x}}_n, x)\,. \tag{2}$$

It is common practice to assume a zero mean function $m = 0$—a reasonable choice if the user has taken care to center data—but most GP implementations also provide an

Table 3: Some kernel functions available in GPy. The Matérn kernel is recommended by Stein (1999). Γ denotes the gamma function, $\mathscr{K}_\nu$ is the modified Bessel function of the second kind.

KERNEL	$r(h)$, $h \in [0,+\infty)$
SQUARED EXPONENTIAL	$\exp(-\frac{1}{2}r^2)$
RATIONAL QUADRATIC	$(1+r^2)^{-\nu}$
MATÉRN WITH PARAM. $\nu > 0$	$\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\sqrt{2\nu}r\right)^{\nu}\mathscr{K}_\nu\left(\sqrt{2\nu}r\right)$

option for setting a constant mean function $m(\,\cdot\,) = \mu \in \mathbb{R}$. In this article, we will include such a constant in our models, and treat it as an additional parameter to be estimated by MLE along with the others. (Alternatively, μ could be endowed with a Gaussian or improper-uniform prior, and then integrated out; see, e.g., O'Hagan (1978).)

The covariance function, aka covariance kernel, models similarity between data points and reflects the user's prior belief about the function to be learned. Most GP implementations provide a couple of stationary covariance functions taken from the literature (e.g., Wendland, 2004; Rasmussen and Williams, 2006). The *squared exponential*, the *rational quadratic* or the *Matérn* covariance functions are popular choices (see Table 3). These covariance functions include a number of parameters: a variance parameter $\sigma^2 > 0$ corresponding to the variance of Z, and a set of range (or length scale) parameters $\rho_1, \ldots, \rho_d$, such that

$$k(x,y) = \sigma^2 r(h)\,, \tag{3}$$

with $h^2 = \sum_{i=1}^d (x_{[i]} - y_{[i]})^2/\rho_i^2$, where $x_{[i]}$ and $y_{[i]}$ denote the elements of x and y. The function $r : \mathbb{R} \to \mathbb{R}$ in (3) is the stationary correlation function of Z. From now on, the vector of model parameters will be denoted by $\theta = (\sigma^2, \rho_1, \ldots, \rho_d, \ldots, \sigma_\varepsilon^2)^\mathsf{T} \in \Theta \subset \mathbb{R}^p$, and the corresponding covariance matrix $\mathrm{K}(\underline{\mathrm{x}}_n, \underline{\mathrm{x}}_n) + \sigma_\varepsilon^2 \mathrm{I}_n$ by K_θ.

2.2 Maximum likelihood estimation

In this article, we focus on GP implementations where the parameters $(\theta, \mu) \in \Theta \times \mathbb{R}$ of the process Z are estimated by maximizing the likelihood $\mathcal{L}(\underline{Z}_n|\theta,\mu)$ of $\underline{Z}_n = (Z_1, \ldots, Z_n)^\mathsf{T}$, or equivalently, by minimizing the negative log-likelihood (NLL)

$$-\log(\mathcal{L}(\underline{Z}_n|\theta,\mu)) = \frac{1}{2}(\underline{Z}_n - \mu\mathbb{1}_n)^\top \mathrm{K}_\theta^{-1}(\underline{Z}_n - \mu\mathbb{1}_n) + \frac{1}{2}\log|\mathrm{K}_\theta| + \text{constant}. \tag{4}$$

This optimization is typically performed by gradient-based methods, although local maxima can be of significant concern as the likelihood is often non-convex. Computing the likelihood and its gradient with respect to (θ, μ) has a $O(n^3 + dn^2)$ computational cost (Rasmussen and Williams, 2006; Petit et al., 2020).

3 Numerical noise

The evaluation of the NLL as well as its gradient is subject to numerical noise, which can prevent proper convergence of the optimization algorithms. Figure 2 shows a typical

situation where the gradient-based optimization algorithm stops before converging to an actual minimum. In this section, we provide an analysis on the numerical noise on the NLL using the concept of local condition numbers. We also show that the popular solution of adding *jitter* cannot be considered as a fully satisfactory answer to the problem of numerical noise.

Numerical noise stems from both terms of the NLL, namely $\frac{1}{2}\underline{Z}_n^\top \mathrm{K}_\theta^{-1}\underline{Z}_n$ and $\frac{1}{2}\log|\mathrm{K}_\theta|$. (For simplification, we assume $\mu = 0$ in this section.)

First, recall that the condition number $\kappa(\mathrm{K}_\theta)$ of K_θ, defined as the ratio $|\lambda_{\max}/\lambda_{\min}|$ of the largest eigenvalue to the smallest eigenvalue (Press et al., 1992), is the key element for analyzing the numerical noise on $\mathrm{K}_\theta^{-1}\underline{Z}_n$. In double-precision floating-point approximations of numbers, $\underline{Z}_n$ is corrupted by an error ε whose magnitude is such that $\|\varepsilon\|/\|\underline{Z}_n\| \simeq 10^{-16}$. Worst-case alignment of $\underline{Z}_n$ and ε with the eigenvectors of K_θ gives

$$\frac{\|\mathrm{K}_\theta^{-1}\varepsilon\|}{\|\mathrm{K}_\theta^{-1}\underline{Z}_n\|} \simeq \kappa(\mathrm{K}_\theta) \times 10^{-16}\,, \tag{5}$$

which shows how the numerical noise is amplified when K_θ becomes ill-conditioned.

The term $\log|\mathrm{K}_\theta|$ is nonlinear in K_θ, but observe, using the identity $\mathrm{d}\log|\mathrm{K}_\theta|/\mathrm{d}\mathrm{K}_\theta = \mathrm{K}_\theta^{-1}$, that the differential of $\log|\cdot|$ at K_θ is given by $H \mapsto \mathrm{Trace}(\mathrm{K}_\theta^{-1}H)$. Thus, the induced operator norm with respect to the Frobenius norm $\|\cdot\|_F$ is $\|\mathrm{K}_\theta^{-1}\|_F$. We can then apply results from Trefethen and Bau (1997) to get a local condition number of the mapping $\mathrm{A} \mapsto \log|\mathrm{A}|$ at K_θ:

$$\kappa(\log|\cdot|, \mathrm{K}_\theta) \triangleq \lim_{\varepsilon\to 0} \sup_{\|\delta_\mathrm{A}\|_F \le \varepsilon} \frac{\big|\log|\mathrm{K}_\theta + \delta_\mathrm{A}| - \log|\mathrm{K}_\theta|\big|}{\big|\log|\mathrm{K}_\theta|\big|} \frac{\|\mathrm{K}_\theta\|_F}{\|\delta_\mathrm{A}\|_F} = \frac{\sqrt{\sum_{i=1}^n \frac{1}{\lambda_i^2}}\sqrt{\sum_{i=1}^n \lambda_i^2}}{|\sum_{i=1}^n \log(\lambda_i)|} \tag{6}$$

where $\lambda_1, \cdots, \lambda_n$ are the (positive) eigenvalues of K_θ. Then, we have

$$\frac{\kappa(\mathrm{K}_\theta)}{|\sum_{i=1}^n \log(\lambda_i)|} \le \kappa(\log|\cdot|, \mathrm{K}_\theta) \le \frac{n\kappa(\mathrm{K}_\theta)}{|\sum_{i=1}^n \log(\lambda_i)|}, \tag{7}$$

which shows that numerical noise on $\log|\mathrm{K}_\theta|$ is linked to the condition number of K_θ.

The local condition number of the quadratic form $\frac{1}{2}\underline{Z}_n^\mathsf{T}\mathrm{K}_\theta^{-1}\underline{Z}_n$ as a function of $\underline{Z}_n$ can also be computed analytically. Some straightforward calculations show that it is bounded by $\kappa(\mathrm{K}_\theta)$.

(When the optimization algorithm stops in the example of Figure 2, we have $\kappa(\mathrm{K}_\theta) \simeq 10^{11}$ and $\kappa(\log|\cdot|, \mathrm{K}_\theta) \simeq 10^{9.5}$. The empirical numerical fluctuations are measured as the residuals of a local second-order polynomial best fit, giving noise levels 10^{-7}, 10^{-8} and $10^{-7.5}$ for $\mathrm{K}_\theta^{-1}\underline{Z}_n$, $\frac{1}{2}\underline{Z}_n^\mathsf{T}\mathrm{K}_\theta^{-1}\underline{Z}_n$ and $\log|\mathrm{K}_\theta|$ respectively. These values are consistent with the above first-order analysis.)

Thus, when $\kappa(\mathrm{K}_\theta)$ becomes large in the course of the optimization procedure, numerical noise on the likelihood and its gradient may trigger an early stopping of the optimization algorithm (supposedly when the algorithm is unable to find a proper direction of improvement). It is well-known that $\kappa(\mathrm{K}_\theta)$ becomes large when $\sigma_\varepsilon^2 = 0$ and one of the following conditions occurs: 1) data points are close, 2) the covariance is very smooth (as for instance when considering the squared exponential covariance), 3) when

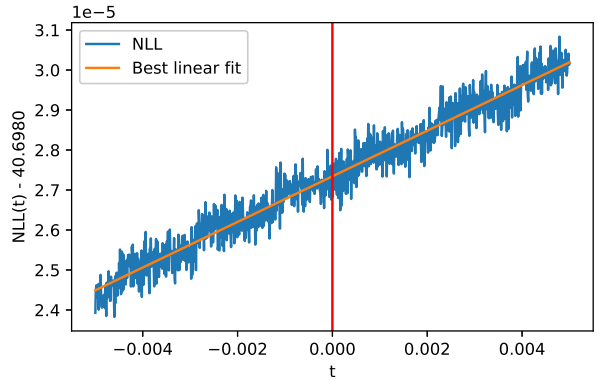


Figure 2: Noisy NLL profile along a particular direction in the parameter space, with a best linear fit (orange line). This example was obtained with GPy while estimating the parameters of a Matérn $5/2$ covariance, using 20 data points sampled from a *Branin function*, and setting $\sigma_\varepsilon^2 = 0$. The red vertical line indicates the location where the optimization of the likelihood stalled.

the range parameters ρ_i are large. These conditions arise more often than not. Therefore, the problem of numerical noise in the evaluation of the likelihood and its gradient is a problem that should not be neglected in GP implementations.

The most classical approach to deal with ill-conditioned covariance matrices is to add a small positive number on the diagonal of the covariance matrix, called *jitter*, which is equivalent to assuming a small observation noise with variance $\sigma_\varepsilon^2 > 0$. In GPy for instance, the strategy consists in always setting a minimal jitter of 10^{-8}, which is automatically increased by an amount ranging from $10^{-6}\sigma^2$ to $10^{-1}\sigma^2$ whenever the Cholesky factorization of the covariance matrix fails (due to numerical non-positiveness). The smallest jitter making K_θ numerically invertible is kept and an error is thrown if no jitter allows for successful factorization. However, note that large values for the jitter may yield smooth, non-interpolating approximations, with possible unintuitive and undesirable effects (see Andrianakis and Challenor, 2012), and causing possible convergence problems in Bayesian optimization.

Table 4 illustrates the behaviour of GP interpolation when σ_ε^2 is increased. It appears that finding a satisfying trade-off between good interpolation properties and low numerical noise level can be difficult. Table 4 also supports the connection in (5) and (7) between noise levels and $\kappa(\mathrm{K}_\theta)$. In view of the results of Figure 1 based on the default settings of GPy and Table 4, we believe that adaptive jitter cannot be considered as a do-it-all solution.

4 Strategies for improving likelihood maximization

In this section we investigate simple but hopefully efficient levers / strategies to improve available implementations of MLE for GP interpolation, beyond the control of the numerical noise on the likelihood using jitter. We mainly focus on 1) initialization methods for the optimization procedure, 2) stopping criteria, 3) the effect of "restart" strategies and 4) the effect of the parameterization of the covariance.

Table 4: Influence of the jitter on the GP model (same setting as in Figure 2). The table reports the condition numbers $\kappa(\mathrm{K}_\theta)$ and $\kappa(\log|\cdot|, \mathrm{K}_\theta)$, and the impact on the relative empirical standard deviations δ_{quad} and δ_{logdet} of the numerical noise on $\underline{Z}_n^T \mathrm{K}_\theta^{-1} \underline{Z}_n$ and $\log|\mathrm{K}_\theta|$ respectively (measured using second-order polynomial regressions). As σ_ε increases, δ_{quad} and δ_{logdet} decrease but the interpolation error $\sqrt{\mathrm{SSR}/\mathrm{SST}} = \sqrt{\frac{1}{n}\sum_{j=1}^n (Z_j - \hat{Z}_n(x_j))^2}/\mathrm{std}(Z_1, ..., Z_n)$ and the NLL increase. Reducing numerical noise while keeping good interpolation properties requires careful attention in practice.

σ_ε^2 / σ^2	0.0	10^{-8}	10^{-6}	10^{-4}	10^{-2}
$\kappa(\mathrm{K}_\theta)$	10^{11}	10^{9}	$10^{7.5}$	$10^{5.5}$	$10^{3.5}$
$\kappa(\log\lvert\cdot\rvert, \mathrm{K}_\theta)$	$10^{9.5}$	$10^{8.5}$	$10^{6.5}$	$10^{4.5}$	$10^{2.5}$
δ_{quad}	10^{-8} $(= 10^{11-19})$	$10^{-9.5}$ $(= 10^{9-18.5})$	$10^{-10.5}$ $(= 10^{7.5-18})$	10^{-12} $(= 10^{5.5-17.5})$	10^{-14} $(= 10^{3.5-17.5})$
δ_{logdet}	$10^{-7.5}$ $(= 10^{9.5-17})$	10^{-9} $(= 10^{8.5-17.5})$	10^{-11} $(= 10^{6.5-17.5})$	$10^{-13.5}$ $(= 10^{4.5-18})$	$10^{-15.5}$ $(= 10^{2.5-18})$
$-\log(\mathcal{L}(\underline{Z}_n\lvert\theta))$	40.69	45.13	62.32	88.81	124.76
$\sqrt{\mathrm{SSR}/\mathrm{SST}}$	$3.3 \cdot 10^{-10}$	$1.2 \cdot 10^{-3}$	0.028	0.29	0.75

4.1 Initialization strategies

Most GP implementations use a gradient-based local optimization algorithm to maximize the likelihood that requires the specification of starting/initial values for the parameters. In the following, we consider different initialization strategies.

Moment-based initialization. A first strategy consists in setting the parameters using empirical moments of the data. More precisely, assuming a constant mean $m = \mu$, and a stationary covariance k with variance σ^2 and range parameters $\rho_1, \ldots, \rho_d$, set

$$\mu_{\text{init}} = \mathrm{mean}\,(Z_1, \ldots, Z_n), \tag{8}$$

$$\sigma^2_{\text{init}} = \mathrm{var}\,(Z_1, \ldots, Z_n), \tag{9}$$

$$\rho_{k,\,\text{init}} = \mathrm{std}\,(x_{1,[k]}, \ldots, x_{n,[k]}), \quad k = 1, \ldots, d, \tag{10}$$

where mean, var and std stand for the empirical mean, variance and standard deviation, and $x_{i,[k]}$ denotes the kth coordinate of $x_i \in \mathbb{R}^d$. The rationale behind (10) (following, e.g., Rasmussen and Williams, 2006) is that the range parameters can be thought of as the distance one has to move in the input space for the function value to change significantly and we assume, a priori, that this distance is linked to the dispersion of data points.

In GPy for instance, the default initialization consists in setting $\mu = 0$, $\sigma^2 = 1$ and $\rho_k = 1$ for all k. This is equivalent to the *moment-based* initialization scheme when the data (both inputs and outputs) are centered and standardized. The practice of standardizing the input domain into a unit length hypercube has been proposed (see, e.g., Snoek et al., 2012) to deal with numerical issues that arise due to large length scale values.

Profiled initialization. Assume the range parameters $\rho_1, \ldots, \rho_d$ (and more generally, all parameters different from σ^2, σ_ε^2 and μ) are fixed, and set $\sigma_\varepsilon^2 = \alpha\sigma^2$, with a prescribed

multiplicative factor $\alpha \geq 0$. In this case, the NLL can be optimized analytically w.r.t. μ and σ^2. Optimal values turn out to be the generalized least squares solutions

$$\mu_{\mathrm{GLS}} = (\mathbb{1}_n^\top \mathrm{K}_{\tilde{\theta}}^{-1} \mathbb{1}_n)^{-1} \mathbb{1}_n^\top \mathrm{K}_{\tilde{\theta}}^{-1} \underline{\mathrm{Z}}_n \,, \tag{11}$$

$$\sigma^2_{\mathrm{GLS}} = \frac{1}{n} (\underline{\mathrm{Z}}_n - \mu_{\mathrm{GLS}}\, \mathbb{1}_n)^\top \mathrm{K}_{\tilde{\theta}}^{-1} (\underline{\mathrm{Z}}_n - \mu_{\mathrm{GLS}}\, \mathbb{1}_n) \,, \tag{12}$$

where $\tilde{\theta} = (\sigma^2, \rho_1, \ldots, \rho_d, \ldots, \sigma_\varepsilon^2)^\top \in \Theta$, with $\sigma^2 = 1$ and $\sigma_\varepsilon^2 = \alpha$. Under the *profiled* initialization scheme, $\rho_1, \ldots, \rho_d$ are set using (10), α is prescribed according to user's preference, and μ and σ^2 are initialized using (11) and (12).

Grid-search initialization. *Grid-search* initialization is a *profiled* initialization with the addition of a grid-search optimization for the range parameters.

Define a nominal range vector ρ_0 such that

$$\rho_{0,[k]} \;=\; \sqrt{d} \left(\max_{1 \leq i \leq n} x_{i,[k]} - \min_{1 \leq i \leq n} x_{i,[k]} \right), \quad 1 \leq k \leq d.$$

Then, define a one-dimensional grid of size L (e.g., $L = 5$) by taking range vectors proportional to ρ_0: $\{\alpha_1 \rho_0, \ldots, \alpha_L \rho_0\}$, where the α_is range, in logarithmic scale, from a "small" value (e.g., $\alpha_1 = 1/50$) to a "large" value (e.g., $\alpha_L = 2$). For each point of the grid, the likelihood is optimized with respect to μ and σ^2 using (11) and (12). The range vector with the best likelihood value is selected. (Note that this initialization procedure is the default initialization procedure in the Matlab/GNU Octave toolbox STK.)

4.2 Stopping condition

Most GP implementations rely on well-tested gradient-based optimization algorithms. For instance, a popular choice in Python implementations is to use the limited-memory BFGS algorithm with box constraints (L-BFGS-B; see Byrd et al., 1995) of the SciPy ecosystem. (Other popular optimization algorithms include the ordinary BFGS, truncated Newton constrained, SQP, etc.; see, e.g., Nocedal and Wright (2006).) The L-BFGS-B algorithm, which belongs to the class of quasi-Newton algorithms, uses limited-memory Hessian approximations and shows good performance on non-smooth functions (Curtis and Que, 2015).

Regardless of which optimization algorithm is chosen, the user usually has the possibility to tune the behavior of the optimizer, and in particular to set the stopping condition. Generally, the stopping condition is met when a maximum number of iterations is reached or when a norm on the steps and/or the gradient become smaller than a threshold.

By increasing the strictness of the stopping condition during the optimization of the likelihood, one would expect better parameter estimations, provided the numerical noise on the likelihood does not interfere too much.

4.3 Restart and multi-start strategies

Due to numerical noise and possible non-convexity of the likelihood with respect to the parameters, gradient-based optimization algorithms may stall far from the global optimum. A common approach to circumvent the issue is to carry out several optimization runs with different initialization points. Two simple strategies can be compared.

Table 5: Two popular reparameterization mappings τ, as implemented, for example, in GPy and STK respectively. For *invsoftplus*, notice parameter $s > 0$, which is introduced when input standardization is considered (see Section 5).

REPARAM. METHOD	$\tau : \mathbb{R}_+^\star \to \mathbb{R}$	$\tau^{-1} : \mathbb{R} \to \mathbb{R}_+^\star$
INVSOFTPLUS(s)	$\log(\exp(\theta/s) - 1)$	$s\log(\exp(\theta') + 1)$
LOG	$\log(\theta)$	$\exp(\theta')$

Restart. In view of Figure 2, a first simple strategy is to restart the optimization algorithm to clear its memory (Hessian approximation, step sizes...), hopefully allowing it to escape a possibly problematic location using the last best parameters as initial values for the next optimization run. The optimization can be restarted a number of times, until a budget N_{opt} of restarts is spent or the best value for the likelihood does not improve.

Multi-start. Given an initialization point $(\theta_{\text{init}}, \mu_{\text{init}}) \in \Theta \times \mathbb{R}$, a multi-start strategy consists in running $N_{\text{opt}} > 1$ optimizations with different initialization points corresponding to perturbations of the initial point $(\theta_{\text{init}}, \mu_{\text{init}})$. In practice, we suggest the following rule for building the perturbations: first, move the range parameters around $(\rho_{1,\text{init}}, \ldots, \rho_{d,\text{init}})^T$ (refer to Section 5 for an implementation); then, propagate the perturbations on μ and σ^2 using (11) and (12). The parameter with the best likelihood value over all optimization runs is selected.

4.4 Parameterization of the covariance function

The parameters of the covariance functions are generally positive real numbers (σ^2, $\rho_1, \rho_2 \ldots$) and are related to scaling effects that act "multiplicatively" on the predictive distributions. Most GP implementations introduce a reparameterization using a monotonic one-to-one mapping $\tau : \mathbb{R}_+^\star \to \mathbb{R}$, acting component-wise on the positive parameters of θ, resulting in a mapping $\tau : \Theta \to \Theta'$. Thus, for carrying out MLE, the actual criterion J that is optimized in most implementations may then be written as

$$J : \theta' \in \Theta' \mapsto -\log(\mathcal{L}(\underline{Z}_n | \tau^{-1}(\theta'), c)) . \tag{13}$$

Table 5 lists two popular reparameterization mappings τ.

The effect of reparameterization is to "reshape" the likelihood. Typical likelihood profiles using the *log* and the so-called *invsoftplus* reparameterizations are shown on Figure 3. Notice that the NLL may be almost flat in some regions depending on the reparameterization. Changing the shape of the optimization criterion, combined with numerical noise, may or may not facilitate the convergence of the optimization.

5 Numerical study

5.1 Methodology

The main metric used in this numerical study is based on empirical cumulative distributions (ECDFs) of differences on NLL values.

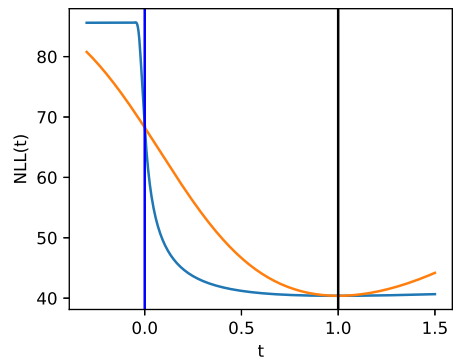


Figure 3: Profiles of the NLL along a linear path t through the *profiled* initialization point (at zero, blue vertical line) and the optimum (at one, black vertical line). Orange (resp. blue) line corresponds to the *log* (resp. *invsoftplus*) reparameterization.

More precisely, consider $N+1$ optimization schemes $S_0, S_1, \ldots, S_N$, where S_0 stands for a "brute-force" optimization scheme based on a very large number of multi-starts, which is assumed to provide a robust MLE, and $S_1, \ldots, S_N$ are optimization schemes to be compared. Each optimization scheme is run on M data sets D_j, $1 \leq j \leq M$, and we denote by $e_{i,j}$ the difference

$$e_{i,j} = \mathrm{NLL}_{i,j} - \mathrm{NLL}_{0,j}, \quad 1 \leq i \leq N, \quad 1 \leq j \leq M,$$

where $\mathrm{NLL}_{i,j}$ the NLL value obtained by optimization scheme S_i on data set D_j.

A good scheme S_i should concentrate the empirical distribution of the sample $E_i = \{e_{i,j}, j = 1, \ldots, M\}$ around zero—in other words, the ECDF is close to the ideal CDF $e \mapsto \mathbb{1}_{[0,\infty[}(e)$. Using ECDF also provides a convenient way to compare performances: a strategy with a "steeper" ECDF, or larger area under the ECDF, is better.

5.2 Optimization schemes

All experiments are performed using GPy version 1.9.9, with the default L-BFGS-B algorithm. We use a common setup and vary the configurations of the optimization levers as detailed below.

Common setup. All experiments use an estimated constant mean-function, an anisotropic Matérn covariance function with regularity $\nu = 5/2$, and we assume no observation noise (the adaptive jitter of GPy ranging from $10^{-6}\sigma^2$ to $10^2\sigma^2$ is used, however).

Initialization schemes. Three initialization procedures from Section 4.1 are considered.

Stopping criteria. We consider two settings for the stopping condition of the L-BFGS-B algorithm, called *soft* (the default setting: `maxiter`= 1000, `factr`=10^7, `pgtol`10^{-5}) and *strict* (`maxiter`= 1000, `factr`=10, `pgtol`= 10^{-20}).

Restart and multi-start. The two strategies of Section 4.3 are implemented using a *log* reparameterization and initialization points $(\theta_{\text{init}}, \mu_{\text{init}})$ determined using a *grid-search* strategy. For the *multi-start* strategy the initial range parameters are perturbed according

to the rule $\rho \leftarrow \rho_{\text{init}} \cdot 10^{\eta}$ where η is drawn from a $\mathcal{N}(0, \sigma_\eta^2)$ distribution. We take $\sigma_\eta = \log_{10}(5)/1.96$ (≈ 0.35), to ensure that about 0.95 of the distribution of ρ is in the interval $[1/5 \cdot \rho_{\text{init}},\ 5 \cdot \rho_{\text{init}}]$.

Reparameterization. We study the *log* reparameterization and two variants of the *invsoftplus*. The first version called *no-input-standardization* simply corresponds to taking $s = 1$ for each range parameter. The second version called *input-standardization* consists in scaling the inputs to a unit standard deviation on each dimension (by taking the corresponding value for s).

5.3 Data sets

The data sets are generated from six well-known test functions in the literature of Bayesian optimization: the Branin function ($d = 2$; see, e.g. Surjanovic and Bingham, 2013), the Borehole function ($d = 8$; see, e.g. Worley, 1987), the Welded Beam Design function ($d = 4$; see Chafekar et al., 2003), the g10 function ($d = 8$; see Ahmed, 2004, p. 128), along with two modified versions, g10mod and g10modmod (see Feliot, 2017).

Each function is evaluated on Latin hypercube samples with a multi-dimensional uniformity criterion (LHS-MDU; Deutsch and Deutsch, 2012), with varying sample size $n \in \{3d, 5d, 10d, 20d\}$, resulting in a total of $6 \times 4 = 24$ data sets.

5.4 Results and findings

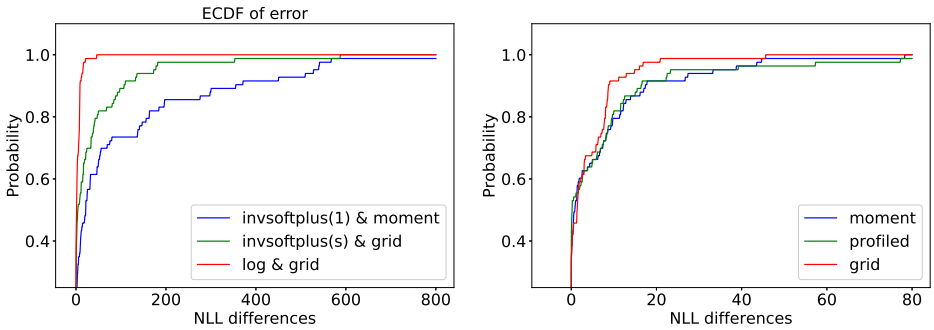
Figure 4 shows the effect of reparameterization and the initialization method. Observe that the *log* reparameterization performs significantly better than the *invsoftplus* reparameterizations. For the *log* reparameterization, observe that the *grid-search* strategy brings a moderate but not negligible gain with respect to the two other initialization strategies, which behave similarly.

Next, we study the effect of the different restart strategies and the stopping conditions, on the case of the *log* reparameterization and *grid-search* initialization. The metric used for the comparison is the area under the ECDFs of the differences of NLLs, computed by integrating the ECDF between 0 and $\text{NLL}_{\max} = 100$. Thus, a perfect optimization strategy would achieve an area under the ECDF equal to 100. Since the *multi-start* strategy is stochastic, results are averaged over 50 repetitions of the optimization procedures (for each N_{opt} value, the optimization strategy is repeated 50 times). The areas are plotted against the computational run time. Run times are averaged over the repetitions in the case of the *multi-start* strategy.

Figure 5 shows that the *soft* stopping condition seems uniformly better. The *restart* strategy yields small improvements using moderate computational overhead. The *multi-start* strategy is able to achieve the best results at the price of higher computational costs.

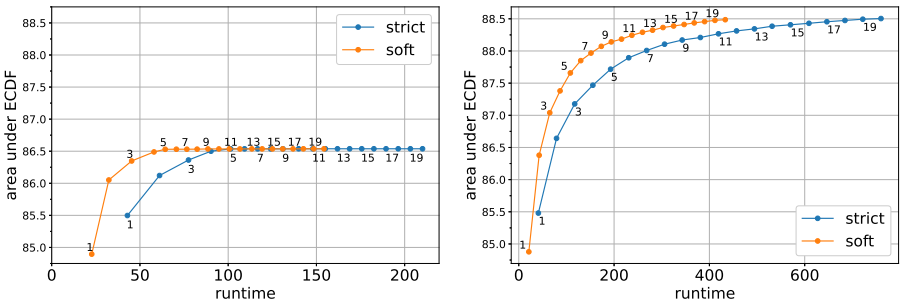
6 Conclusions and recommendations

Our numerical study has shown that the parameterization of the covariance function has the most significant impact on the accuracy of MLE in GPy. Using *restart* / *multi-start* strategies is also very beneficial to mitigate the effect of the numerical noise on the likelihood. The two other levers have second-order but nonetheless measurable influence.



(a) effect of reparameterization (b) effect of initialization

Figure 4: Initialization and reparameterization methods. (a) ECDFs corresponding to the best initialization method for each of the three reparameterizations—red line: *log* reparam. with *grid-search* init.; green line: *invsoftplus* with *input-standardization* reparam. and *grid-search* init; blue line: *invsoftplus* with *no-input-standardization* reparam. and *moment-based* init. (b) ECDFs for different initialization methods for the log reparameterization.



(a) restart with $N_{\text{opt}} = 1, \ldots, 20$ (b) multi-start with $N_{\text{opt}} = 1, \ldots, 20$, $\sigma_\eta = 0.35$

Figure 5: Area under the ECDF against run time: (a) *restart* strategy; (b) *multi-start* strategy. The maximum areas obtained are respectively 86.538 and 88.504.

These observations make it possible to devise a recommended combination of improvement levers—for GPy at least, but hopefully transferable to other software packages as well. When computation time matters, an improved optimization procedure for MLE consists in choosing the combination of a *log* reparameterization, with a *grid-search* initialization, the *soft* (GPy's default) stopping condition, and a small number, say $N_{\text{opt}} = 5$, of restarts.

Figure 1 and Table 2 are based on the above optimization procedure, which results in significantly better likelihood values and smaller prediction errors. The *multi-start* strategy can be used when accurate results are sought.

Several topics could be investigated in the future: the optimization of alternative selection criteria, in particular criteria based on leave-one-out procedures, the case of regression, the problem of parameterization in relation to that of the identifiability of the parameters (see, e.g., Anderes, 2010).

As a conclusion, our recommendations are not intended to be universal, but will hopefully encourage researchers and users to develop and use more reliable and more robust GP implementations, in Bayesian optimization or elsewhere.

Bibliography

Ahmed, A. R. H. A. (2004). *Studies on Metaheuristics for Continuous Global Optimization Problems*. PhD thesis, Kyoto Univ.

Anderes, A. (2010). On the consistent separation of scale and variance for Gaussian random fields. *Ann. Stat.*, 38(2):870–893.

Andrianakis, I. and Challenor, P. G. (2012). The effect of the nugget on Gaussian process emulators of computer models. *Comput. Stat. Data Anal.*, 56(12):4215–4228.

Baudin, M., Dutfoy, A., Iooss, B., and Popelin, A. L. (2017). OpenTURNS: an industrial software for uncertainty quantification in simulation. In Ghanem, R., Higdon, D., and Owhadi, H., editors, *Handbook of Uncertainty Quantification*, pages 2001–2038. Springer, Switzerland.

Bect, J., Vazquez, E., et al. (2011–2021). STK: a Small (Matlab/Octave) Toolbox for Kriging. Release 2.6. http://kriging.sourceforge.net.

Byrd, R., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208.

Chafekar, D., Xuan, J., and Rasheed, K. (2003). Constrained multi-objective optimization using steady state genetic algorithms. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, L. D., Roy, R., O'Reilly, U. M., Beyer, H. G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J., editors, *Genetic and Evolutionary Computation — GECCO 2003*, pages 813–824, Berlin, Heidelberg. Springer.

Curtis, F. E. and Que, X. (2015). A quasi-Newton algorithm for nonconvex, nonsmooth optimization with global convergence guarantees. *Math. Program. Comput.*, 7(4):399–428.

Deutsch, J. L. and Deutsch, C. V. (2012). Latin hypercube sampling with multidimensional uniformity. *J. Stat. Plann. Inference*, 142(3):763–772.

Emmerich, M. T. M., Giannakoglou, K. C., and Naujoks, B. (2006). Single- and multi-objective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Trans. Evol. Comput.*, 10(4):421–439.

Erickson, C. B., Ankenman, B. E., and Sanchez, S. M. (2018). Comparison of Gaussian process modeling software. *Eur. J. Oper. Res.*, 266(1):179–192.

Feliot, P. (2017). *Une approche bayésienne pour l'optimisation multi-objectif sous contrainte*. PhD thesis, Univ. Paris-Saclay.

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

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *J. Global Optim.*, 13(4):455–492.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd Intern. Conf. on Learning Representations, ICLR 2015, San Diego, USA*.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). GPflow: A Gaussian process library using TensorFlow. *J. Mach. Learn. Res.*, 18(40):1–6.

Mockus, J. (1975). On Bayesian methods for seeking the extremum. In Marchuk, G. I., editor, *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pages 400–404, Berlin, Heidelberg. Springer.

Nash, S. G. (1984). Newton-type minimization via the Lanczos method. *SIAM J. Numer. Anal.*, 21(4):770–788.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, USA.

O'Hagan, A. (1978). Curve fitting and optimal design for prediction. *J. R. Stat. Soc. B*, 40:1–24.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830.

Petit, S., Bect, J., Da Veiga, S., Feliot, P., and Vazquez, E. (2020). Towards new cross-validation-based estimators for Gaussian process regression: efficient adjoint computation of gradients. *arXiv:2002.11543*.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical recipes in C. The art of scientific computing*. Cambridge Univ. Press.

Rasmussen, C. E. and Nickisch, H. (2010). Gaussian processes for machine learning (GPML) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, USA.

Roustant, O., Ginsbourger, D., and Deville, Y. (2012). DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *J. Statist. Software*, 51(1):1–55.

Santner, T. J., Williams, B. J., and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments.* Springer series in statistics. Springer.

Sheffield machine learning group (2012–2020). GPy: A Gaussian process framework in Python, version 1.9.9. Available from `http://github.com/SheffieldML/GPy`.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *25th Intern. Conf. on Neural Information Processing Systems. Volume 2*, pages 2951–2959. Curran Associates Inc.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: no regret and experimental design. In *27th Intern. Conf. on Machine Learning (ICML)*, pages 1015–1022.

Stein, M. L. (1999). *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Series in Statistics. Springer New York.

Surjanovic, S. and Bingham, D. (2013). Virtual library of simulation experiments: Test functions and datasets. Retrieved October 13, 2020, from http://www.sfu.ca/~ssurjano/branin.html.

Trefethen, L. N. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM.

Vanhatalo, J., Riihimäki, J., Hartikainen, J., Jylänki, P., Tolvanen, V., and Vehtari, A. (2012). Bayesian modeling with Gaussian processes using the MATLAB toolbox GPstuff (v3.3). *CoRR*, abs/1206.5754.

Wendland, H. (2004). *Scattered data approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge Univ. press.

Worley, B. A. (1987). Deterministic uncertainty analysis. Technical Report ORNL–6428, Oak Ridge National Laboratory, TN, USA.