Numerical issues in maximum likelihood parameter estimation for Gaussian process regression

Subhasish Basak, Sébastien Petit, Julien Bect, Emmanuel Vazquez

Numerical issues in maximum likelihood parameter estimation for Gaussian process regression

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

1. Laboratoire des Signaux et Systèmes, CentraleSupélec, CNRS, Univ. Paris-Saclay, Gif-sur-Yvette, France
2. Safran Aircraft Engines, Moissy-Cramayel, France

Abstract

This article focuses on numerical issues in maximum likelihood parameter estimation for Gaussian process regression (GPR). This article investigates the origin of the numerical issues and provides simple but effective improvement strategies. This work targets a basic problem but a host of studies, particularly in the literature of Bayesian optimization, rely on off-the-shelf GPR implementations. For the conclusions of these studies to be reliable and reproducible, robust GPR implementations are critical.

1   Introduction

Gaussian process regression (GPR; 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 GPR has also raised considerably due to the development of Bayesian optimization (Mockus, 1975; Jones et al., 1998; Emmerich et al., 2006; Villemonteix et al., 2009; Srinivas et al., 2010, . . .).

This context has fostered the development of a fairly large number of open-source packages to facilitate the use of GPR. Some of the popular choices are the Python modules scikit-learn (Pedregosa et al., 2011), GPy (GPy authors, 2019), GPflow (Matthews et al., 2017), GPyTorch (Gardner et al., 2018), OpenTURNS (Baudin et al., 2015); the R package DiceKriging (Roustant et al., 2012); and the MATLAB/Octave toolboxes GPML (Rasmussen and Nickisch, 2010), STK (Bect et al., 2019) and GPstuff (Vanhatalo et al., 2012).

Table 1: Inconsistencies in the results across different python packages. The results were obtained by fitting a GPR 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 variance and length scale parameters of the kernel, along with the empirical root mean squared prediction error (ERMSPE). The last line shows the improvement using the recommendations in this study.

LIBRARY	Variance	Length scales	ERMSPE
SCIKIT-LEARN 0.23.2	$9.9 \cdot 10^4$	$(13, 43)$	1.482
GPY 1.9.9	$1.8 \cdot 10^9$	$(100, 590)$	0.237
GPYTORCH 1.2.1	$1.7 \cdot 10^0$	$(2, 2)$	12.770
GPFLOW 1.3.0	$1.3 \cdot 10^9$	$(95, 540)$	0.245
OPENTURNS 1.13	$1.1 \cdot 10^2$	$(8, 19)$	3.301
GPY "IMPROVED"	$9.4 \cdot 10^{10}$	$(220, 1500)$	0.175

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 select the model parameters, we can safely say that the most popular approach consists is the *maximum likelihood estimation* (MLE) method—thus, GPR involves an optimization task. However, a simple numerical experiment summarized in Table 1 shows that different MLE implementations from different Python packages produce dramatically dispersed numerical results, even when the same data and Gaussian process prior are used. (Moreover, it turns out that none of the standard results in Table 1 are really satisfactory compared to the result using the recommendations in this study.) These stark differences were also observed by Erickson et al. (2018).

In this article, our first objective is to understand the origin

of the inconsistencies across available implementations. Our second objective is to provide simple but effective strategies for improving these implementations, with a focus on the well-established GPy package. We shall propose solutions and 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 leave-one-out mean squared error (LOO-MSE) 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 GPR. Indeed, a host of studies, particularly in the literature of Bayesian optimization, rely on off-the-shelf GPR implementations: for the conclusions of these studies to be reliable and reproducible, robust GPR implementations are critical.

The article is organized as follows. Section 2 provides a brief review of GPR 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 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.

Table 2: *Improved* (cf. Section 6) vs *default* setups in GPy for the Borehole function with $n \in \{5d, 10d, 20d\}$ data points (input space dimension is $d = 8$). The columns report the LOO-MSE values, and parentheses give the ratio of the LOO-MSE to the total standard deviation of the data.

METHOD	$n = 5d$	$n = 10d$	$n = 20d$
DEFAULT	6.04 (0.12)	2.84 (0.06)	1.59 (0.03)
IMPROVED	0.87 (0.017)	0.23 (0.005)	0.12 (0.002)

2 Background

2.1 Gaussian processes

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

In the GPR framework, the objective is to predict $\xi(x)$ 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: $Z_i = \xi(x_i) + \varepsilon_i$, $1 \leq i \leq n$. 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 ξ. (In rarer cases, heteroscedasticity

is assumed.)

Knowing m and k, recall (see, e.g. Rasmussen and Williams, 2006) that the posterior distribution of ξ is such that

$$\xi \mid Z_1, \ldots, Z_n, m, k \sim GP(\widehat{\xi}_n, k_n),$$

where $\widehat{\xi}_n$ and k_n stand respectively for the posterior mean and posterior covariance. The posterior mean at $x \in \mathbb{R}^d$ can be written as

$$\widehat{\xi}_n(x) = m(x) + \sum_{i=1}^{n} w_i(x; \underline{x}_n)(Z_i - m(x_i)), \qquad (1)$$

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

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

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

$$k_n(x, y) = k(x, y) - \mathrm{w}(y; \underline{x}_n)^\mathsf{T} \mathrm{K}(\underline{x}_n, x). \qquad (3)$$

It is common practice in GPR to assume a zero mean function $m = 0$—a reasonable choice if the user has taken care to center data—but most GPR tools also provide an option for setting a constant mean function $m(\cdot) = c \in \mathbb{R}$. 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, c 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 GPR numerical implementations provide a couple of stationary covariance functions taken from the literature (e.g., Wendland, 2004; Rasmussen and Williams, 2006). The *squared exponential* covariance function is often seen as a "default" covariance function, especially when the function to be learned is believed to be very smooth, but *rational quadratic* or *Matérn* covariance functions are also popular choices (see Table 3). These covariance functions include a number of parameters: a variance parameter $\sigma^2 > 0$ corresponding to the variance of ξ, and a set of range (or length scale) parameters $\rho_1, \ldots, \rho_d$, such that

$$k(x, y) = \sigma^2 r(h), \qquad (4)$$

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 (4) is the stationary correlation function of ξ, which may also be simply called *kernel*.

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 $\mathrm{K}(\underline{x}_n, \underline{x}_n) + \sigma_\varepsilon^2 \mathrm{I}_n$ by K_θ.

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

(a) optimized NLL
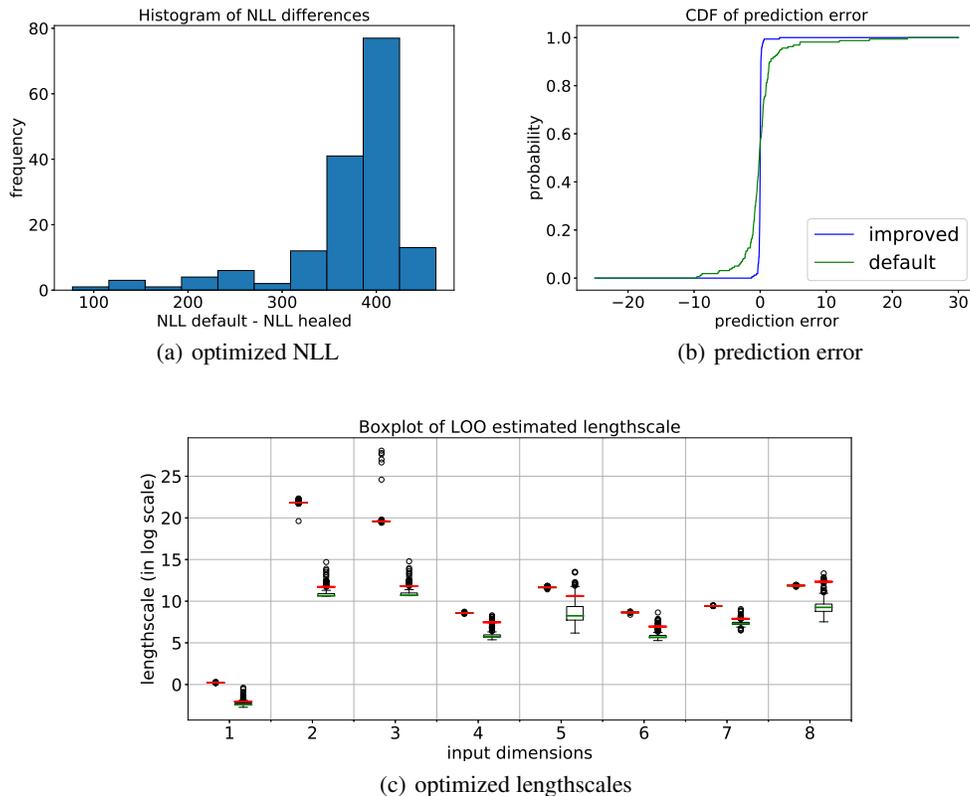
(b) prediction error

(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 CDFs of the prediction errors at points removed; (c) pairs of box-plots for the estimated range parameters (for each dimension, 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 for the whole data set). Notice that the parameter distributions of the *default* setup are more spread out.

Table 3: Some kernel functions available in GPy. The Matérn kernel is recommended by Stein (1999). Γ denotes the gamma function, $\mathcal{K}_\nu$ is the modified Bessel function of the second kind (Abramowitz and Stegun, 1964).

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

2.2 Maximum likelihood parameter estimation

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

log-likelihood (NLL)

$$-\log(\mathcal{L}(\underline{Z}_n | \theta, c)) = \frac{1}{2}(\underline{Z}_n - c\mathbb{1}_n)^\top K_\theta^{-1}(\underline{Z}_n - c\mathbb{1}_n)$$
$$+ \frac{1}{2}\log|K_\theta| + \frac{n}{2}\log(2\pi). \quad (5)$$

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 w.r.t. (θ, c) has a

$$O(n^3 + dn^2)$$

computational cost (Rasmussen and Williams, 2006; Petit et al., 2020).

3 Numerical noise on the likelihood

The evaluation of the likelihood as well as its gradient is subject to numerical noise, which can prevent proper convergence of the optimization algorithms. To illustrate this fact,

we show a typical situation on Figure 2 where the gradient-based optimization algorithm stops before converging to a good minimum. Numerical noise stems from both terms of the likelihood, namely $\frac{1}{2}\underline{Z}_n^\top K_\theta^{-1}\underline{Z}_n$ and $\frac{1}{2}\log|K_\theta|$. (For simplification, we assume $c = 0$ in this section.)

It is quite common, as in the case of GPy, to compute the first term by solving the linear system of equations $K_\theta \, a = Z_n$ using a Cholesky decomposition of K_θ. Recall that the condition number $\kappa(K_\theta)$ of K_θ, defined as the ratio $|\lambda_{\max}/\lambda_{\min}|$ of the largest eigenvalue to the smallest eigenvalue is a key element for analyzing numerical noise (Press et al., 1992). 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

$$\frac{\|K_\theta^{-1}\varepsilon\|}{\|K_\theta^{-1}\underline{Z}_n\|} \simeq \kappa(K_\theta) \times 10^{-16}, \tag{6}$$

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

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

$$\kappa(\log|\cdot|, K_\theta)$$
$$\triangleq \lim_{\varepsilon \to 0} \sup_{\|\delta_A\|_F \le \varepsilon} \frac{\big|\log|K_\theta + \delta_A| - \log|K_\theta|\big|/\big|\log|K_\theta|\big|}{\|\delta_A\|_F/\|K_\theta\|_F}$$
$$= \frac{\sqrt{\sum_{i=1}^n \frac{1}{\lambda_i^2}}\sqrt{\sum_{i=1}^n \lambda_i^2}}{\big|\sum_{i=1}^n \log(\lambda_i)\big|} \tag{7}$$

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

$$\frac{\kappa(K_\theta)}{\big|\sum_{i=1}^n \log(\lambda_i)\big|} \le \kappa(\log|\cdot|, K_\theta) \le \frac{n\kappa(K_\theta)}{\big|\sum_{i=1}^n \log(\lambda_i)\big|}, \tag{8}$$

which shows that numerical noise on $\log|K_\theta|$ is linked again to the condition number of K_θ.

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

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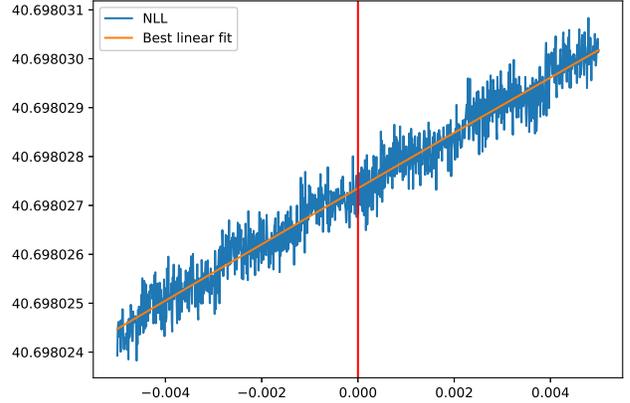

Figure 2: Noisy likelihood profile along a particular direction in the parameter space, with a best linear fit (orange line). This example was obtained with GPy while estimating the parameters of a Matérn-$v = 5/2$ covariance, using 20 data points sampled from a *Branin function*, and setting $\sigma_\varepsilon^2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

Thus, when $\kappa(K_\theta)$ 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 $\kappa(K_\theta)$ 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 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 GPR 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 GPR when σ_ε^2 is increased. It appears that finding a satisfying trade-of between good interpolation properties and low numerical noise level

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

can be difficult. (Table 4 also supports the connection in (6) and (8) between noise levels and $\kappa(K_\theta)$.)

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 Simple strategies for improving implementations of MLE

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

4.1 Initialization strategies

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

4.1.1 Moment-based initialization

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

$$
\begin{align}
c_{\text{init}} &= \text{mean}(Z_1, \ldots, Z_n) \tag{9}\\
\sigma^2_{\text{init}} &= \text{var}(Z_1, \ldots, Z_n) \tag{10}\\
\rho_{k,\text{init}} &= \text{std}(x_{1,[k]}, \ldots, x_{n,[k]}), \quad k = 1, , \ldots, d, \tag{11}
\end{align}
$$

where mean, var and std stand for the empirical mean, variance and standard deviation, and $x_{i,[k]}$s denotes the kth coordinate of the point $x_i \in \mathbb{R}^d$. The rationale behind (11) (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 before the function value changes 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 $c = 0$, $\sigma^2 = 1$ and $\rho_k = 1$ for all $k = 1, \ldots, d$. This is equivalent to the *moment-based* initialization scheme when the data (both inputs and outputs) are centered and standardized.

4.1.2 Profiled initialization

Assume the range parameters $\rho_1, \ldots, \rho_d$ (and more generally, all parameters different from σ^2, σ^2_ε and c) are fixed,

and set $\sigma^2_\varepsilon = \alpha\sigma^2$, with a prescribed multiplicative factor $\alpha \geq 0$. In this case, the NLL can be optimized analytically w.r.t. c and σ^2. Optimal values turn out to be the generalized least squares solutions

$$
\begin{align}
c_{\text{GLS}} &= (\mathbb{1}_n^\mathsf{T} K_{\tilde{\theta}}^{-1} \mathbb{1}_n)^{-1} \mathbb{1}_n^\mathsf{T} K_{\tilde{\theta}}^{-1} \underline{Z}_n, \tag{12}\\
\sigma^2_{\text{GLS}} &= \frac{1}{n}(\underline{Z}_n - \hat{c}\mathbb{1}_n)^\mathsf{T} K_{\tilde{\theta}}^{-1}(\underline{Z}_n - \hat{c}\mathbb{1}_n), \tag{13}
\end{align}
$$

where $\tilde{\theta} = (\sigma^2, \rho_1, \ldots, \rho_d, \ldots, \sigma^2_\varepsilon)^\mathsf{T} \in \Theta$, with $\sigma^2 = 1$ and $\sigma^2_\varepsilon = \alpha$.

Under the *profiled* initialization scheme, $\rho_1, \ldots, \rho_d$ are set using (11), α is prescribed according to user's preference, and c and σ^2 are initialized using (12) and (13).

4.1.3 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 as

$$
\rho_0 = \sqrt{d}\left(\max_{1 \leq i \leq n} x_{i,[1]} - \min_{1 \leq i \leq n} x_{i,[1]}, \ldots, \max_{1 \leq i \leq n} x_{i,[d]} - \min_{1 \leq i \leq n} x_{i,[d]}\right)^\mathsf{T}.
$$

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 c and σ^2 using (12) and (13). 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 GPR implementations rely on well-tested gradient-based optimization algorithms. For instance, a popular choice in Python implementations is to call the limited-memory BFGS 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 optimization, 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 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

Table 4: Influence of the jitter on the GPR model. $\widehat{\sigma}_{\text{quad}}$ and $\widehat{\sigma}_{\log|\cdot|}$ stand for the empirical standard deviations of the numerical noise on $\underline{Z}_n^T K_\theta^{-1} \underline{Z}_n$ and $\log|K_\theta|$ respectively, measured using second-order polynomial regressions. The table also reports the condition numbers $\kappa(K_\theta)$ and $\kappa(\log|\cdot|, K_\theta)$, the interpolation error $\sqrt{\text{SSR/SST}} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(Z_j - \widehat{\xi}_n(x_j))^2}/\text{std}(Z_1,...,Z_n)$, and the NLL. Reducing numerical noise while keeping good interpolation properties requires careful attention in practice.

$\sigma_\varepsilon^2 / \sigma^2$	0.0	10^{-8}	10^{-6}	10^{-4}	10^{-2}
$\widehat{\sigma}_{\text{quad}} / \underline{Z}_n^T K_\theta^{-1} \underline{Z}_n$	10^{-8}	$10^{-9.5}$	$10^{-10.5}$	10^{-12}	10^{-14}
$\widehat{\sigma}_{\log|\cdot|} / \log|K_\theta|$	$10^{-7.5}$	10^{-9}	10^{-11}	$10^{-13.5}$	$10^{-15.5}$
$\kappa(\log|\cdot|, K_\theta)$	$10^{9.5}$	$10^{8.5}$	$10^{6.5}$	$10^{4.5}$	$10^{2.5}$
$\kappa(K_\theta)$	10^{11}	10^{9}	$10^{7.5}$	$10^{5.5}$	$10^{3.5}$
$\sqrt{\text{SSR/SST}}$	$3.3 \cdot 10^{-10}$	$1.2 \cdot 10^{-3}$	0.028	0.29	0.75
$-\log(\mathcal{L}(\underline{Z}_n|\theta))$	40.69	45.13	62.32	88.81	124.76

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.

Restart— In the 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 possible 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 $(\theta_{\text{init}}, c_{\text{init}}) \in \Theta \times \mathbb{R}$, a multi-start strategy consists in running $N_{\text{opt}} > 1$ optimizations with different initialization points corresponding to perturbations of the initial point $(\theta_{\text{init}}, c_{\text{init}})$. 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 c and σ^2 using (12) and (13). 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 ($\sigma^2, \rho_1, \rho_2 \ldots$) and are related to scaling effects that act "multiplicatively" on the predictive distributions. Since it is easier to carry out optimization on unbounded domains, most GPR implementations introduce a reparameterization using a monotonic one-to-one mapping $\delta : \mathbb{R}_+^\star \to \mathbb{R}$, acting component-wise on the positive parameters of θ, resulting in a mapping $\Delta : \Theta \to \Theta'$. Thus, for carrying out MLE, the actual criterion J that is optimized in most implementations may then be written as

$$J : \theta' \in \Theta' \mapsto -\log(\mathcal{L}(\underline{Z}_n|\Delta^{-1}(\theta'), c)). \qquad (14)$$

Table 5 lists two popular reparameterization mappings δ.

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.

Table 5: Two popular reparameterization mappings δ, 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.	$\delta : \mathbb{R}_+^\star \to \mathbb{R}$	$\delta^{-1} : \mathbb{R} \to \mathbb{R}_+^\star$
INVSOFTPLUS(s)	$\log(\exp(\theta/s) - 1)$	$s\log(\exp(\delta) + 1)$
LOG	$\log(\theta)$	$\exp(\delta)$

5 Numerical study

In this section we empirically quantify the effects of the levers of Section 4.

5.1 Methodology

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

More precisely, consider $N + 1$ optimization schemes $S_0, S_1, \ldots, S_N$. S_0 is a "bruteforce" optimization scheme based on a very large number of multi-starts which is assumed to provide a very robust MLE. $S_1, \ldots, S_N$ are optimization schemes to be tested, corresponding

Subhasish Basak [1], Sébastien Petit[1,2], Julien Bect[1], Emmanuel Vazquez[1]
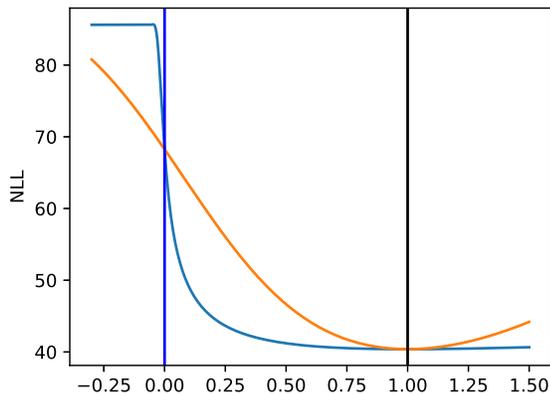
Figure 3: Profiles of the NLL along a linear path through the *profiled* initialization point (at zero, blue vertical line) and the optimum (at one, black vertical line). Orange (resp. blue) line corresponds to the *log* (resp. *invsoftplus*) reparameterization.

to different combinations for the levers of Section 4. Each optimization scheme is run on M data sets $D_j = \left\{ (x_i, z_i) \in \mathbb{R}^{d_j} \times \mathbb{R}; i = 1, \dots, n_j \right\}, j = 1, \dots, M$.

Denote by $\text{NLL}_{i,j}$ the estimation of the optimal NLL value obtained by optimization scheme S_i on data set D_j, and let $e_{i,j}, i = 1, \dots, N, j = 1, \dots, M$, be the difference

$$e_{i,j} = \text{NLL}_{i,j} - \text{NLL}_{0,j}.$$

A good scheme S_i should concentrate the empirical distribution of the sample $E_i = \{e_{i,j}, j = 1, \dots, M\}$ 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 parameter $\nu = 5/2$, and assumes no observation noise (but we use the adaptive jitter of GPy ranging from $10^{-6}\sigma^2$ to $10^2\sigma^2$).

Initialization schemes— The three initialization procedures detailed in 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) and *strict*. Details are given in Table 6.

Table 6: *Soft* and *Strict* settings for the stopping of the L-BFGS-B optimizer.

CONFIGURATION	SOFT (DEFAULT)	STRICT
Max. # steps (`maxiter`)	1000	1000
Step size param. (`factr`)	10^7	10
Gradient threshold (`pgtol`)	10^{-5}	10^{-20}

Restart and multi-start— The two strategies of Section 4.3 are implemented using a *log* reparameterization and initialization points $(\theta_{\text{init}}, c_{\text{init}})$ determined using a *grid-search* strategy. For the *multi-start* strategy the initial range parameters are perturbed according to the rule $\rho \leftarrow 10^{\log_{10}(\rho)+\eta}$ where η is drawn from a $\mathcal{N}(0, \sigma_\eta^2)$ distribution with $\sigma_\eta^2 = 3$.

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 component of the range parameters. The second version called *input-standardization* consists in scaling the inputs to a unit standard deviation on each dimension (by taking the corresponding values 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 WeldedBeam 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. Since the *multi-start* strategy is stochastic,

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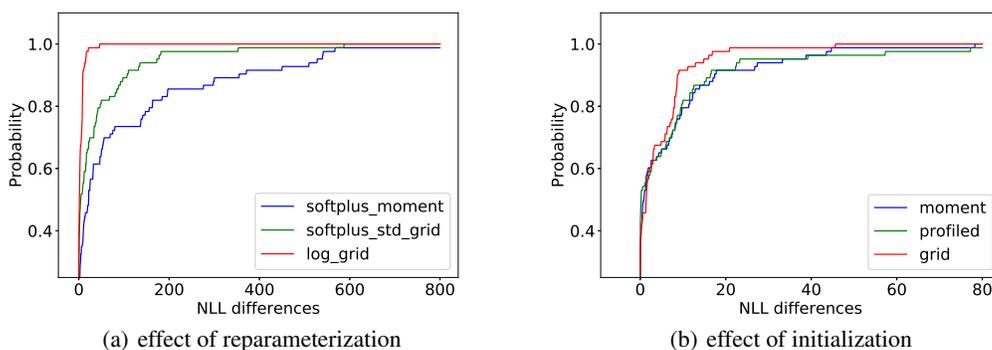
(b) effect of initialization

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 $N_{\mathrm{opt}} = 1, \ldots, 20$

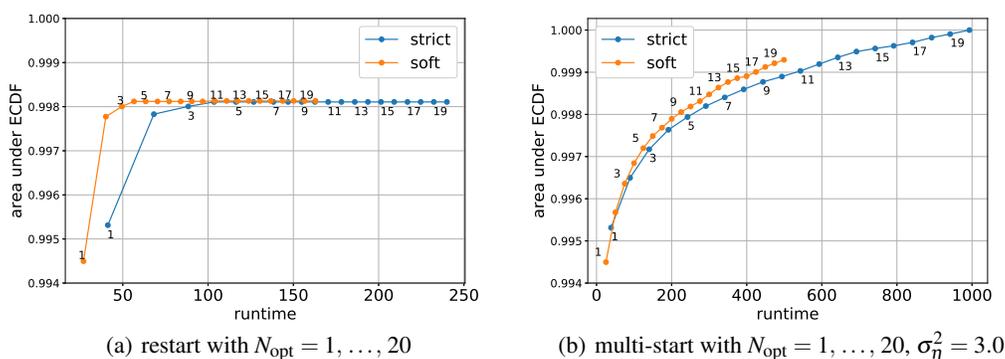(b) multi-start with $N_{\mathrm{opt}} = 1, \ldots, 20$, $\sigma_\eta^2 = 3.0$

Figure 5: Area under the ECDF against run time: (a) *restart* strategy; (b) *multi-start* strategy. The y-axes in (a) and (b) are normalized w.r.t the maximum area 387.01.

results are averaged over 50 repetitions of the optimization procedures (for each N_{opt} value, the optimization strategy is repeated 50 times). To make comparisons easier, the areas are normalized, with a normalizing factor equal to the area under the ECDF of the best strategy, namely, the *multi-start* strategy with $N_{\mathrm{opt}} = 20$. 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 better improvements than the *multi-start* strategy for small computational budgets, but 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.

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 $N_{\mathrm{opt}} = 5$, 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.

As a conclusion, we believe that this elementary empirical study is a worthwhile contribution, since it tackles a basic building block of sometimes very elaborate GPR-based methods (in Bayesian optimization or elsewhere). Our recommendations are not intended to be universal, but will hopefully encourage researchers and users to develop and use more reliable and more robust GPR implementations.

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

References

Abramowitz, M. and Stegun, I. A. (1964). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth dover printing, tenth gpo printing edition.

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

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

Baudin, M., Dutfoy, A., Iooss, B., and Popelin, A.-L. (2015). OpenTURNS: an industrial software for uncertainty quantification in simulation. In Ghanem, R., Higdon, D., and Owhadi, H., editors, *Handbook of Uncertainty Quantification*. Springer, Cham, Switzerland.

Bect, J., Vazquez, E., et al. (2019). 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 Journal on Scientific Computing*, 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. *Journal of Statistical Planning and Inference*, 142(3):763–772.

Emmerich, M. T., Giannakoglou, K. C., and Naujoks, B. (2006). Single-and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439.

Erickson, C. B., Ankenman, B. E., and Sanchez, S. M. (2018). Comparison of Gaussian process modeling software. *European Journal of Operational Research*, 266(1):179–192.

Feliot, P. (2017). *Une approche bayésienne pour l'optimisation multi-objectif sous contrainte*. PhD thesis, Université 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 Neural Information Processing Systems*.

GPy authors (2019). GPy: A Gaussian process framework in Python, version 1.9.9. Available from http://github.com/SheffieldML/GPy.

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

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

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 Berlin Heidelberg.

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

O'Hagan, A. (1978). Curve fitting and optimal design for prediction. *Journal of the royal statistical society series b-methodological*, 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. *Journal of Machine Learning Research*, 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.

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

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

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

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. *Journal of Statistical 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.

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

Stein, M. (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.

Villemonteix, J., Vazquez, E., and Walter, E. (2009). An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509.

Wendland, H. (2004). *Scattered data approximation*, volume 17. Cambridge university press.

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