hetGP: Heteroskedastic Gaussian Process Modeling and Sequential Design in R

Mickaël Binois
Argonne National Laboratory

Robert B. Gramacy
Virginia Tech

Abstract

An increasing number of time-consuming simulators exhibit a complex noise structure that depends on the inputs. For conducting studies with limited budgets of evaluations, new surrogate methods are required in order to model simultaneously the mean and variance fields. To this end, we present the **hetGP** package, implementing many recent advances in Gaussian process modeling with input-dependent noise. First, we describe a simple, yet efficient, joint modeling framework that relies on replication for both speed and accuracy. Then we tackle the issue of leveraging replication and exploration in a sequential manner for various goals, such as for obtaining a globally accurate model, for optimization, or for contour finding. Reproducible illustrations are provided throughout.

Keywords: input-dependent noise, level-set estimation, optimization, replication, stochastic kriging.

1. Introduction

Historically, computer experiments have been associated with deterministic black-box functions; see, for example, Sacks, Welch, Mitchell, and Wynn (1989). Gaussian process (GP) interpolators are the canonical model in this setting. Recently, determinism has become less common for more complex simulators, e.g., those relying on agents. Stochastic simulation has opened up many avenues for inquiry in the applied sciences. Data in geostatistics (Banerjee, Carlin, and Gelfand 2004) and machine learning (Rasmussen and Williams 2006), where high-fidelity modeling also involves GPs, not only is noisy but frequently involves signal-to-noise ratios that may be low or changing over the input space. In this context, whether for stochastic simulation or for data from observational studies, more samples are needed in order to isolate the signal, and learning the variance is at least as important as the mean. Yet GPs buckle under the weight of even modestly big data. Moreover, few options for heteroskedastic modeling exist.

Replication provides a powerful tool for separating signal from noise, yields computational savings, and holds the potential to capture input-dependent dynamics in variance. In the operations research community, stochastic kriging (SK; Ankenman, Nelson, and Staum 2010) offers approximate methods that exploit large degrees of replication, coupling GP modeling on the mean and the variance, likelihood, and method-of-moments-based inference, toward efficient heteroskedastic modeling. Binois, Gramacy, and Ludkovski (2018a) provide similar results but relaxing the necessity of having a large degree of replication, and places inference fully within a likelihood framework. That methodology is the primary focus of the implementation described here. Other approaches for achieving a degree of input-dependent variance include quantile kriging (Plumlee and Tuo 2014); the use of pseudoinputs (Snelson and Ghahramani 2005), also sometimes called a predictive process (Banerjee, Gelfand, Finley, and Sang 2008); and (non-GP-based) tree methods (Pratola, Chipman, George, and McCulloch 2017a). Although the **hetGP** package has many aspects in common, and in some cases is directly inspired by these approaches, none of these important methodological contributions are, to our knowledge, coupled with an open source R implementation.

In the computer experiments and machine learning communities, sequential design of experiments/active learning (e.g., Seo, Wallat, Graepel, and Obermayer 2000; Gramacy and Lee 2009) and Bayesian optimization (e.g., Snoek, Larochelle, and Adams 2012) are a primary goal of modeling. The **hetGP** package provides hooks for learning and design to reduce predictive variance, organically determining the input-dependent degree of replication required to efficiently separate signal from noise (Binois, Huang, Gramacy, and Ludkovski 2018b); for Bayesian optimization; for contour finding (Lyu, Binois, and Ludkovski 2018), and for leptokurtic responses (Shah, Wilson, and Ghahramani 2014; Chung, Binois, Gramacy, Moquin, Smith, and Smith 2018). A description and examples are provided herein.

Although many R packages are available on CRAN for GP spatial and computer surrogate modeling (e.g., Erickson, Ankenman, and Sanchez 2017, provide a nice empirical review and comparison), we are not aware of any others that provide an efficient approach to fully likelihood-based coupled GP mean and variance modeling for heteroskedastic processes and, moreover, that provide a comprehensive approach to sequential design in that context. The **tgp** (Gramacy 2007; Gramacy and Taddy 2010) and **laGP** (Gramacy 2016) packages offer a limited degree of nonstationary and heteroskedastic GP modeling features via partitioning, which means they cannot capture smooth dynamics in mean and variance as **hetGP** can, nor can they efficiently handle replication in the design. The **DiceKriging** (Roustant, Ginsbourger, and Deville 2012) package is often used for SK experiments; however, users must preprocess the data and calculate their own moment-based variance estimates. The **mlegp** package (Dancik and Dorman 2008) offers a more hands-off SK experience, facilitating replicate detection, but without coupled modeling and sequential design features. Other R packages include **GPfit** (MacDonald, Ranjan, and Chipman 2015), which focuses expressly on deterministic modeling, whereas those like **kergp** (Deville, Ginsbourger, and Durrande. 2018) target flexible covariance kernels, including additive or qualitative versions, in the homoskedastic setting.

The remainder of this paper is organized as follows. Section 2 reviews **hetGP**'s approach to ordinary (homoskedastic) GP regression, and linear algebra identities that enable substantial speedups when the design has a high degree of replication. Section 3 covers **hetGP**'s latent variance approach to joint GP likelihood-based inference to accommodate heteroskedastic processes. Section 4 covers sequential design for **hetGP** models targeting reduction in prediction error, Bayesian optimization, and contour finding. We conclude in Section 5 with a brief summary and discussion.

2. Gaussian process modeling under replication

A computer experiment involves a function $f(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^d \to \mathbb{R}$ requiring expensive simulation to *approximate* a real-world (physical) relationship. An *emulator* $\hat{f}_N$ is a regression or response surface fit to input-output pairs $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$, where $y_i \sim f(\mathbf{x}_i)$. Predictive equations from $\hat{f}_N$, notated as $\hat{f}_N(\mathbf{x}')$ for a new location $\mathbf{x}'$, may serve as a cheap *surrogate* for $f(\mathbf{x}')$ at new inputs $\mathbf{x}'$ for visualization, sensitivity analysis, optimization, and so forth. The **hetGP** package targets thrifty GP surrogates for stochastic computer model simulations whose variance and mean are both believed to vary nonlinearly.

Although our emphasis and vocabulary are tilted toward computer surrogate modeling throughout, many of the techniques we describe are equally well suited to applications in machine learning and geostatistics or anywhere else GP regression may be applied. Many of the methods presented here are inspired by advances in those areas. We begin by reviewing **hetGP**'s approach to conventional, homoskedastic GP modeling and a linear algebra trick that enables thrifty computation when the design involves a large degree of replication relative to the number of unique sites where computer simulation responses are measured.

2.1. Gaussian process review

Gaussian process regression is a form of spatial modeling via multivariate normal (MVN) distributions. In the computer surrogate modeling community, one commonly takes a mean-zero GP formulation, moving all the modeling action to the covariance structure, which is usually specified as a decreasing function of distance. The formulation below uses a scaled separable (or anisotropic) Gaussian kernel.

$$\mathbf{Y}_N \sim \mathcal{N}_N(0, \nu \mathbf{K}_N) \tag{1}$$

$$\text{with} \quad \mathbf{K}_N = (K_\theta(\mathbf{x}_i, \mathbf{x}_j) + \tau^2 \delta_{i=j})_{1 \leq i,j \leq N}, \text{ and } K_\theta(\cdot,\cdot) = \exp\left\{ -\sum_{k=1}^{d} \frac{(x_k - x'_k)^2}{\theta_k} \right\}$$

Lengthscales $\boldsymbol{\theta} = (\theta_1, \dots, \theta_p)$ govern the rate of decay of correlation as a function of coordinate-wise distance; *scale* ν adjusts the amplitude of $\mathbf{Y}_N$ realizations, and *nugget* τ^2 implements an iid (i.e., nonspatial) variance component for noisy data. Historically, when (almost exclusively) modeling deterministic computer simulations, a zero nugget ($\tau^2 = 0$) is common, although authors have argued that sometimes that can be inefficient (Gramacy and Lee 2012). Other forms for the correlation kernel $K_\theta(\cdot,\cdot)$ are common, and we shall discuss several others and their "hyperparameters" in due course. In the computer experiments literature, one commonly takes a mean of zero; however, parameterized extensions are common. Although we shall mostly assume a mean of zero throughout for notational conciseness, extensions are relatively straightforward. For example, the **hetGP** allows a constant mean to be estimated; and in the heteroskedastic setup discussed further, this is how the mean variance is estimated.

GPs make popular surrogates because their predictions are accurate, have appropriate coverage, and can interpolate when the nugget is zero. The beauty of GP modeling is evident in the form of their predictive equations, which are a simple application of conditioning for MVN joint distributions. Let $D_N = (\mathbf{X}_N, \mathbf{Y}_N)$ denote the data. Generically, for covariance structure $\Sigma(\cdot,\cdot)$, GP prediction $Y(\mathcal{X}) \mid D_N$ at new locations $\mathcal{X}$ follows

$$Y(\mathcal{X}) \mid D_N \sim \mathcal{N}_{n'}(\mu(\mathcal{X}), \boldsymbol{\Sigma}(\mathcal{X}))$$

with

$$\text{mean} \quad \mu(\mathcal{X}) = \boldsymbol{\Sigma}(\mathcal{X}, \mathbf{X}_N) \boldsymbol{\Sigma}_N^{-1} \mathbf{Y}_N \tag{2}$$

$$\text{and variance} \quad \boldsymbol{\Sigma}(\mathcal{X}) = \boldsymbol{\Sigma}(\mathcal{X}, \mathcal{X}) - \boldsymbol{\Sigma}(\mathcal{X}, \mathbf{X}_N) \boldsymbol{\Sigma}_N^{-1} \boldsymbol{\Sigma}(\mathcal{X}, \mathbf{X}_N)^\top, \tag{3}$$

where $\mathbf{\Sigma}(\mathcal{X}, \mathbf{X}_N)$ is an $n' \times N$ matrix. These are the so-called kriging equations in spatial statistics. They describe the best (minimizing MSPE) linear unbiased predictor (BLUP).

If the covariance structure is hyperparameterized, as it is in Equation (1), the multivariate structure can emit a log likelihood that may be utilized for inference for unknown hyperparameters $(\nu, \boldsymbol{\theta}, \tau^2)$:

$$\ell = \log L = -\frac{N}{2} \log 2\pi - \frac{N}{2} \log \nu - \frac{1}{2} \log |\mathbf{K}_N| - \frac{1}{2\nu} \mathbf{Y}_N^\top \mathbf{K}_N^{-1} \mathbf{Y}_N.$$

To maximize ℓ with respect to ν, say, one just differentiates and solves:

$$\begin{aligned} 0 \overset{\text{set}}{=} \ell'(\nu) \equiv \frac{\partial \ell}{\partial \nu} &= -\frac{N}{2\nu} + \frac{1}{2\nu^2} \mathbf{Y}_N^\top \mathbf{K}_N^{-1} \mathbf{Y}_N \\ \hat{\nu} &= \frac{\mathbf{Y}_N^\top \mathbf{K}_N^{-1} \mathbf{Y}_N}{N}. \end{aligned}$$

The quantity $\hat{\nu}$ is like a mean residual sum of squares. Now, plugging $\hat{\nu}$ into ℓ gives the so-called concentrated log-likelihood,

$$\begin{aligned} \ell(\tau^2, \boldsymbol{\theta}) &= -\frac{N}{2} \log 2\pi - \frac{N}{2} \log \hat{\nu} - \frac{1}{2} \log |\mathbf{K}_N| - \frac{1}{2\hat{\nu}} \mathbf{Y}_N^\top \mathbf{K}_N^{-1} \mathbf{Y}_N \\ &= c - \frac{N}{2} \log \mathbf{Y}_N \mathbf{K}_N^{-1} \mathbf{Y}_N - \frac{1}{2} \log |\mathbf{K}_N|. \end{aligned}$$

Using the chain rule and that

$$\frac{\partial \mathbf{K}_N^{-1}}{\partial \phi} = -\mathbf{K}_N^{-1} \frac{\partial \mathbf{K}_N}{\partial \phi} \mathbf{K}_N^{-1} \quad \text{and} \quad \frac{\partial \log |\mathbf{K}_N|}{\partial \phi} = \mathrm{tr}\left\{ \mathbf{K}_N^{-1} \frac{\partial \mathbf{K}_N}{\partial \phi} \right\}$$

yields closed-form expressions for the partial derivatives with respect to $(\theta_1, \ldots, \theta_k)$ and τ^2. Unfortunately these cannot be set to zero and solved analytically. But numerical methods are fairly good.

To illustrate, and to expose the essence of the implementation automated in the **hetGP** package (but for more involved settings discussed momentarily), we provide the following hand-coded example. The code block below implements the negative log-likelihood for parameters `par`, whose first `ncol(X)` settings correspond to the lengthscales $\boldsymbol{\theta}$, followed by the nugget τ^2.

```
R> library("hetGP")
R> nl <- function(par, X, Y) {
+    theta <- par[1:ncol(X)]
+    tau2 <- par[ncol(X) + 1]
+    n <- length(Y)
+    K <- cov_gen(X1 = X, theta = theta) + diag(tau2, n)
+    Ki <- solve(K)
+    ldetK <- determinant(K, logarithm = TRUE)$modulus
+    ll <- - (n / 2) * log(t(Y) %*% Ki %*% Y) - (1 / 2) * ldetK
+    return(-ll)
+  }
```

The gradient is provided by the code below.

```
R> gnl <- function(par, X, Y) {
+    n <- length(Y)
+    theta <- par[1:ncol(X)]; tau2 <- par[ncol(X) + 1]
+    K <- cov_gen(X1 = X, theta = theta) + diag(tau2, n); Ki <- solve(K)
+    KiY <- Ki %*% Y
+    dlltheta <- rep(NA, length(theta))
+    for(k in 1:length(dlltheta)) {
+      dotK <- K * as.matrix(dist(X[, k]))^2 / (theta[k]^2)
+      dlltheta[k] <- (n / 2) * t(KiY) %*% dotK %*% KiY / (t(Y) %*% KiY) -
+        sum(diag(Ki %*% dotK)) / 2
+    }
+    dlltau2 <- (n / 2) * t(KiY) %*% KiY / (t(Y) %*% KiY)
+      - sum(diag(Ki)) / 2
+    return(-c(dlltheta, dlltau2))
+  }
```

Consider a $p = 2$-dimensional input space and responses observed with noise, coded to the unit square. The data-generating function coded below was introduced by (Gramacy and Lee 2009) to illustrate challenges in GP regression when the response surface is nonstationary. (Heteroskedasticity is a form of nonstationarity; and although here the nonstationarity is in the mean, Binois *et al.* (2018a, see Appendix C) showed that nevertheless **hetGP** methods can offer an appropriate quantification of predictive uncertainty on this data.) In order to help separate signal from noise, degree 2 replication is used. Replication is an important focus of this manuscript, with further discussion in Section 2.2. Initial designs are Latin hypercube samples, generated by using the **lhs** package (Carnell 2018).

```
R> library("lhs")
R> X <- randomLHS(40, 2)
R> X <- rbind(X, X)
R> X[, 1] <- (X[, 1] - 0.5) * 6 + 1
R> X[, 2] <- (X[, 2] - 0.5) * 6 + 1
R> y <- X[, 1] * exp(-X[, 1]^2 - X[, 2]^2) + rnorm(nrow(X), sd = 0.01)
```

Estimating hyperparameters is easy with `optim`, plugging in our negative log-likelihood (for minimizing) and gradient. We use the L-BFGS-B method (via `optim` in R), which supports bound constraints. For all parameters, we choose a small (nearly zero) lower bound. For the lengthscales the upper bound is 10, which is much longer than the square of the longest distance in the unit square ($\sqrt{2}$); and for the nugget τ^2 we chose the marginal variance.)

```
R> Lwr <- sqrt(.Machine$double.eps)
R> Upr <- 10
R> out <- optim(c(rep(0.1, 2), 0.1 * var(y)), nl, gnl, method = "L-BFGS-B",
+    lower = Lwr, upper = c(rep(Upr, 2), var(y)), X = X, Y = y)
R> out$par
```

```
[1] 0.734517891 1.520252804 0.009420701
```

Apparently, the lengthscale in x_2 (θ_2) is about $2\times$ longer than for x_1 (θ_1). Interpreting these estimated quantities is challenging and often not a primary focus of inference. What is important is how to map to predictive quantities. Deriving those equations requires rebuilding the covariance structure with estimated hyperparameters, decomposing the covariance structure, and calculating $\hat{\nu}$.

```
R> K <- cov_gen(X, theta = out$par[1:2]) + diag(out$par[3], nrow(X))
R> Ki <- solve(K)
R> nuhat <- drop(t(y) %*% Ki %*% y / nrow(X))
```

Actual prediction requires a set of testing inputs. Below we design a regular grid in two dimensions.

```
R> xx <- seq(-2, 4, length = 40)
R> XX <- as.matrix(expand.grid(xx, xx))
```

The code below extends that covariance structure to the testing inputs, both between themselves and with the training set.

```
R> KXX <- cov_gen(XX, theta = out$par[1:2]) + diag(out$par[3], nrow(XX))
R> KX <- cov_gen(XX, X, theta = out$par[1:2])
R> mup <- KX %*% Ki %*% y
R> Sigmap <- nuhat * (KXX - KX %*% Ki %*% t(KX))
```

Using those quantities, the code below generates the plots shown in Figure 1, illustrating the resulting predictive surface.

```
R> sdp <- sqrt(diag(Sigmap))
R> par(mfrow = c(1,2))
R> cols <- heat.colors(128)
R> persp(xx, xx, matrix(mup, ncol = 40), theta = -30, phi = 30,
+    main = "mean surface", xlab = "x1", ylab = "x2", zlab = "y")
R> image(xx, xx, matrix(sdp, ncol = length(xx)), main = "variance",
+    xlab = "x1", ylab = "x2", col = cols)
R> points(X[, 1], X[, 2])
```

A characteristic feature of GP predictive or kriging surfaces is that the predictive variance is lower at the training data sites and higher elsewhere. This is exemplified by the cooler/redder heat colors near the open circles in the right panel, and hotter/whiter colors elsewhere. In 1D applications the error bars derived from predictive mean and variance surfaces take on a sausage shape, being fat away from the training data sites and thin, or "pinched," at the training locations.

Many libraries, such as those cited in our introduction, offer automations of these procedures. In this manuscript we highlight features of the **hetGP** package, which offers similar calculations as a special case. The code below provides an illustration.

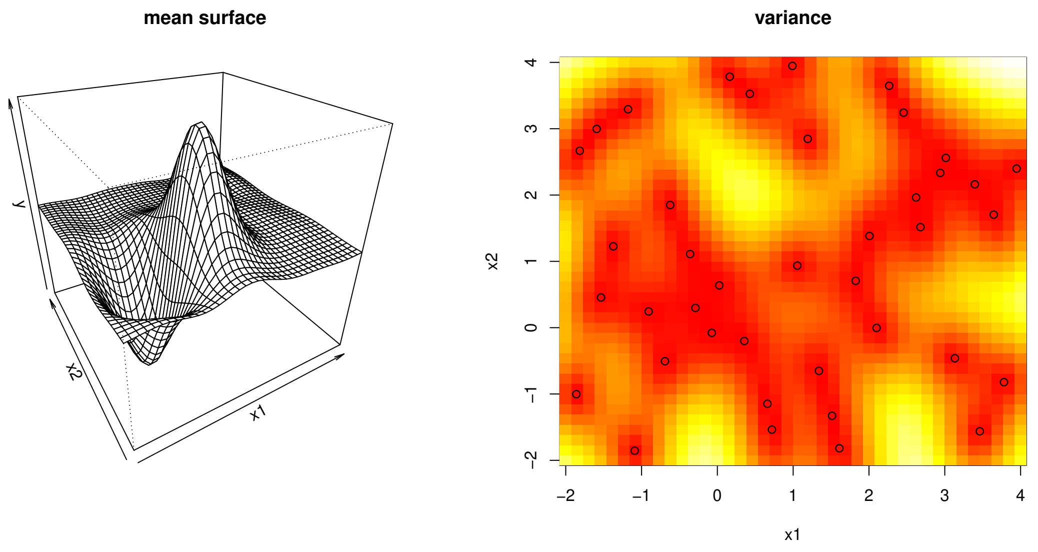

Figure 1: Example predictive surface from a GP. Open circles are the training locations.

```
R> fit <- mleHomGP(X, y, rep(Lwr, 2), rep(Upr, 2), known = list(beta0 = 0),
+    init = c(list(theta = rep(0.1, 2), g = 0.1 * var(y))))
R> c(fit$theta, fit$g)
```

```
[1] 0.723867950 1.570740488 0.008091318
```

These estimated parameters are nearly identical to the ones above, obtained "by hand." There are subtle differences between the objective and optimization being performed, which accounts for the slight differences in higher significant digits. The **hetGP** package offers an S3 `predict` method that can facilitate prediction exactly in the manner illustrated above; but rather than duplicate Figure 1 here, we shall delay an illustration until later.

So, GPs are relatively simple to specify, and inference involves a few dozen lines of code to define an objective (log-likelihood) and its gradient, and a library routine for optimization. Where do the challenges lie? Increasingly, data in geostatistics, machine learning, and computer simulation experiments involves signal-to-noise ratios that may be low and/or possibly changing over the input space. Stochastic (computer) simulators, from physics, business, and epidemiology, may exhibit both of those features simultaneously, but let's start with the first one first. With noisy processes, more samples are needed to isolate the signal. But GPs buckle under the weight of even modestly big data. Evident in the equations above is the need to decompose an $N \times N$ matrix in order to obtain $\mathbf{K}_N^{-1}$ and $|\mathbf{K}_N|$, at $\mathcal{O}(N^3)$ cost. What can be done?

2.2. Speedup from replication

Replication can be a powerful device for separating signal from noise and can yield computational savings as well. If $\bar{\mathbf{Y}} = (\bar{y}_1, \dots, \bar{y}_n)^\top$ collects averages of a_i replicates at $n \ll N$ unique locations $\bar{\mathbf{x}}_i$

$$\bar{y}_i = \frac{1}{a_i}\sum_{j=1}^{a_i} y_i^{(j)} \quad \text{and} \quad \hat{\sigma}_i^2 = \frac{1}{a_i - 1}\sum_{j=1}^{a_i}(y_i^{(j)} - \bar{y}_i)^2,$$

then the "unique-n" predictive equations are a BLUP:

$$\mu_n(\mathbf{x}) = \nu \mathbf{k}_n(\mathbf{x})^\top (\nu \mathbf{C}_n + \mathbf{S}_n)^{-1}\bar{\mathbf{Y}}_n \tag{4}$$

$$\sigma_n^2(\mathbf{x}) = \nu K_\theta(\mathbf{x},\mathbf{x}) - \nu^2 \mathbf{k}_n(\mathbf{x})^\top (\nu \mathbf{C}_n + \mathbf{S}_n)^{-1}\mathbf{k}_n(\mathbf{x}), \tag{5}$$

where $\mathbf{k}_n(\mathbf{x}) = (K_\theta(\mathbf{x},\bar{\mathbf{x}}_1),\ldots,K_\theta(\mathbf{x},\bar{\mathbf{x}}_n))^\top$, $\mathbf{S}_n = [\hat{\sigma}_{1:n}^2]\mathbf{A}_n^{-1} = \mathrm{Diag}(\hat{\sigma}_1^2/a_1,\ldots,\hat{\sigma}_n^2/a_n)$, $\mathbf{C}_n = (K_\theta(\bar{\mathbf{x}}_i,\bar{\mathbf{x}}_j)_{1\le i,j\le n})$, and $a_i \gg 1$. This is the basis of the stochastic kriging predictor (Ankenman *et al.* 2010), which is implemented as an option in the **DiceKriging** and **mleGP** packages on CRAN. The simplicity of this setup is attractive. Basically, an independent moments-based estimate of variance is used in lieu of the more traditional, likelihood-based (hyperparametric) alternative. This could be advantageous if the variance is changing throughout the input space, as we shall discuss further in Section 3. Computationally, the advantage is readily apparent. Only $\mathcal{O}(n^3)$ matrix decompositions are required, which could represent a huge savings compared with $\mathcal{O}(N^3)$ if the degree of replication is high.

However, independent calculations also have their drawbacks, e.g., lacking the ability to specify a priori that variance evolves smoothly over the input space. More fundamentally, the numbers of replicates a_i must be relatively large in order for the $\hat{\sigma}_i^2$ values to be reliable. Ankenman *et al.* (2010) recommend $a_i \ge 10$ for all i, which can be prohibitive. Thinking more homoskedastically, so as not to get too far ahead of our Section 3 discussions, the problem with this setup is that it does not emit a likelihood for inference for the other hyperparameters, such as lengthscale $\boldsymbol{\theta}$ and scale ν. This is because the $\mathbf{S}_n$, $\bar{y}_i$ and a_i values do not constitute a set of sufficient statistics, although they are close to being so.

The fix involves Woodbury linear algebra identities. Although these are not unfamiliar to the spatial modeling community (see, e.g., Opsomer, Ruppert, Wand, Holst, and Hossler 1999; Banerjee *et al.* 2008; Ng and Yin 2012), we believe they have not been used toward precisely this end until recently (Binois *et al.* 2018a). Here, the goal is to make the SK idea simultaneously more general and more prescriptive, facilitating full likelihood-based inference. Specifically, the Woodbury identities are

$$(\mathbf{D} + \mathbf{UBV})^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{U}(\mathbf{B}^{-1} + \mathbf{V}\mathbf{D}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{D}^{-1}$$

$$|\mathbf{D} + \mathbf{UBV}| = |\mathbf{B}^{-1} + \mathbf{V}\mathbf{D}^{-1}\mathbf{U}| \times |\mathbf{B}| \times |\mathbf{D}|.$$

To build $\mathbf{K}_N = \mathbf{U}\mathbf{C}_n\mathbf{V}^\top + \mathbf{D}$, we can take $\mathbf{U} = \mathbf{V}^\top = \mathrm{Diag}(\mathbf{1}_{a_1,1},\ldots,\mathbf{1}_{a_n,1})$ where $\mathbf{1}_{k,l}$ is a $k \times l$ matrix of ones, $\mathbf{D} = \tau^2\mathbb{I}_N$ (or later $\mathbf{D} = \boldsymbol{\Lambda}_N$ in the heteroskedastic setting with a diagonal matrix of variances $\boldsymbol{\Lambda}_N$), and $\mathbf{B} = \mathbf{C}_n$.

Before detailing how the Woodbury maps to prediction (kriging) and likelihood identities for $\mathcal{O}(n^3)$ rather than $\mathcal{O}(N^3)$ calculations, consider how it helps operate on the full covariance structure $\mathbf{K}_N$ via its unique-design counterpart $\mathbf{K}_n := \mathbf{C}_n + \tau^2\mathbf{A}_n^{-1}$. The example below creates a matrix $\mathbf{X}_n$ with $n = 50$ unique (random) rows and then builds $\mathbf{X}_N$ identical to $\mathbf{X}_n$ except that four of its rows have been replicated a number of times, leading to a much bigger $N = 150$ matrix.

```
R> n <- 50
R> Xn <- matrix(runif(2 * n), ncol = 2)
R> Xn <- Xn[order(Xn[, 1]),]
R> ai <- c(8, 27, 38, 45)
R> mult <- rep(1, n); mult[ai] <- c(25, 30, 9, 40)
R> XN <- Xn[rep(1:n, times = mult),]
R> N <- sum(mult)
```

The code below calculates the covariance matrices $\mathbf{K}_n$ and $\mathbf{K}_N$ corresponding to $\mathbf{X}_n$ and $\mathbf{X}_N$, respectively. An arbitrary lengthscale of $\theta = 0.2$ and nugget of $\tau^2 = 0$ is used.

```
R> KN <- cov_gen(XN, theta = 0.2)
R> Kn <- cov_gen(Xn, theta = 0.2)
```

Next, the code below builds the $\mathbf{U}$ matrix for use in the Woodbury formulas.

```
R> U <- c(1, rep(0, n - 1))
R> for(i in 2:n){
+    tmp <- rep(0, n)
+    tmp[i] <- 1
+    U <- rbind(U, matrix(rep(tmp, times = mult[i]), nrow = mult[i],
+      byrow = TRUE))
+ }
R> U <- U[, n:1]
```

Figure 2 shows these three matrices side by side, illustrating the mapping of $\mathbf{K}_n \rightarrow \mathbf{U} \rightarrow \mathbf{K}_N$ with the choices of $\mathbf{B}$ and $\mathbf{U} = \mathbf{V}^\top$ described above. Note that in this case $\mathbf{C}_n = \mathbf{K}_n$ since the nugget is $\tau^2 = 0$.

```
R> layout(matrix(c(1, 2, 3), 1, 3, byrow = TRUE), widths = c(2, 1, 2))
R> par(mar = c(5, 4, 3, 2))
R> image(Kn, x = 1:n, y = 1:n, main = "unique-n: Kn", xlab = "1:n",
+    ylab = "1:n")
R> image(t(U), x = 1:n, y = 1:N, asp = 1, main = "U", xlab = "1:n",
+    ylab = "1:N")
R> image(KN, x = 1:N, y = 1:N, main = "full-N: KN", xlab = "1:N",
+    ylab = "1:N")
```

Not only is storage of $\mathbf{K}_n$ and $\mathbf{U}$, which may be represented sparsely (or even implicitly), more compact than $\mathbf{K}_N$, but the Woodbury formulas show how to calculate the requisite inverse and determinants by acting on $\mathcal{O}(n^2)$ quantities rather than $\mathcal{O}(N^2)$. Pushing that application of the Woodbury identities through to the predictive equations, via the mapping $\nu\mathbf{C}_N + \mathbf{S}_N = \nu(\mathbf{C}_N + \mathbf{\Lambda}_N) \equiv \nu(\mathbf{C}_N + \tau^2\mathbb{I}_N)$ between SK and our more conventional notation, yields the following predictive identities:

$$\nu\mathbf{k}_N(\mathbf{x})^\top(\nu\mathbf{C}_N + \mathbf{S}_N)^{-1}\mathbf{Y} = \mathbf{k}_n(\mathbf{x})^\top(\mathbf{C}_n + \mathbf{\Lambda}_n\mathbf{A}_n^{-1})\bar{\mathbf{Y}}$$
$$\nu K_\theta(\mathbf{x},\mathbf{x}) - \nu^2\mathbf{k}_N(\mathbf{x})^\top(\nu\mathbf{C}_N + \mathbf{S}_N)^{-1}\mathbf{k}_N(\mathbf{x}) = \nu K_\theta(\mathbf{x},\mathbf{x}) - \nu\mathbf{k}_n(\mathbf{x})^\top(\mathbf{C}_n + \mathbf{\Lambda}_n\mathbf{A}_n^{-1})^{-1}\mathbf{k}_n(\mathbf{x}).$$

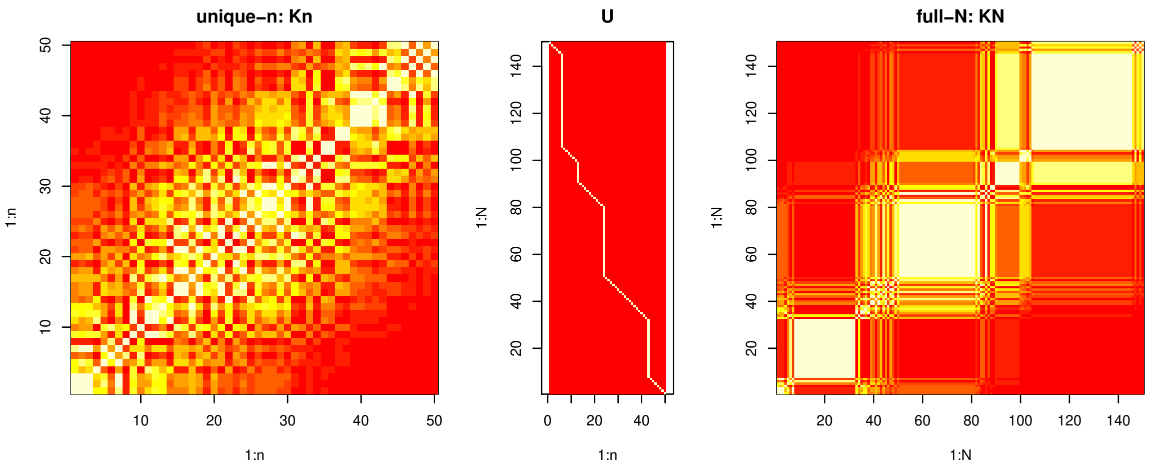

Figure 2: Example mapping of $\mathbf{K}_n \to \mathbf{U} \to \mathbf{K}_N$ through the Woodbury identities.

In words, the typical "full-N" predictive quantities may be calculated identically via "unique-n" counterparts, with potentially dramatic savings in computational time and space. The unique-n predictor is unbiased and minimizes MSPE by virtue of the fact that those properties hold for the full-N one. No asymptotic or frequentist arguments are required. Crucially, no minimum data or numbers of replicates (e.g., $a_i \geq 10$ for SK) are required, although replication can still be helpful from an efficiency perspective [see Section 4.1].

The same trick can be played with the concentrated log-likelihood. Recall that $\mathbf{K}_n = \mathbf{C}_n + \mathbf{A}_n^{-1}\mathbf{\Lambda}_n$; where for now $\mathbf{\Lambda}_n = \tau^2\mathbb{I}_n$. Later we shall generalize $\mathbf{\Lambda}_n$ for the heteroskedastic setting. Using these quantities, we have

$$\ell = c + \frac{N}{2}\log\hat{\nu}_N - \frac{1}{2}\sum_{i=1}^{n}[(a_i - 1)]\log\lambda_i + \log a_i] - \frac{1}{2}\log|\mathbf{K}_n|,$$
$$\text{where } \hat{\nu}_N = N^{-1}(\mathbf{Y}^\top\mathbf{\Lambda}_N^{-1}\mathbf{Y} - \bar{\mathbf{Y}}^\top\mathbf{A}_n\mathbf{\Lambda}_n^{-1}\bar{\mathbf{Y}} + \bar{\mathbf{Y}}^\top\mathbf{K}_n^{-1}\bar{\mathbf{Y}}).$$

Notice the additional terms in $\hat{\nu}_N$ compared with $\hat{\nu}_n := n^{-1}\bar{\mathbf{Y}}^\top\mathbf{K}_n^{-1}\bar{\mathbf{Y}}$. Since $\mathbf{\Lambda}_n$ is diagonal, evaluation of ℓ requires just $\mathcal{O}(n^3)$ operations, which means hyperparameter inference can proceed far more computationally efficiently than in typical setups. The derivative is available in $\mathcal{O}(n^3)$ times, too, facilitating numerical schemes such as L-BFGS-B.

$$\frac{\partial\ell}{\partial\cdot} = \frac{N}{2}\frac{\partial(\mathbf{Y}^\top\mathbf{\Lambda}_N\mathbf{Y} - \bar{\mathbf{Y}}\mathbf{A}_n\mathbf{\Lambda}_n^{-1}\bar{\mathbf{Y}} + n\hat{\nu}_n)}{\partial\cdot} \times (N\hat{\nu}_N)^{-1}$$
$$- \frac{1}{2}\sum_{i=1}^{n}\left[(a_i - 1)\frac{\partial\log\lambda_i}{\partial\cdot}\right] - \frac{1}{2}\mathrm{tr}\left(\mathbf{K}_n^{-1}\frac{\partial\mathbf{K}_n}{\partial\cdot}\right),$$

where "$\cdot$" is a place holder for the hyperparameter(s) of interest.

As an illustration of the potential computational benefit to such a mapping, consider the small example below, with 100 unique input locations, each having a random number of replicates uniformly sampled in $[1, 50]$. The setup is otherwise identical to the 2D example provided in Section 2.1.

```
R> Xbar <- randomLHS(100, 2)
R> Xbar[, 1] <- (Xbar[, 1] - 0.5) * 6 + 1
R> Xbar[, 2] <- (Xbar[, 2] - 0.5) * 6 + 1
R> ytrue <- Xbar[, 1] * exp(-Xbar[, 1]^2 - Xbar[, 2]^2)
R> a <- sample(1:50, 100, replace = TRUE)
R> N <- sum(a)
R> X <- matrix(NA, ncol = 2, nrow = N)
R> y <- rep(NA, N)
R> nf <- 0
R> for(i in 1:100) {
+    X[(nf + 1):(nf + a[i]),] <- matrix(rep(Xbar[i,], a[i]), ncol = 2,
+      byrow = TRUE)
+    y[(nf + 1):(nf + a[i])] <- ytrue[i] + rnorm(a[i], sd = 0.01)
+    nf <- nf + a[i]
+ }
```

The code below invokes `mleHomGP` from the **hetGP** package in two ways. The first bypasses `mleHomGP` preprocessing subroutines that calculate the requisite summary statistics for unique-n modeling, forcing a more cumbersome full-N calculation. The second does things in the more thrifty unique-n way (which is the default).

```
R> fN <- mleHomGP(list(X0 = X, Z0 = y, mult = rep(1, N)), y, rep(Lwr, 2),
+    rep(Upr, 2))
R> un <- mleHomGP(X, y, rep(Lwr, 2), rep(Upr, 2))
```

The code below compares execution times saved above, showing a dramatic difference.

```
R> c(fN = fN$time, un = un$time)

fN.elapsed un.elapsed
    14.233      0.056
```

In this example, using calculations on the unique-n quantities via the Woodbury identities results in execution that is 254 times faster compared with the typical full-N analog. The outcome of these calculations, exemplified below through a reporting of the estimated lengthscales, is nearly identical in both versions.

```
R> rbind(fN = fN$theta, un = un$theta)

       [,1]     [,2]
fN 1.047909 1.796778
un 1.037043 1.814403
```

Besides the SK feature offered by the **mleGP** package, we are not aware of any other software package, for R or otherwise, that automatically preprocesses the data into a form that can exploit these Woodbury identities to speed calculations in the face of heavy replication, or with comparable computational advantage. Replication is a common feature in the sampling of noisy processes, and there is a substantial computational benefit to handling this form of data efficiently.

3. Heteroskedastic modeling

The typical GP formulation, utilizing a covariance structure based on Euclidean distance between inputs, emits a stationary process where the input-output relationship's features are identical throughout the input space. Many data-generating mechanisms are at odds with the assumption of stationarity. A process can diverge from stationarity (i.e., be nonstationary) in various ways, but few are well accommodated by computationally viable modeling methodology. Even fewer come with public software, such as **tgp** (Gramacy 2007; Gramacy and Taddy 2010) and **laGP** (Gramacy 2016). Both of those packages offer a divide-and-conquer approach to accommodate disparate covariance structures throughout the input space. Such a mechanism is computationally thrifty, but it is not without drawbacks such as lack of continuity of the resulting predictive surfaces.

Input-dependent variance, or heteroskedasticity, is a particular form of nonstationarity that is often encountered in practice and that is increasingly encountered in stochastic computer experiment settings. An example is the so-called motorcycle accident data, available in **MASS** (Venables and Ripley 2002). The data arises from a series of simulation experiments measuring the acceleration on the helmet of a motorcycle rider before and after an impact, triggering a whiplash-like effect. Ordinary homoskedastic GPs are notoriously inadequate on this example. Consider the following fit via `mleHomGP`.

```
R> library("MASS")
R> hom <- mleHomGP(mcycle$times, mcycle$accel)
```

The code below comprises of our first demonstration of the generic `predict` method provided by the **hetGP** package, utilizing a dense grid from the smallest to the largest values in the input space (`times`).

```
R> Xgrid <- matrix(seq(0, 60, length = 301), ncol = 1)
R> p <- predict(x = Xgrid, object = hom)
```

Figure 3 shows the resulting predictive surface overlaid on a scatter plot of the data. The solid red line is the predictive mean, and the dashed red lines trace out a 90% predictive interval. The output from `predict` separates variance in terms of mean (`p$sd2`) and residual (nugget-based `p$nugs`) estimates, which we combine in the figure to show the full predictive uncertainty of the response $Y(\mathbf{x}) \mid D_N$.

```
R> plot(mcycle, main = "Predictive Surface")
R> lines(Xgrid, p$mean, col = 2, lwd = 2)
R> lines(Xgrid, qnorm(0.05, p$mean, sqrt(p$sd2 + p$nugs)), col = 2, lty = 2)
R> lines(Xgrid, qnorm(0.95, p$mean, sqrt(p$sd2 + p$nugs)), col = 2, lty = 2)
```

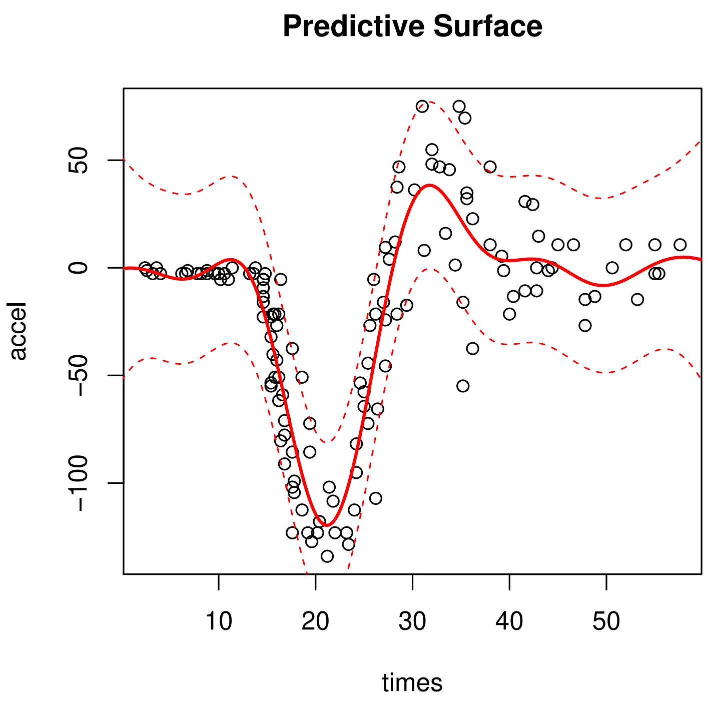

Figure 3: Homoskedastic GP fit to the motorcycle data via mean (solid red) and 95% error bars (dashed red).

Observe in the figure that the fit is less than desirable on two fronts. The most obvious is the overestimated uncertainty across the first third of `times`, from left to right. This corresponds to the preimpact regime in the experiment. Apparently, the predictive surface is overwhelmed by the latter two-thirds of the data, comprising of the higher-variance whiplash regime. Since the model assumes stationarity, a compromise must be made between these two regimes, and the one with stronger support (more data, etc.) wins in this instance. The second is mild mean nonstationarity, whereby the predicted mean in the first third of the data is wiggly when it should be flat. The helmet experiences no measurable acceleration until the impact, so again inferences from the latter two-thirds are bleeding into the first third.

The methodology behind **hetGP** was designed to cope with exactly this sort of behavior, but in a more ambitious setting: larger experiments, in higher dimension. If replication is an important tool in separating signal from noise in homoskedastic settings, it is doubly so in the face of input-dependent noise. Learning how the variance is changing is key to learning about mean dynamics. Therefore, handling and prescribing degrees of replication feature prominently in the methodology, and as a means of remaining computationally thrifty in implementation and execution. Actually, the motorcycle data contains replicates. Only about two-thirds of the inputs are unique in the `mcycle` data. This situation will prove to be helpful in our examples later.

3.1. Joint Gaussian process modeling

SK, introduced in Section 2.2, accommodates a notion of in-sample heteroskedasticity "for free" via independently calculated moments $\mathbf{S}_n = \mathrm{Diag}(\hat{\sigma}_1^2, \dots, \hat{\sigma}_n^2)$, but that is useless out of sample. By fitting each $\hat{\sigma}_i^2$ separately there is no pooling effect for interpolation. Ankenman *et al.* (2010) suggest the quick fix of fitting a second GP to the variance observations with "data,"

$$(\bar{\mathbf{x}}_1, \hat{\sigma}_1^2), (\bar{\mathbf{x}}_2, \hat{\sigma}_2^2), \dots, (\bar{\mathbf{x}}_n, \hat{\sigma}_n^2),$$

to obtain a smoothed variance for use out of sample.

A more satisfying approach from the machine learning literature (Goldberg, Williams, and Bishop 1998) involves introducing latent (log variance) variables under a GP prior and performing joint inference of all unknowns, including hyperparameters for both "mean" and "noise" GPs and latent variances, via MCMC. The overall method, which is effectively on the order of $\mathcal{O}(TN^4)$ for T MCMC samples, is totally impractical but works well on small problems. Several authors have economized on aspects of this framework (Kersting, Plagemann, Pfaff, and Burgard 2007; Lazaro-Gredilla and Titsias 2011) with approximations and simplifications of various sorts, but none (to our knowledge) have resulted in public R software.[1] The key ingredient in these works, of latent variance quantities smoothed by a GP, has merits and can be effective when handled gingerly. The methods implemented in **hetGP** are built around the following methodology.

Let $\delta_1, \delta_2, \dots, \delta_n$ be latent variance variables (or equivalently latent nuggets), each corresponding to one of the n unique design sites $\bar{x}_i$ under study. It is important to introduce latents only for the unique-n locations. A similar approach on the full-N setup, that is, without exploiting the Woodbury identities, is fraught with identifiability and numerical stability challenges. Store these latents diagonally in a matrix $\mathbf{\Delta}_n$, and place them under a GP prior with mean $\mu_{(g)}$,

$$\mathbf{\Delta}_n \sim \mathcal{N}_n(\mu_g, \nu_{(g)}(\mathbf{C}_{(g)} + g\mathbf{A}_n^{-1})),$$

for the purpose of spatial smoothing, with lengthscales $\boldsymbol{\theta}_{(g)}$. Then smooth λ_i values can be calculated by plugging the above $\mathbf{\Delta}_n$ quantities into the mean predictive equations.

$$\mathbf{\Lambda}_n = \mathbf{C}_{(g)}(\mathbf{C}_{(g)} + g\mathbf{A}_n^{-1})^{-1}(\mathbf{\Delta}_n - \mu_{(g)}\mathbb{I}_n) =: \mathbf{C}_{(g)}\mathbf{K}_{(g)}^{-1}(\mathbf{\Delta}_n - \mu_{(g)}\mathbb{I}_n) \tag{6}$$

Smoothly varying $\mathbf{\Lambda}_n$ generalize $\mathbf{\Lambda}_n = \tau^2\mathbb{I}_n$ from our earlier homoskedastic setup, when describing our Woodbury shortcuts under replication in Section 2.2. They also generalize the method of moments estimated $\mathbf{S}_n$ from SK. A nugget parameter g controls the smoothness of λ_i's relative to the δ_i's. A nonzero mean is preferable for this second GP since the predictions tend to revert to this mean far from the observations. Instead of estimating this additional hyperparameter or asking the user for a value, we found it better to use the classical plugin estimator of the mean for a GP, that is, $\hat{\mu}_g = \mathbf{\Delta}_n^\top \mathbf{K}_{(g)}^{-1}\mathbf{\Delta}_n(\mathbf{1}_n^\top \mathbf{K}_{(g)}^{-1}\mathbf{1}_n)^{-1}$.

Variances must be positive, and the equations above give nonzero probability to negative δ_i and λ_i values. One solution is to threshold values at zero. Another is to model $\log(\mathbf{\Delta}_n)$ in this way instead, as originally described by Binois *et al.* (2018a). Differences in the development are slight. Here we favor the simpler, more direct version, in part to offer a complementary presentation to the one in that paper.

[1] A partial implementation is available for Python via **GPy**: `https://sheffieldml.github.io/GPy/`

Rather than Goldberg's MCMC, Binois *et al.* (2018a) describe how to stay within a (Woodbury) MLE framework, by defining a joint log-likelihood over both mean and variance GPs:

$$\tilde{\ell} = c - \frac{N}{2}\log \hat{\nu}_N^2 - \frac{1}{2}\sum_{i=1}^{n}\left[(a_i - 1)\log \lambda_i + \log a_i\right] - \frac{1}{2}\log|\mathbf{K}_n| \tag{7}$$
$$- \frac{n}{2}\log \hat{\nu}_{(g)} - \frac{1}{2}\log|\mathbf{K}_{(g)}|.$$

Maximization may be facilitated via closed-form derivatives with respect to all unknown quantities, all in $\mathcal{O}(n^3)$ time. For example, the derivative with respect to the latent $\mathbf{\Delta}_n$ values follows.

$$\frac{\partial \ell}{\partial \mathbf{\Delta}_n} = \frac{\partial \mathbf{\Lambda}_n}{\partial \mathbf{\Delta}_n}\frac{\partial \log L}{\partial \mathbf{\Lambda}_n} = \mathbf{C}_{(g)}\mathbf{K}_{(g)}^{-1}\frac{\partial \ell}{\partial \mathbf{\Lambda}_n}$$
$$\text{where} \quad \frac{\partial \ell}{\partial \lambda_i} = \frac{N}{2} \times \frac{\frac{a_i s_i^2}{\lambda_i^2} + \frac{(\mathbf{K}_n^{-1}\bar{\mathbf{Y}}_n)_i^2}{a_i}}{\hat{\nu}_N} - \frac{a_i - 1}{2\lambda_i} - \frac{1}{2a_i}(\mathbf{K}_n)_{i,i}^{-1}$$

Binois *et al.* (2018a) show that $\tilde{\ell}$ is maximized when $\mathbf{\Delta}_n = \mathbf{\Lambda}_n$ and $g = 0$. In other words, smoothing the latent noise variables (6) is unnecessary at the MLE. Optimizing the objective naturally smooths the latent $\mathbf{\Delta}_n$ values, leading to GP predictions that interpolate those smoothed values. However, smoothing is useful as a device in three ways: (1) connecting SK to Goldberg's latent representation; (2) annealing to avoid local minima; and (3) yielding a smooth solution when the numerical optimizer is stopped prematurely, which may be essential in big data (large unique-n) contexts.

3.2. Implementation details

Three kernels families are available in **hetGP**, parameterized as follows (univariate form):

- Gaussian: $k_{\mathrm{G}}(x, x') = \exp\left(-\frac{(x-x')^2}{\theta}\right)$;
- Matérn with smoothness parameter 3/2:
$$k_{\mathrm{M32}}(x, x') = \left(1 + \frac{\sqrt{3}|x - x'|}{\theta}\right)\exp\left(-\frac{\sqrt{3}|x - x'|}{\theta}\right);$$
- Matérn with smoothness parameter 5/2:
$$k_{\mathrm{M52}}(x, x') = \left(1 + \frac{\sqrt{5}|x - x'|}{\theta} + \frac{5(x - x')^2}{3\theta^2}\right)\exp\left(-\frac{\sqrt{5}|x - x'|}{\theta}\right).$$

In the **hetGP** package they are referred to respectively as `Gaussian`, `Matern3_2`, and `Matern5_2` and may be specified through the `covtype` argument. Multivariate kernels are defined simply as the product of univariate ones, with one lengthscale per dimension. Isotropic versions, i.e., with the same lengthscale parameter in each dimension, can be obtained by providing scalar values of the bound arguments `upper` and `lower`.

Selecting an appropriate range for lengthscale hyperparameters bounds (with `lower` and `upper`) is difficult: specifications that are too small lead to basically no learning, while overly large values result in oscillations and matrix conditioning issues. Rules of thumb may be based on distances between points or other a priori considerations. To automate the process of choosing these bounds, **hetGP** relies on distances between design points. Specifically, considering a $[0,1]^d$ domain over coded inputs, a lower bound is chosen such that the correlation for a distance between any two points equal to the 5% quantile on distances is at least 1%. Likewise, the upper bound is defined such that the correlation between two points at the 95% quantile distance does not exceed 50%. They are further rescaled if the domain is not $[0,1]^d$. Unless provided, the initial value for g in a homoskedastic model is based on the variance at replicates if there are any; otherwise, it is set to 0.1.

By default, GP and TP models estimate an unknown mean (unless provided by the user via the `known` argument). In the computer experiments literature this setup is known as *ordinary kriging*, as opposed to *simple kriging* with a zero mean. The plugin value of the mean, via MLE calculation, may be derived as $\hat{\mu} = \frac{\mathbf{1}_N^\top \mathbf{K}_N^{-1} \mathbf{Y}_N}{\mathbf{1}_N^\top \mathbf{K}_N^{-1} \mathbf{1}_N} = \frac{\mathbf{1}_n^\top \mathbf{A}_n^{-1} \mathbf{K}_n^{-1} \mathbf{Y}_n}{\mathbf{1}_n^\top \mathbf{K}_n^{-1} \mathbf{1}_n}$. Predictive equations may by modified as follows.

$$\mu_n(\mathbf{x}) = \hat{\mu} + k_n(\mathbf{x})^\top \mathbf{K}_n^{-1}(\mathbf{Y}_n - \hat{\mu}\mathbf{1}_n)$$
$$\sigma^2_{n+1,OK}(\mathbf{x}) = \sigma^2_{n+1,SK}(\mathbf{x}) + \frac{(1 - k_n(\mathbf{x})^\top \mathbf{K}_n^\top \mathbf{Y}_N)^2}{\mathbf{1}_n^\top \mathbf{K}_n^{-1} \mathbf{1}_n}$$

In particular, utilizing $\hat{\mu}$ results in an extra term in the predictive variance.

The heteroskedastic setup entertained in **hetGP** relies on several key points: careful initialization of the latent variables, linking of lengthscale hyperparameters between latent GP and main GP, and safeguards on the variance GP component of the likelihood, namely, the variance smoothness regularization/penalty term. A default initialization procedure, provided in Algorithm 1 and invoked via (`settings$initStrategy = "residual"`), utilizes known or initial values of hyperparameters that can be passed as a list to `known` and `init`, with elements `theta` for the main GP lengthscale, `theta_g` for the latent GP ones, `g_H` the nugget parameter for an homoskedastic GP (i.e., τ^2), `g` the smoothing parameter (that ultimately goes to zero), and the latent variables `Delta`.

To reduce the number of hyperparameters and thus ease maximization of the joint log-likelihood, by default the lengthscale hyperparameters of the latent GP `theta_g` are linked to the ones of the main GP `theta` by a scalar `k_theta_g` in $[1, 100]$. If this assumption that the noise variance is varying more smoothly than the mean is too limiting, as is the case in one example below, the linking between `theta` and `theta_g` can be removed with `settings$linkThetas = "none"`. Another option is to use `settings$linkThetas = "constr"`, to use $\boldsymbol{\theta}$ estimated in step 11 of Algorithm 1 as a lower bound for `theta_g`.

The procedure described so far, with a good initialization, is sufficient for the majority of cases. But there are some pitfalls related to the joint likelihood objective in Equation (7). In fact, this setup may be seen as bi-objective optimization problem, giving a set of compromise solutions between the log-likelihood of the main and latent GPs, with a set of Pareto optimal solutions. However, that perspective would require selecting a solution afterwards, putting a human in the loop. If equal weights between objectives are used, as we do here, a solution on nonconvex parts of the Pareto front may not exist or may be impossible to find. Consequently, the solution returned corresponds to high likelihood for the latent GP and low likelihood for the mean GP. We usually observe this behavior when there is not much heteroskedasticity in the data.

```
Require: $D_N$, plus, optionally, initial values of $\boldsymbol{\theta}$, $\boldsymbol{\theta}_{(g)}$, $g_{\text{Hom}}$, $g$, $\boldsymbol{\Delta}$.
 1: if Initial $\boldsymbol{\theta}$ not provided then
 2:     $\boldsymbol{\theta} = \sqrt{\boldsymbol{\theta}_{\max}\boldsymbol{\theta}_{\min}}$.
 3: end if
 4: if $g_{\text{Hom}}$ is not provided then
 5:     if any $a_i > 5$ then
 6:         $g_{\text{Hom}} \leftarrow \frac{1}{\mathsf{VAR}(\mathbf{Y}_N)} \times \left(\sum_{i=1}^n \delta_{a_i>5}\hat{\sigma}_i^2\right) / \left(\sum_{i=1}^n \delta_{a_i>5}\right)$
 7:     else
 8:         $g_{\text{Hom}} \leftarrow 0.05$
 9:     end if
10: end if
11: Apply mleHomGP on $D_N$ with $\boldsymbol{\theta}$ and $g_{\text{Hom}}$, obtain or update $\boldsymbol{\theta}$, $g_{\text{Hom}}$ and $\hat{\nu}_{\text{Hom}}$.
12: if $\boldsymbol{\Delta}$ not provided then
13:     Compute residuals from homoskedastic fit:
            $$\delta_i = \frac{1}{a_i\hat{\nu}_{\text{Hom}}}\sum_{j=1}^{a_i}\left(\mu(\mathbf{x}_i) - y_i^{(j)}\right)^2 \quad i = 1, \dots, n.$$
14: end if
15: if $\boldsymbol{\theta}_{(g)}$ or $g$ not provided then
16:     Fit homoskedastic GP on $(\mathbf{x}_i, \delta_i)_{1\leq i\leq n}$ with $\boldsymbol{\theta}$, obtain $\boldsymbol{\theta}_g$, $g$.
17: end if
18: Fit heteroskedastic GP on $D_N$ with initial hyperparameters $\boldsymbol{\theta}_{\text{Hom}}$, $\boldsymbol{\theta}_{(g)}$, $g$ and $\boldsymbol{\Delta}$.
```

Algorithm 1: Pseudo code for the default initialization procedure in `mleHetGP`

To circumvent this undesirable behavior, we observe that our goal here is primarily to improve upon a homoskedastic fit of the data. This is enforced in two ways. First, if the likelihood of the main GP (without penalty) is lower than that of its homoskedastic counterpart, the penalty is dropped from the object. From the bi-objective point of view, this amounts to putting a constraint on the mean GP likelihood. Second, at the end of the joint likelihood optimization, if the likelihood of the mean GP with heteroskedastic noise is not better than that of the homoskedastic one, then the homoskedastic model is returned. This latter check can be deactivated with `settings$checkHom = FALSE`.

3.3. Student-t variants

Student-t processes generalize GPs, keeping most of their benefits at almost no extra cost, offering an improved robustness to outliers and larger tail noise. Several choices exist in the literature; see, for example, the work of Wang, Shi, and Lee (2017), and we follow the one described by Shah *et al.* (2014). Briefly, assuming a multivariate-t prior, $\mathbf{Y}_N \sim \mathcal{T}_N(\alpha, 0, \mathbf{K}_N))$ with $\alpha \in (2, \infty]$ being the additional degree of freedom parameter, the modified predictive distribution is multivariate-t,

$$\mathbf{Y}_N|D_N(\mathcal{X}) \sim \mathcal{T}_N\left(\alpha + N, \mu(\mathcal{X}), \frac{\alpha + \beta - 2}{\alpha + N - 2}\boldsymbol{\Sigma}(\mathcal{X})\right) \tag{8}$$

with $\beta = \mathbf{Y}_N^\top \mathbf{K}_N^{-1} \mathbf{Y}_N$.

The corresponding likelihood has a closed form:

$$\log(L) = -\frac{N}{2}\log((\alpha-2)\pi) - \frac{1}{2}\log(|\mathbf{K}_N|) + \log\left(\frac{\Gamma\left(\frac{\alpha+N}{2}\right)}{\Gamma\left(\frac{\alpha}{2}\right)}\right) - \frac{(\alpha+N)}{2}\log\left(1+\frac{\beta}{\alpha-2}\right),$$

and so does its derivatives.

As shown by Chung *et al.* (2018), the Woodbury simplification and heteroskedastic extension can be carried out just as for GPs. We employ a different parameterization of $\mathbf{K}_N = \sigma^2\mathbf{C}_N + \tau^2\mathbb{I}_N$ in this case, because the plugin value of $\hat{\nu}$ does not apply here. But this parameterization of the covariance enables enables an SK variant of this model (similar to that of Xie and Chen (2017)), by setting `log = FALSE` in `settings` and giving the empirical variance estimates in `Delta` with `known` (or `init` for initialization) in `mleHetTP`.

We note that while TPs are more flexible that GPs, as N goes to infinity they become equivalent. In addition, the estimation of the parameter α can become unreliable (see, e.g., Wang *et al.* 2017), such that prespecifying it at a low value may be beneficial.

3.4. Illustrations

Here we consider several examples in turn, returning first to the motorcycle data and then introducing three challenging real-world computer model simulation examples.

Motorcycle data: The code below shows how to fit a heteroskedastic (coupled) GP via a smoothed initialization and Matérn covariance structure. This is but one of several potential ways to obtain a good fit to this data by using the methods provided by the **hetGP** package.

```
R> het2 <- mleHetGP(mcycle$times, mcycle$accel, covtype = "Matern5_2",
+    lower = 15, upper = 50, settings = list(initStrategy = 'smoothed'))
R> het2$time

elapsed
  0.464
```

The time it takes to perform the calculations is trivial, although this is not a big problem. The built-in `predict` method can perform the calculations required in order to get predictions on our grid from earlier.

```
R> p2 <- predict(het2, Xgrid)
R> ql <- qnorm(0.05, p2$mean, sqrt(p2$sd2 + p2$nugs))
R> qu <- qnorm(0.95, p2$mean, sqrt(p2$sd2 + p2$nugs))
```

Figure 4, which is generated by the code below, shows the resulting predictive surface in two views.

```
R> par(mfrow = c(1, 2))
R> plot(mcycle$times, mcycle$accel, ylim = c(-160, 90), ylab = "acc",
+    xlab = "time", main = "Predictive Surface")
```

```
R> lines(Xgrid, p2$mean, col = 2, lwd = 2)
R> lines(Xgrid, ql, col = 2, lty = 2)
R> lines(Xgrid, qu, col = 2, lty = 2)
R> plot(Xgrid, p2$nugs, type = "l", lwd = 2, ylab = "s2", xlab = "time",
+    main = "Variance Surface", ylim = c(0, 2e3))
R> points(het2$X0, sapply(find_reps(mcycle[, 1], mcycle[, 2])$Zlist, var),
+    col = 3, pch = 20)
```

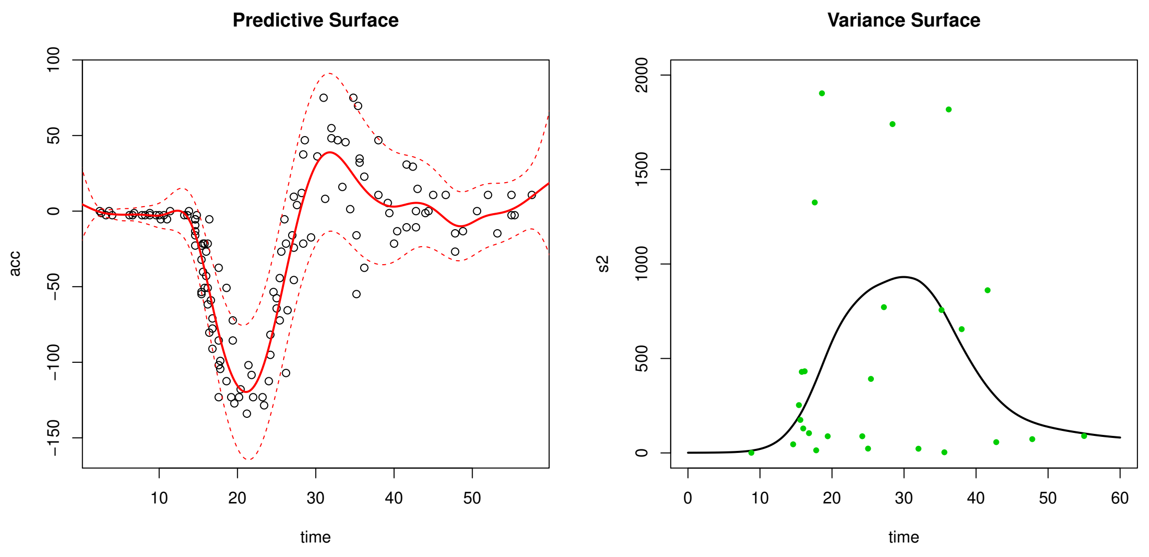

Figure 4: Heteroskedastic GP fit to the motorcycle data. Left panel shows the predictive distribution via mean (solid red) and 90% error bars (dashed red). Right panel shows the estimated variance surface and moment-based estimates of variance.

The first view, in the left panel, complements Figure 3. Observe how the surface is heteroskedastic, learning the low-variance region in the first third of the data and higher variance for the later two-thirds. As a consequence of being able to better track the signal-to-noise relationship over the input space, the estimate of the signal (particularly for the first third of `times`) is better. The second view, in the right panel, provides more detail on the estimated variance surface. Predictive uncertainty is highest for the middle third of the `times`, which makes sense because this is where the whiplash effect is most prominent. The green dots in that panel indicate moment-based estimates of variance obtained from the limited number of replicates in this example. (There are nowhere near enough for an SK-like approach.) Notice how the black curve, smoothing the latent variance values $\mathbf{\Delta}_n$, extracts the essence of the pattern of those values. This variance fit uses the full data set, leveraging the smoothness of the GP prior to leverage data with only one replicate, which in this case represents most of the data.

Susceptible, infected, recovered model: Binois *et al.* (2018a) consider a 2D problem arising from a simulation of the spread of an epidemic in a susceptible, infected, recovered (SIR) setting, as described by Hu and Ludkovski (2017). A function generating the data for standardized inputs in the unit square (corresponding to S and I) is provided by `sirEval` in the **hetGP** package. Consider a space-filling design of size $n = 200$ unique runs, with a random number of replicates $a_i \in \{1, \dots, 100\}$, for $i = 1, \dots, n$. The x_1 coordinate represents the initial number of infecteds I_0, and the x_2 coordinate the initial number of susceptibles S_0.

```
R> Xbar <- randomLHS(200, 2)
R> a <- sample(1:100, nrow(Xbar), replace = TRUE)
R> X <- matrix(NA, ncol = 2, nrow = sum(a))
R> nf <- 0
R> for(i in 1:nrow(Xbar)) {
+    X[(nf + 1):(nf + a[i]),] <- matrix(rep(Xbar[i,], a[i]), ncol = 2,
+      byrow = TRUE)
+    nf <- nf + a[i]
+ }
R> nf

[1] 10634
```

The result is a full data set with $N = 10634$ runs. The code below gathers our response, which is the expected number of infecteds at the end of the simulation.

```
R> Y <- apply(X, 1, sirEval)
```

The code below fits our `hetGP` model. By default, lengthscales for the variance GP are linked to those from the mean GP, requiring that the former be a scalar multiple $k > 1$ of the latter. That can be undone, however, as we do below primarily for illustrative purposes.

```
R> fit <- mleHetGP(X, Y, lower = rep(0.05, 2), upper = rep(10, 2),
+    settings = list(linkThetas = "none"), covtype = "Matern5_2", maxit = 1e4)
R> fit$time

elapsed
  1.562
```

Around 1.6 seconds are needed to train the model. To visualize the resulting predictive surface, the code below creates a dense grid in 2D and calls the `predict` method on the `"hetGP"`-class object.

```
R> xx <- seq(0, 1, length = 100)
R> XX <- as.matrix(expand.grid(xx, xx))
R> p <- predict(fit, XX)
R> psd <- sqrt(p$sd2 + p$nugs)
```

Figure 5 shows the resulting predictive surface, as obtained via the code below.

```
R> par(mfrow = c(1, 2))
R> cols <- terrain.colors(128)
R> image(xx, xx, matrix(p$mean, 100), xlab = "S0", ylab = "I0", col = cols,
+    main = "Mean Infected")
R> text(Xbar, labels = a, cex = 0.75)
R> image(xx, xx, matrix(psd, 100), xlab = "S0", ylab = "I0", col = cols,
+    main = "SD Infected")
R> text(Xbar, labels = a, cex = 0.75)
```

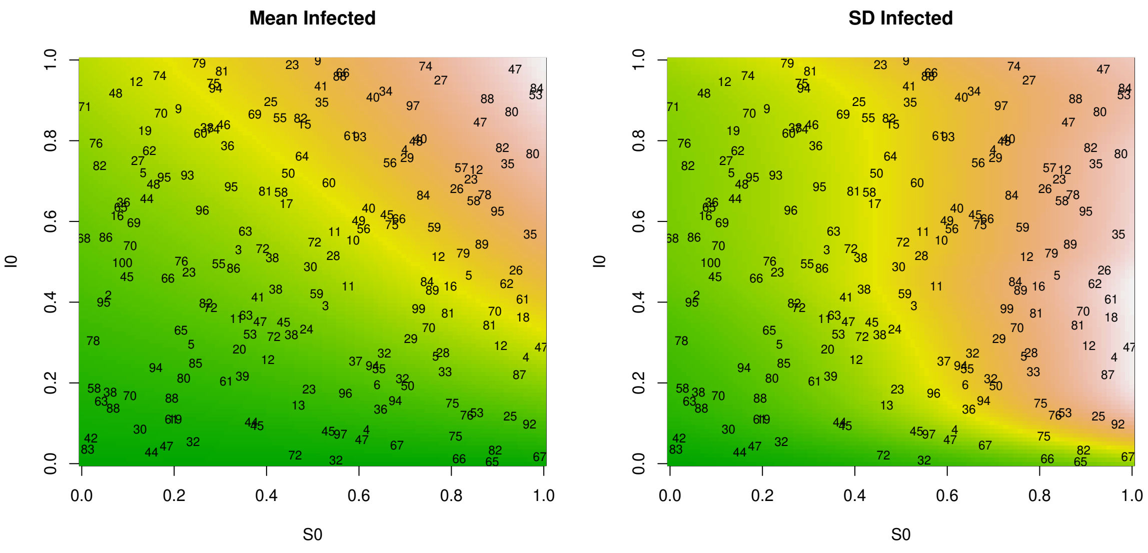

Figure 5: Heteroskedastic GP fit to the SIR data. Left panel shows the predictive mean surface; right panel shows the estimated standard deviation. Text in both panels shows numbers of replicates.

The left panel in the figure shows the predictive mean surface. The right panel shows the predicted standard deviation. Text overlaid on the panels indicates the location of the training data inputs and the number of replicates observed thereon.

Bayes factor surfaces: Bayes factor (BF) calculation for model selection is known to be sensitive to hyperparameter settings, which is further complicated (and obscured) by Monte Carlo evaluation that interjects a substantial source of noise. To study BF surfaces in such settings, Franck and Gramacy (2018) propose treating expensive BF calculations, via MCMC say, as a (stochastic) computer simulation experiment. The idea is that BF calculations at a space-filling design in the (input) hyperparameter space can be used to map out the space and better understand the sensitivity of model selection to those settings.

As a simple warmup example, consider an experiment described by Gramacy and Pantaleo (2010, Section 3.3–3.4) involving BF calculations to determine whether data is leptokurtic (Student-t errors) or not (simply Gaussian) as a function of the prior parameterization on the Student-t degrees of freedom parameter, which they took to be $\nu \sim \text{Exp}(\theta = 0.1)$. Their intention was to be diffuse, but ultimately they lacked an appropriate framework for studying sensitivity to this choice. Franck and Gramacy (2018) created a grid of hyperparameter values in θ, evenly spaced in $\log_{10}$ space from 10^{-3} to 10^6 spanning "solidly Student-t" (even Cauchy) to "essentially Gaussian" in terms of the mean of the prior over ν. For each θ_i on the grid they ran the RJ-MCMC to approximate $\mathrm{BF}_{\mathrm{St}\mathcal{N}}$ by feeding in sample likelihood evaluations provided by **monomvn**'s (Gramacy 2017) `blasso` to compute a BF, following a postprocessing scheme described by Jacquier, Polson, and Rossi (2004). In order to understand the Monte Carlo variability in those calculations, ten replicates of the BFs under each hyperparameter setting were collected. Each $\mathrm{BF}_{\mathrm{St}\mathcal{N}}$ evaluation, utilizing $T = 100000$ MCMC samples, takes about 36 minutes to obtain on a 4.2 GHz Intel Core i7 processor, leading to a total runtime of about 120 hours to collect all 200 values saved. The data is provided with the **hetGP** package.

```
R> data("bfs")
R> thetas <- matrix(bfs.exp$theta, ncol = 1)
R> bfs <- as.matrix(t(bfs.exp[, -1]))
```

For reasons that will become clear momentarily, Franck and Gramacy (2018) fit a heteroskedastic Student-t process, described briefly in Section 3.3. Even though they fit in log-log space, the process is still heteroskedastic and has heavy tails.

```
R> bfs1 <- mleHetTP(X = list(X0 = log10(thetas), Z0 = colMeans(log(bfs)),
+    mult = rep(nrow(bfs), ncol(bfs))), Z = log(as.numeric(bfs)),
+    lower = 10^(-4), upper = 5, covtype = "Matern5_2")
```

Predictive evaluations were then extracted on a grid in the input space.

```
R> dx <- seq(0, 1, length = 100)
R> dx <- 10^(dx * 4 - 3)
R> p <- predict(bfs1, matrix(log10(dx), ncol = 1))
```

The results are shown in Figure 6, via plotting commands shown below. In the figure, each open circle is a $\mathrm{BF}_{\mathrm{St}\mathcal{N}}$ evaluation, plotted in $\log_{10} - \log_e$ space.

```
R> matplot(log10(thetas), t(log(bfs)), col = 1, pch = 21, ylab = "log(bf)",
+    main = "Bayes factor surface")
R> lines(log10(dx), p$mean, lwd = 2, col = 2)
R> lines(log10(dx), p$mean + 2 * sqrt(p$sd2 + p$nugs), col = 2, lty = 2,
+    lwd = 2)
R> lines(log10(dx), p$mean - 2 * sqrt(p$sd2 + p$nugs), col = 2, lty = 2,
+    lwd = 2)
R> legend("topleft", c("hetTP mean", "hetTP interval"), col = 2, lty = 1:2,
+    lwd = 2)
```

Clearly the $\mathrm{BF}_{\mathrm{St}\mathcal{N}}$ surface is heteroskedastic, even after log transform, and has heavy tails. The take-home message from these plots is that the BF surface is extremely sensitive to hyperparameterization. When θ is small, the Student-t (BF below 1) is essentially a foregone conclusion, whereas if θ is large, the Gaussian (BF above 1) is. A seemingly innocuous hyperparameter setting is essentially determining the outcome of a model selection enterprise.

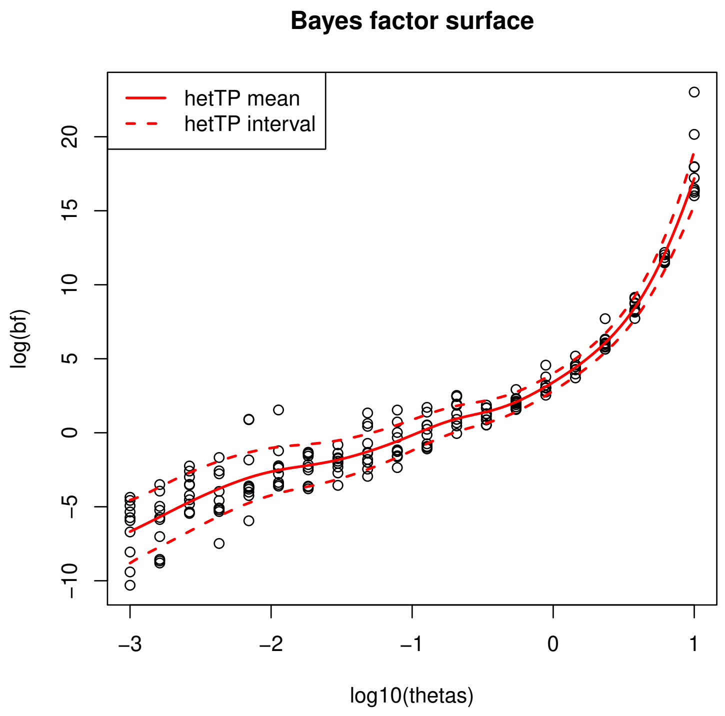

Figure 6: Heteroskedastic TP fit to the Bayes factor data under exponential hyperprior.

Although the computational burden involved in this experiment, 120 hours, is tolerable, extending the idea to higher dimensions is problematic. Suppose one wished to entertain $\nu \sim \text{Gamma}(\alpha, \beta)$, where the $\alpha = 1$ case reduces to $\nu \sim \text{Exp}(\beta \equiv \theta)$ above. Over a similarly dense hyperparameter grid, the runtime would balloon to 100 days, which is clearly unreasonable. Instead it makes sense to build a surrogate model from a more limited space-filling design and use the resulting posterior predictive surface to understand variability in BFs in the hyperparameter space. Responses on a space-filling design in $\alpha \times \beta$-space, via a Latin hypercube sample of size 80, using a recently updated version of the **monomvn** library to accommodate the Gamma prior, with replicates were obtained at each input setting, for a total of 400 runs. The data is also provided by the `bfs` data object in **hetGP**.

```
R> D <- as.matrix(bfs.gamma[, 1:2])
R> bfs <- as.matrix(t(bfs.gamma[, -(1:2)]))
```

A similar `hetTP` fit can be obtained with the following command.

```
R> bfs2 <- mleHetTP(X=list(X0 = log10(D), Z0 = colMeans(log(bfs)),
+    mult = rep(nrow(bfs), ncol(bfs))), Z = log(as.numeric(bfs)),
+    lower = rep(10^(-4), 2), upper = rep(5, 2), covtype = "Matern5_2")
```

Prediction on a dense grid in the 2D input space can be extracted as follows.

```
R> dx <- seq(0, 1, length = 100)
R> dx <- 10^(dx * 4 - 3)
R> DD <- as.matrix(expand.grid(dx, dx))
R> p <- predict(bfs2, log10(DD))
```

Figure 7 shows the outcome of that experiment, via the plotting commands below. In that figure, the mean surface is shown on the left and standard deviation surface on the right. The numbers overlaid on the figure are the average $\mathrm{BF}_{\mathrm{St}\mathcal{N}}$ obtained for the five replicates at each input location.

```
R> par(mfrow = c(1, 2))
R> mbfs <- colMeans(bfs)
R> image(log10(dx), log10(dx), t(matrix(p$mean, ncol=length(dx))),
+    col = heat.colors(128), xlab = "log10 alpha", ylab = "log10 beta",
+    main = "mean log BF")
R> text(log10(D[, 2]), log10(D[, 1]), signif(log(mbfs), 2), cex = 0.5)
R> contour(log10(dx), log10(dx), t(matrix(p$mean, ncol = length(dx))),
+    levels = c(-5, -3, -1, 0, 1, 3, 5), add = TRUE, col = 4)
R> image(log10(dx), log10(dx), t(matrix(sqrt(p$sd2 + p$nugs),
+    ncol = length(dx))), col = heat.colors(128), xlab = "log10 alpha",
+    ylab = "log10 beta", main = "sd log BF")
R> text(log10(D[, 2]), log10(D[, 1]), signif(apply(log(bfs), 2, sd), 2),
+    cex = 0.5)
```

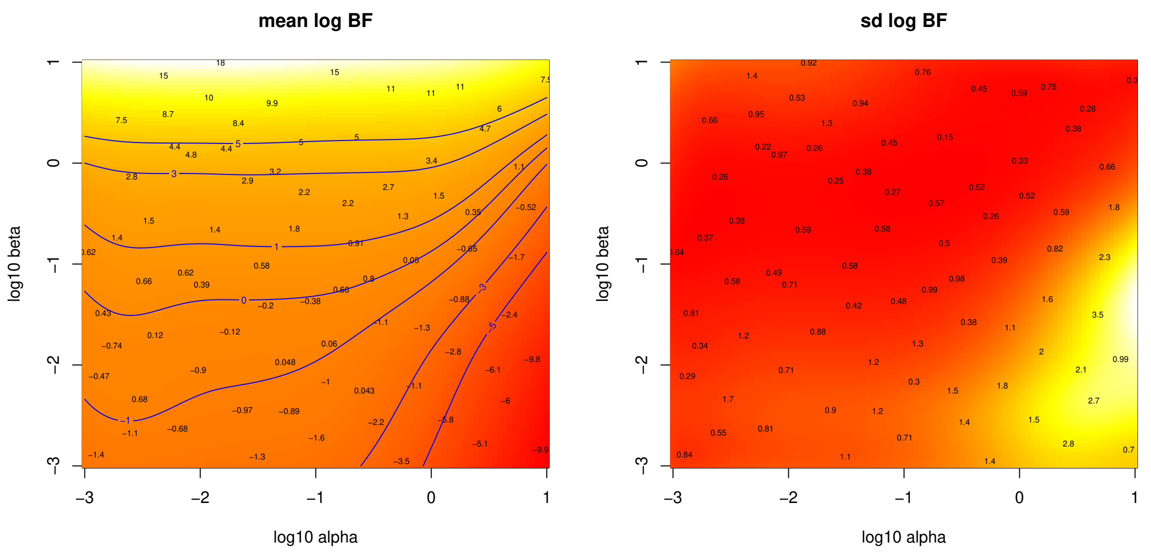

Figure 7: Heteroskedastic TP fit to the Bayes factor data under Gamma hyperprior.

The story here is much the same as before in terms of β, which maps to θ in the earlier experiment, especially near $\alpha = 1$ (i.e., $\log_{10} \alpha = 0$) where the equivalence is exact. The left panel shows that along that slice one can get just about whatever conclusion one wants.

Smaller α values tell a somewhat more nuanced story, however. A rather large range of smaller α values leads to somewhat less sensitivity in the outcome because of the β hyperparameter setting, except when β is quite large. Apparently, having a small α setting is essential if the data is going to have any influence on model selection via BF. The right panel shows that the variance surface is indeed changing over the input space, justifying the heteroskedastic surrogate.

Another example with `hetTP` is provided by Chung *et al.* (2018), who augment with a latent variable scheme to account for missing data and to enforce a monotonicity property for solving a challenging class of inverse problems motivated by an influenza example.

Assemble to order: The assemble-to-order (ATO) problem (Hong and Nelson 2006) involves a queuing simulation targeting inventory management scenarios. The setup is as follows. A company manufactures m products. Products are built from base parts called items, some of which are "key" in that the product cannot be built without them. If a random request comes in for a product that is missing a key item, a replenishment order is executed, which is filled after a random delay. Holding items in inventory is expensive, so there is a balance between inventory costs and revenue. Hong and Nelson built a MATLAB simulator for this setup, which was reimplemented by Xie, Frazier, and Chick (2012). Binois *et al.* (2018a) describe an out-of-sample experiment based on this latter implementation in its default setting (originally from Hong and Nelson), specifying item cost structure, product makeup (their items) and revenue, and distribution of demand and replenishment time, under target stock vector inputs $b \in \{0, 1, \dots, 20\}^8$ for eight items.

Here we provide code that can be used to replicate results from that experiment, which involved a uniform design of size $n_{\text{tot}} = 2000$ in 8D space, with ten replicates for a total design size of $N_{\text{tot}} = 20000$. The R code below loads in that data, which is stored in the data object `ato`, provided with **hetGP**.

```
R> data("ato")
```

In order to create an out-of-sample testing setting, random training–testing partitions were constructed by randomly selecting $n = 1000$ unique locations and a uniform number of replicates $a_i \in \{1, \dots, 10\}$ thereupon. The `ato` data object also contains one such random partition, which is subsequently coded into the unit cube $[0, 1]^8$. Further detail is provided in the package documentation for the `ato` object. Actually the object provides two testing sets. One is a genuine out-of-sample testing set, where the testing sites do not intersect with any of the training sets. The other is replicate based, involving replicates $10 - a_i$ not selected for training. The training set is large, which makes MLE calculations slow, so the `ato` object provides a fitted model for comparison. In the examples section of the `ato` documentation, code is provided to reproduce that fit or to create a new one based on a new random training–testing partition.

```
R> c(n = nrow(Xtrain), N = length(unlist(Ztrain)), time = out$time)

           n            N time.elapsed
    1000.000     5594.000     8583.767
```

The main reason for storing these objects is to enable a fast illustration of prediction and out-of-sample comparison, in particular to have something to compare with a more thoughtful sequential design scheme outlined in Section 4. The code below performs predictions at the held-out testing locations and then calculates a proper score (Gneiting and Raftery 2007, Equation (27)) against the ten replicates observed at each of those locations. Higher scores are better.

```
R> phet <- predict(out, Xtest)
R> phets2 <- phet$sd2 + phet$nugs
R> mhet <- as.numeric(t(matrix(rep(phet$mean, 10), ncol = 10)))
R> s2het <- as.numeric(t(matrix(rep(phets2, 10), ncol = 10)))
R> sehet <- (unlist(t(Ztest)) - mhet)^2
R> sc <- - sehet/s2het - log(s2het)
R> mean(sc)

[1] 3.393781
```

A similar calculation can be made for the held-out training replicates, shown below. These are technically out of sample, but accuracy is higher since training data was provided at these locations.

```
R> phet.out <- predict(out, Xtrain.out)
R> phets2.out <- phet.out$sd2 + phet.out$nugs
R> s2het.out <- mhet.out <- Ztrain.out
R> for(i in 1:length(mhet.out)) {
+    mhet.out[[i]] <- rep(phet.out$mean[i], length(mhet.out[[i]]))
+    s2het.out[[i]] <- rep(phets2.out[i], length(s2het.out[[i]]))
+  }
R> mhet.out <- unlist(t(mhet.out))
R> s2het.out <- unlist(t(s2het.out))
R> sehet.out <- (unlist(t(Ztrain.out)) - mhet.out)^2
R> sc.out <- - sehet.out/s2het.out - log(s2het.out)
R> mean(sc.out)

[1] 5.089897
```

The two testing sets can be combined to provide a single score calculated on the entire corpus of held-out data. As expected, the result is a compromise between the two score statistics calculated earlier.

```
R> mean(c(sc, sc.out))

[1] 3.925302
```

Binois *et al.* (2018a) repeat that training-testing partition 100 times to produce score boxplots that are then compared with simpler (e.g., homoskedastic) GP-based approaches. We refer the reader to Figure 2 in that paper for more details. To make a long story short, fits accommodating heteroskedasticity in the proper way—via coupled GPs and fully likelihood-based inference—are superior to all other (computationally tractable) ways entertained.

4. Sequential design

We have been using uniform or space-filling designs in our examples above, and with replication. The thinking is that coverage in the input space is sensible and that replication will help separate signal from noise (and yield computational advantages). Model-based designs, such as maximum entropy and related criteria, are almost never appropriate in this setting. Such designs, since they condition on the model, are hyperparameter sensitive; and before data is collected, we have little to go on to choose good settings for those values. This situation is exacerbated in the heteroskedastic setting, requiring the setting of latent inputs and additional hyperparameters. A far better approach is to take things one step at a time: start with a small space-filling design (small N), perhaps with some replication, and choose the next point, $\mathbf{x}_{N+1}$, by exploring its impact on the model fit, say through the predictive equations. An interesting question, in such settings, is how much such criteria would prefer to replicate (repeat existing inputs) versus explore (try new locations).

In the classical GP setup for computer simulations, with low or no noise and an assumption of stationarity (i.e., constant stochasticity), one can argue that replication is of little or no value. When signal-to-noise ratios are low and/or changing over the input space, however, intuition suggests that some replication will be valuable. Until recently, however, it was not known how such choices would manifest in typical sequential design or data acquisition decisions. Of course, the details depend on the goal of modeling and prediction. Below we consider two settings: (1) obtaining accurate predictions globally [Section 4.1] and (2) optimizing or targeting particular aspects of the predictive distribution such as level sets or contours (Section 4.2).

4.1. IMSPE

Binois *et al.* (2018b) studied the exploration vs. replication question in the context of improving predictive accuracy by means of sequential design. A common criterion in that setting is the *integrated mean-square prediction error* (IMSPE), which is just predictive variance averaged over the input space, specifically,

$$I_{n+1} \equiv \mathrm{IMSPE}(\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_n, \mathbf{x}_{n+1}) = \int_{\mathbf{x} \in D} \sigma^2_{n+1}(\mathbf{x})\, d\mathbf{x}. \tag{9}$$

This formula is written in terms of unique-n inputs for reasons that we shall clarify shortly. It could, however, instead be phrased in terms of full-N equations. The idea would be to choose the next design location, x_{N+1}, which could be a new unique location ($\bar{\mathbf{x}}_{n+1}$) or a repeat of an existing one (i.e., one of $\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_n$), by maximizing I_{n+1}. The presentation in Binois *et al.* is more careful, but also more cumbersome, with the notation in this regard.

In general, the integral in Equation (9) usually requires numerical evaluation (Seo *et al.* 2000; Gramacy and Lee 2009; Gauthier and Pronzato 2014; Gorodetsky and Marzouk 2016; Pratola, Harari, Bingham, and Flowers 2017b). For example, the **tgp** package (option `Ds2x=TRUE`) sums over a reference set, as well as averaging over the posterior distribution of (treed) Gaussian process parameterization. However, conditional on GP hyperparameters, and when the study region D is an easily integrable domain such as a hyperrectangle, the requisite integration may be calculated in closed form. Although examples exist in the literature (e.g., Ankenman *et al.* 2010; Anagnostopoulos and Gramacy 2013; Burnaev and Panov 2015; Leatherman, Santner, and Dean 2017) for particular cases (and approximations), we are not aware of examples providing the level of specificity and generality in derivation, or implementation in code as provided in **hetGP**.

The essence is as follows.

$$\begin{aligned} I_{n+1} = \mathsf{E}\{\sigma^2_{n+1}(X)\} &= \mathsf{E}\{K_\theta(X,X) - \mathbf{k}_{n+1}(X)^\top \mathbf{K}^{-1}_{n+1}\mathbf{k}_{n+1}(X)\} \\ &= \mathsf{E}\{K_\theta(X,X)\} - \mathrm{tr}(\mathbf{K}^{-1}_{n+1}\mathbf{W}_{n+1}), \end{aligned} \tag{10}$$

where $W_{ij} = \int_{\mathbf{x}\in D} k(\mathbf{x}_i,\mathbf{x})k(\mathbf{x}_j,\mathbf{x})\, d\mathbf{x}$. Closed forms for the W_{ij} exist with D being a hyperrectangle, say. Binois *et al.* (2018b) provide forms for several popular covariance functions, including the Matérn. In the case of the Gaussian, we quote as follows:

$$W_{ij} = \prod_{k=1}^{d} \frac{\sqrt{2\pi\theta_k}}{4} \exp\left\{-\frac{(x_{i,k}-x_{j,k})^2}{2\theta_k}\right\} \left[\mathrm{erf}\left\{\frac{2-(x_{i,k}+x_{j,k})}{\sqrt{2\theta_k}}\right\} + \mathrm{erf}\left\{\frac{x_{i,k}+x_{j,k}}{\sqrt{2\theta_k}}\right\}\right].$$

Having a closed form is handy for evaluation. Even better is that a closed form exists for the gradient, which facilitates numerical optimization for sequential design. We leave the details of that calculation to Binois *et al.*

To investigate how replication can be favored by IMSPE in choosing $\mathbf{x}_{n+1}$, consider the following setup. Let $r(\mathbf{x}) = \mathsf{VAR}[Y(\mathbf{x}) \mid f(\mathbf{x})]$ denote a belief about the (otherwise zero mean and iid) noise process. The form of $r(\mathbf{x})$ can be arbitrary. Below we set up two univariate examples that follow splines that agree at five "knots." In this illustration, the x-locations of those knots could represent design locations x_i where responses have been observed.

```
R> rn <- c(4.5, 5.5, 6.5, 6, 3.5)
R> X0 <- matrix(seq(0.05, 0.95, length.out = length(rn)))
R> X1 <- matrix(c(X0, 0.2, 0.4))
R> Y1 <- c(rn, 5.2, 6.3)
R> r1 <- splinefun(x = X1, y = Y1, method = "natural")
R> X2 <- matrix(c(X0, 0.0, 0.3))
R> Y2 <- c(rn, 7, 4)
R> r2 <- splinefun(x = X2, y = Y2, method = "natural")
```

Figure 8 provides a visual of these two variance surface hypotheses, evaluated over the predictive grid provided below.

```
R> XX <- matrix(seq(0, 1, by = 0.005))
```

Below we shall refer to these surfaces as "green" and "blue," respectively, referencing the colors from Figure 8. The "knots" are shown as red open circles.

```
R> plot(X0, rn, xlab = "x", ylab = "r(x)", xlim = c(0, 1), ylim = c(2, 8),
+    col = 2, main = "Two Variance Hypotheses")
R> lines(XX, r1(XX), col = 3)
R> lines(XX, r2(XX), col = 4)
```

The code below implements the closed form IMSPE of Equation (10) for a generic variance function `r`, like one of our splines from above. The implementation uses some internal `hetGP` functions such as `Wij` and `cov_gen`. (We shall illustrate the intended hooks momentarily; these low-level functions are of value here in this toy illustration.)

```
R> IMSPE.r <- function(x, X0, theta, r) {
+    x <- matrix(x, nrow = 1)
+    Wijs <- Wij(mu1 = rbind(X0, x), theta = theta, type = "Gaussian")
+    K <- cov_gen(X1 = rbind(X0, x), theta = theta)
+    K <- K + diag(apply(rbind(X0, x), 1, r))
+    return(1 - sum(solve(K) * Wijs))
+ }
```

The next step is to apply this function on a grid for each of the two choices for $r(\mathbf{x})$, green and blue.

```
R> imspe1 <- apply(XX, 1, IMSPE.r, X0 = X0, theta = 0.25, r = r1)
R> imspe2 <- apply(XX, 1, IMSPE.r, X0 = X0, theta = 0.25, r = r2)
R> xstar1 <- which.min(imspe1)
R> xstar2 <- which.min(imspe2)
```

Figure 8 shows these two surfaces along with the minimizing values. The x-locations of the knots, our design sights $\mathbf{x}_1, \ldots, \mathbf{x}_n$, are shown as dashed red vertical bars.

```
R> plot(XX, imspe1, type = "l", col = 3, ylab = "IMSPE", xlab = "x",
+    ylim = c(0.6, 0.7), main = "IMSPE for two variances")
R> lines(XX, imspe2, col = 4)
R> abline(v = X0, lty = 3, col = 'red')
R> points(XX[xstar1], imspe1[xstar1], pch = 23, bg = 3)
R> points(XX[xstar2], imspe2[xstar2], pch = 23, bg = 4)
```

In the figure the blue variance function hypothesis is minimized at a novel $\mathbf{x}_{n+1}$ location, not coinciding with any of the previous design sites. However, the green hypothesis is minimized at $\mathbf{x}_2$, reading from left to right. The IMSPE calculated on this particular variance function $r(\mathbf{x})$ prefers replication. That is a coincidence or fabrication. Binois *et al.* showed that the next point $\mathbf{x}_{N+1}$ will be a replicate, that is, be one of the existing unique locations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ rather than a new $\mathbf{x}_{n+1}$ when

$$\begin{aligned} r(\mathbf{x}_{N+1}) \geq & \frac{\mathbf{k}_n(\mathbf{x}_{N+1})^\top \mathbf{K}_n^{-1} \mathbf{W}_n \mathbf{K}_n^{-1} \mathbf{k}_n(\mathbf{x}_{N+1}) - 2\mathbf{w}_{n+1}^\top \mathbf{K}_n^{-1} \mathbf{k}_n(\mathbf{x}_{N+1}) + w_{n+1,n+1}}{\mathrm{tr}(\mathbf{B}_{k^*} \mathbf{W}_n)} \\ & - \sigma_n^2(\mathbf{x}_{N+1}), \end{aligned}$$

where $k^* = \mathrm{argmin}_{k \in \{1,\ldots,n\}} \mathrm{IMSPE}(\mathbf{x}_k)$ and $\mathbf{B}_k = \frac{(\mathbf{K}_n^{-1})_{.,k} (\mathbf{K}_n^{-1})_{k,.}}{\frac{\nu \lambda_k}{a_k(a_k+1)} - (\mathbf{K}_n)^{-1}_{k,k}}$.
Basically, this relationship says that IMSPE will prefer replication when the variance is "large enough." Rather than read tea leaves more deeply than that, let us see it in action in our toy example. The code below utilizes some of **hetGP**'s internals to enable evaluation of the right-hand side of the inequality above, in other words, treating it as an equality.

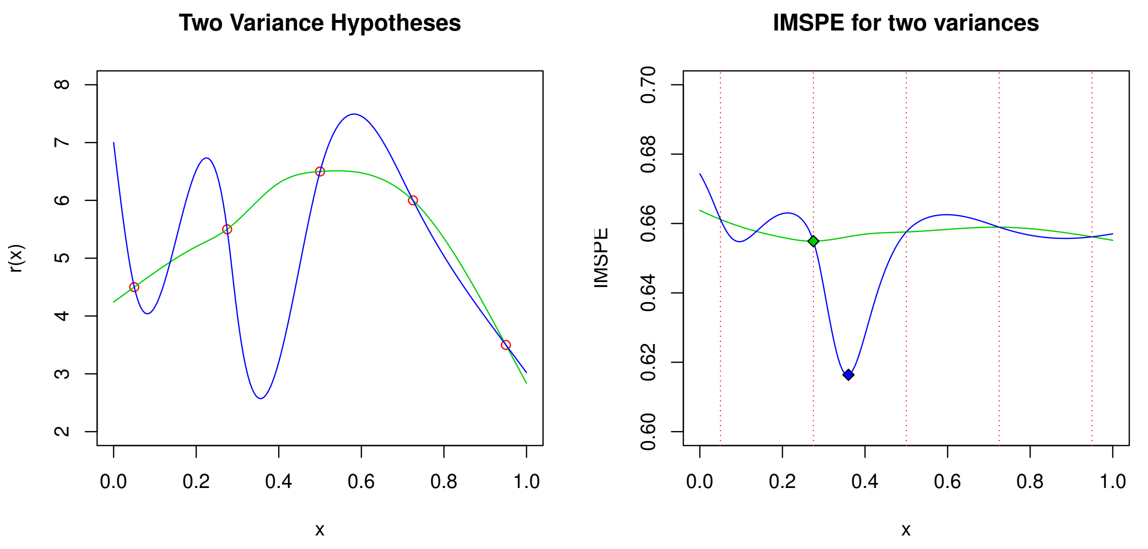

Figure 8: Two hypothetical variance functions (left) and the resulting IMSPE surfaces (right).

```
R> rx <- function(x, X0, rn, theta, Ki, kstar, Wijs) {
+    x <- matrix(x, nrow = 1); kn1 <- cov_gen(x, X0, theta = theta)
+    wn <- Wij(mu1 = x, mu2 = X0, theta = theta, type = "Gaussian")
+    a <- kn1 %*% Ki %*% Wijs %*% Ki %*% t(kn1) - 2*wn %*% Ki %*% t(kn1)
+    a <- a + Wij(mu1 = x, theta = theta, type = "Gaussian")
+    Bk <- tcrossprod(Ki[, kstar], Ki[kstar,]) /
+      (2 / rn[kstar] - Ki[kstar, kstar])
+    b <- sum(Bk * Wijs); sn <- 1 - kn1 %*% Ki %*% t(kn1)
+    return(a / b - sn)
+ }
```

Calling that function with the `XX` grid defined above, with covariance structure details prescribed (more or less) arbitrarily, commences as follows.

```
R> bestk <- which.min(apply(X0, 1, IMSPE.r, X0 = X0, theta = 0.25, r = r1))
R> Wijs <- Wij(X0, theta = 0.25, type = "Gaussian")
R> Ki <- solve(cov_gen(X0, theta = 0.25, type = "Gaussian") + diag(rn))
R> rx.thresh <- apply(XX, 1, rx, X0 = X0, rn = rn, theta = 0.25, Ki = Ki,
+    kstar = bestk, Wijs = Wijs)
```

Augmenting our original IMSPE plots from Figure 8, the code below adds a gray line indicating that threshold, and in particular showing that the green hypothesis is everywhere above that threshold in this instance.

```
R> plot(X0, rn, xlab = "x", ylab = "r(x)", xlim = c(0, 1), ylim = c(2, 8),
+    lty = 2, col = 2, main = "Which variance is large enough?")
R> lines(XX, r1(XX), col = 3); lines(XX, r2(XX), col = 4)
R> lines(XX, rx.thresh, lty = 2, col = "darkgrey")
```

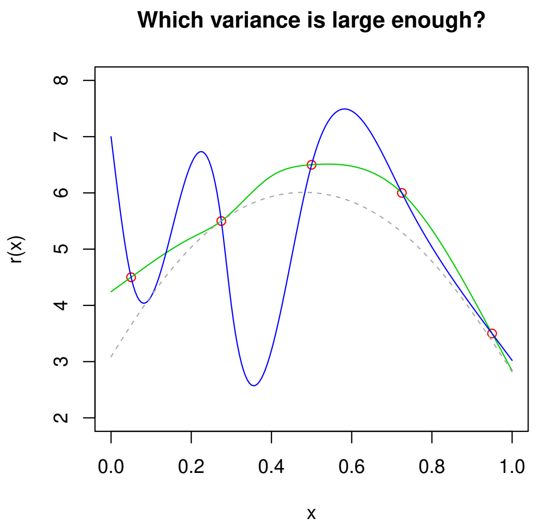

Figure 9: IMSPE calculations for two variance hypotheses, with replicating threshold added in gray.

That green hypothesis is, of course, just one instance of a variance function that is above the requisite threshold for replication. Also, the hypotheses we used were not derived from GP predictive equations. But the example is designed to illustrate potential. Before turning to a more prescriptive search for new sites, be they replicates or new unique locations, let us summarize what we know. Replication (1) is good for the GP calculations ($n^3 \ll N^3$); (2) is preferred by IMSPE under certain (changing) variance regimes; and (3) helps separate signal from noise. But how often is IMSPE going to "ask" for replications? The short answer is, not often enough, at least from an empirical standpoint.

One challenge is numerical precision in optimization when mixing discrete and continuous search. Given a continuum of potential new locations, up to floating-point precision, a particular setting corresponding to a finite number of replicate sites is not likely to be preferred over all other settings, such as ones infinitesimally nearby (no matter what the theory prefers). Another issue, which is more fundamental, is that IMSPE is myopic. The value the current selection, be it a replicate or new unique location, is not assessed *vis á vis* its impact on the future decision landscape. In general, entertaining such decision spaces is all but impossible, although that does not stop people from trying. See Section 4.2 for a related discussion in an optimization context.

Replication-biased lookahead

Binois *et al.* (2018b) found that a "replication-biased" lookahead is manageable. Specifically, consider a horizon h into the future. Looking ahead over those choices, one can select a new unique $\bar{\mathbf{x}}_{n+1}$ and h replicates, each at one of the $n+1$ unique locations. Or alternatively, one can take a replicate first and a unique design element later (and there are h ways to do that). A longer horizon means a greater chance of replication but also more calculation: $h+1$ continuous searches for new locations and $(h+1)(h+2)/2-1$ discrete ones in search of potential replicates. The horizon h can thus be thought of as a tuning parameter. Before considering choices for how to set this value, let us look at an example for fixed horizon, $h = 3$.

Take $f(x) = (6x-2)^2 \sin(12x-4)$ from Forrester, Sobester, and Keane (2008), which is implemented as `f1d` in **hetGP**, and observe $Y(x) \sim \mathcal{N}(f(x), r(x))$, where the noise variance function is $r(x) = (1.1 + \sin(2\pi x))^2$.

```
R> fn <- function(x) { (1.1 + sin(x * 2 * pi)) }
R> fr <- function(x) { f1d(x) + rnorm(length(x), sd = fn(x)) }
```

Consider an initial uniform design of size $N = n = 10$ (i.e., without replicates) and an initial **hetGP** fit based on a Gaussian kernel.

```
R> X <- seq(0, 1, length = 10)
R> Y <- fr(X)
R> mod <- mleHetGP(X = X, Z = Y, lower = 0.0001, upper = 10)
```

Next, let us search via IMSPE with horizon $h = 5$ lookahead over replication. The `IMSPE_optim` call below utilizes the closed form IMSPE (10) and its derivatives within an `optim` call using `method="L-BFGS-B"`.

```
R> opt <- IMSPE_optim(mod, h = 5)
R> c(X, opt$par)
```

```
 [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556
 [7] 0.6666667 0.7777778 0.8888889 1.0000000 0.4444444
```

Whether or not the chosen location, in position eleven above (0.444), is a replicate depends on the random seed used to compile this document, so it is difficult to provide precise commentary in-line here. The **hetGP** package provides an efficient updating method, utilizing $\mathcal{O}(n^2)$ or $\mathcal{O}(n)$ updating calculations for new data points, depending on whether that point is unique or a replicate, respectively. Details of those updates are provided by Binois *et al.* (2018b), extending existing ones in the literature (e.g., Gramacy and Polson 2011; Chevalier, Ginsbourger, and Emery 2014a; Gramacy and Apley 2015) to the heteroskedastic case.

```
R> X <- c(X, opt$par)
R> Ynew <- fr(opt$par)
R> Y <- c(Y, Ynew)
R> mod <- update(mod, Xnew = opt$par, Znew = Ynew, ginit = mod$g * 1.01)
```

That is the basic idea. Let us continue and gather a total of 500 samples in this way, in order to explore the aggregate nature of the sequential design so constructed. Periodically (every 25 iterations in the code below), it can be beneficial to restart the MLE calculations to help "unstick" any solutions found in local modes of the likelihood surface. Gathering 500 points is somewhat of an overkill for this simple 1D problem, but it helps create a nice visualization.

```
R> for(i in 1:489) {
+    opt <- IMSPE_optim(mod, h = 5)
+    X <- c(X, opt$par)
+    Ynew <- fr(opt$par)
+    Y <- c(Y, Ynew)
+    mod <- update(mod, Xnew = opt$par, Znew = Ynew, ginit = mod$g * 1.01)
+    if(i %% 25 == 0){
+      mod2 <- mleHetGP(X = list(X0 = mod$X0, Z0 = mod$Z0, mult = mod$mult),
+      Z = mod$Z, lower = 0.0001, upper = 1)
+      if(mod2$ll > mod$ll) mod <- mod2
+    }
+ }
R> nrow(mod$X0)

[1] 54
```

Of the total of $N = 500$, the final design contained $n = 54$ unique locations. To help visualize and assess the quality of the final surface with that design, the code below gathers predictive quantities on a dense grid in the input space.

```
R> xgrid <- seq(0, 1, length = 1000)
R> p <- predict(mod, matrix(xgrid, ncol = 1))
R> pvar <- p$sd2 + p$nugs
```

Figure 10, via the code below, shows the resulting predictive surface in red, with additional calculations to show the true surface, via $f(x)$ and error-bars from $r(x)$, in black. Gray bars help visualize the degree of replication at each input location.

```
R> plot(xgrid, f1d(xgrid), type = "l", xlab = "x", ylab = "y",
+    main="Forrester Example, IMSPE h=5", ylim = c(-8, 18))
R> lines(xgrid, qnorm(0.05, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> lines(xgrid, qnorm(0.95, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> points(X, Y)
R> segments(mod$X0, rep(0, nrow(mod$X0)) - 8, mod$X0, (mod$mult - 8) * 0.5,
+    col = "gray")
R> lines(xgrid, p$mean, col = 2)
R> lines(xgrid, qnorm(0.05, p$mean, sqrt(pvar)), col = 2, lty = 2)
R> lines(xgrid, qnorm(0.95, p$mean, sqrt(pvar)), col = 2, lty = 2)
R> legend("top", c("truth", "estimate"), col = 1:2, lty = 1:2)
```

Observe that the degree of replication, as well as the density of unique design locations, is higher in the high-noise region than it is in the low-noise region. In a batch design setting and in the unique situation where relative noise levels were known, a rule of thumb of more samples or replicates in the higher noise regime is sensible. The trouble is that such regimes are rarely known in advance, and neither are the optimal density differentials and degrees of replication. Proceeding sequentially allows us to learn and adapt the design as we go.

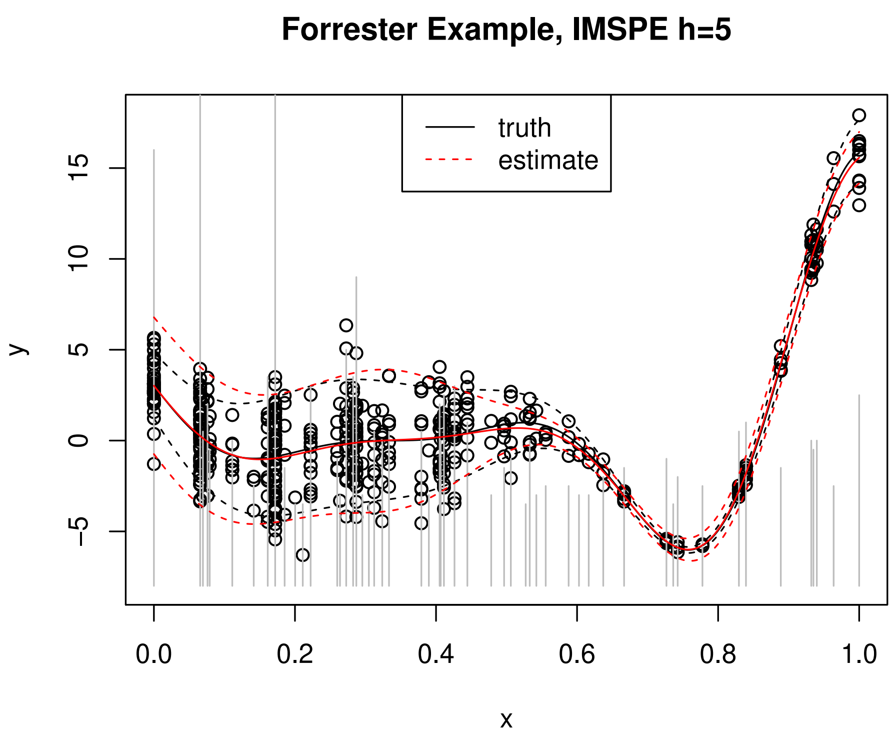

Figure 10: Sequential design with horizon $h = 5$. The truth is in black and the predictive distribution in red.

Tuning the horizon

Previously we used horizon $h = 5$, but that was rather arbitrary. Although it is difficult to speculate on details regarding the quality of the surface obtained above, improvements are likely possible. Chances are that the uncertainty is overestimated in some regions and underestimated in others. A solution lies in adapting the lookahead horizon online. Binois *et al.* (2018b) proposed two simple on-line adjustments that tune the horizon. The first adjusts the horizon of the next IMSPE search in order to *target* a desired ratio $\rho = n/N$ and thus manage the surrogate modeling cost:

$$\text{Target:} \quad h_{N+1} \leftarrow \begin{cases} h_N + 1 & \text{if } n/N > \rho \quad \text{and a new } \bar{\mathbf{x}}_{n+1} \text{ is chosen} \\ \max\{h_N - 1, -1\} & \text{if } n/N < \rho \quad \text{and a replicate is chosen} \\ h_N & \text{otherwise.} \end{cases}$$

The second attempts to *adapt* to minimize IMSPE regardless of computational cost:

$$\text{Adapt:} \quad h_{N+1} \sim \text{Unif}\{a'_1, \ldots, a'_n\} \quad \text{with} \quad a'_i := \max(0, a^*_i - a_i),$$

and $a^*_i \propto \sqrt{r(\bar{x}_i)(\mathbf{K}_n^{-1}\mathbf{W}_n\mathbf{K}_n^{-1})_{i,i}}$ comes from a criterion in the SK literature (Ankenman *et al.* 2010).

The code below duplicates the example above with the *adapt* heuristic. Alternatively, `horizon` call can be provided `target` and `previous_ratio` arguments to implement the *target* heuristic instead. First, reinitialize the design.

```
R> X <- seq(0, 1, length = 10)
R> Y <- fr(X)
R> mod.a <- mleHetGP(X = X, Z = Y, lower = 0.0001, upper = 10)
R> h <- rep(NA, 500)
```

Next, loop to obtain $N = 500$ observations under the adaptive horizon scheme.

```
R> for(i in 1:490) {
+    h[i] <- horizon(mod.a)
+
+    opt <- IMSPE_optim(mod.a, h = h[i])
+    X <- c(X, opt$par)
+    Ynew <- fr(opt$par)
+    Y <- c(Y, Ynew)
+    mod.a <- update(mod.a, Xnew = opt$par, Znew = Ynew,
+      ginit = mod.a$g * 1.01)
+
+    if(i %% 25 == 0){
+      mod2 <- mleHetGP(X = list(X0 = mod.a$X0, Z0 = mod.a$Z0,
+        mult = mod.a$mult), Z = mod.a$Z, lower = 0.0001, upper = 1)
+      if(mod2$ll > mod.a$ll) mod.a <- mod2
+    }
+  }
```

Then, obtain predictions on the grid.

```
R> p.a <- predict(mod.a, matrix(xgrid, ncol = 1))
R> pvar.a <- p.a$sd2 + p.a$nugs
```

The code below provides the calculations behind the visualization in Figure 11. The left panel of that figure shows the adaptively selected horizon over the iterations of sequential design. The right panel shows the final design and predictions, versus the truth, matching Figure 10 for the fixed horizon ($h = 5$) case.

```
R> par(mfrow = c(1, 2))
R> plot(h, main = "Horizon", xlab = "Iteration")
R> plot(xgrid, f1d(xgrid), type = "l", xlab = "x", ylab = "y",
+    main = "Adaptive Horizon Design", ylim = c(-8, 18))
R> lines(xgrid, qnorm(0.05, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> lines(xgrid, qnorm(0.95, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> points(X, Y)
R> segments(mod$X0, rep(0, nrow(mod$X0)) - 8, mod$X0, (mod$mult - 8) * 0.5,
+    col = "gray")
R> lines(xgrid, p$mean, col = 2)
R> lines(xgrid, qnorm(0.05, p$mean, sqrt(pvar.a)), col = 2, lty = 2)
R> lines(xgrid, qnorm(0.95, p$mean, sqrt(pvar.a)), col = 2, lty = 2)
```

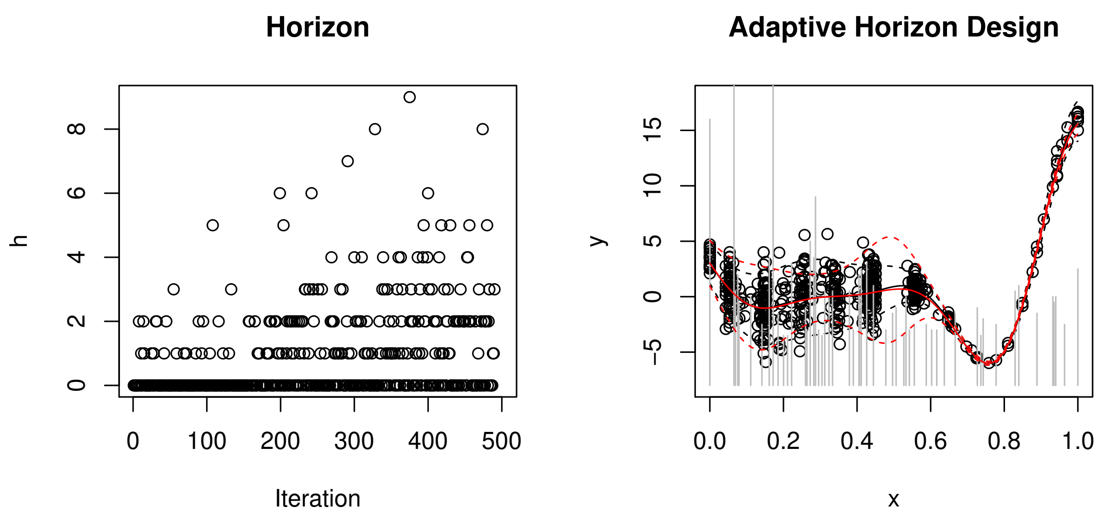

Figure 11: *Left:* Horizons chosen per iteration; *right:* final design and predictions versus the truth, similar to Figure 10

The left panel of the figure reveals that a horizon of $h = 5$ is indeed not totally uncommon, but it is also higher than generally preferred by the adaptive scheme, which had $n = 109$ unique sites—more than the $h = 5$ case, but in the same ballpark compared with the full size of $N = 500$.

```
R> nrow(mod.a$X0)
```

```
[1] 109
```

The code below offers an out-of-sample comparison via RMSE (lower is better) to the truth and score (higher is better) against a noisy analog.

```
R> ytrue <- f1d(xgrid)
R> yy <- fr(xgrid)
R> rbind(rmse = c(h5 = mean((ytrue - p$mean)^2),
+    ha = mean((ytrue - p.a$mean)^2)),
+    score = c(h5 = - mean((yy - p$mean)^2 / (pvar) + log(pvar)),
+    ha = -mean((yy - p.a$mean)^2 / pvar.a + log(pvar.a))))
```

```
              h5          ha
rmse   0.0220554  0.04002842
score -0.8789815 -1.26984744
```

Although speculating on the precise outcome of this comparison is difficult because of the noisy nature of sampling and horizon updates, the typical outcome is that the two comparators are similar on RMSE, which is quite small, but that score prefers the adaptive scheme.

Being able to adjust the horizon enables the adaptive scheme to better balance exploration versus replication in order to efficiently learn the signal-to-noise ratio in the data-generating mechanism.

A larger example

For a larger example, let us revisit the ATO application from Section 3.1. Binois *et al.* (2018b) described a sequential design scheme utilizing fixed, adaptive, and target horizon schemes. The `ato` data object loaded earlier contained a `"hetGP"`-class model called `out.a` that was trained with an adaptive horizon IMSPE-based sequential design scheme. The size of that design and the time it took to train are quoted by the output of the R code below.

```
R> c(n = nrow(out.a$X0), N = length(out.a$Z), time = out.a$time)
```

```
           n            N time.elapsed
     1194.00      2000.00     38737.97
```

Recall that the earlier experiment involved $n = 1000$ unique sites with an average of five replicates at each, for a total of about $N = 5000$ training data points. The training set here is much smaller, having $N = 2000$ at $n = 1194$ unique locations. Thus, the adaptive design has more unique locations but still a nontrivial degree of replication, resulting in many fewer overall runs of the ATO simulator. Utilizing the same out-of-sample testing set from the previous score-based comparison, the code below calculates predictions and scores with this new sequential design.

```
R> phet.a <- predict(out.a, Xtest)
R> phets2.a <- phet.a$sd2 + phet.a$nugs
R> mhet.a <- as.numeric(t(matrix(rep(phet.a$mean, 10), ncol = 10)))
R> s2het.a <- as.numeric(t(matrix(rep(phets2.a, 10), ncol = 10)))
R> sehet.a <- (unlist(t(Ztest)) - mhet.a)^2
R> sc.a <- - sehet.a/s2het.a - log(s2het.a)
R> c(batch = mean(sc), adaptive = mean(sc.a))
```

```
   batch adaptive
3.393781 3.616211
```

The sequential design leads to more accurate predictors than the batch design does, despite having being trained on 40% as many runs. To illustrate how that design was built, we first need to "rebuild" the `out.a` object. For compact storage, the covariance matrices, inverses, and so forth have been deleted via `return.matrices=FALSE` in the `mleHetGP` command.

```
R> out.a <- rebuild(out.a)
```

The calculation sequence for each step of the search for this design involves first determining the horizon and then searching with that horizon via IMSPE. In code, that amounts to the following.

```
R> Wijs <- Wij(out.a$X0, theta = out.a$theta, type = out.a$covtype)
R> h <- horizon(out.a, Wijs = Wijs)
R> control <- list(tol_dist = 1e-4, tol_diff = 1e-4, multi.start = 30)
R> opt <- IMSPE_optim(out.a, h, Wijs = Wijs, control = control)
```

Precalculating the `Wijs` saves a little time since these are needed for both `horizon` and `IMSPE_optim`.

```
R> opt$par
```

```
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.8947368 0.6842105 0.8947368 0.8421053 0.6315789 0.2105263
          [,7]      [,8]
[1,] 0.7894737 0.6842105
```

The 8D location above would then be fed into the computer model simulator and the input output pair subsequently added into the design. An indication of whether or not the new location is unique (i.e., actually new) or a replicate is provided with `path` along with the `IMSPE_optim` output, `par`, and `value`. The `path` list contains the best sequence of points found by the lookahead procedure, with elements `par`, `value`, and `new` of the next design plus h further ones selected successively.

```
R> opt$path[[1]]$new
```

```
[1] FALSE
```

Since ATO inputs are actually on a grid, we would have to first snap this continuous solution to that grid (after undoing the coding of the inputs). In so doing we could create a replicate as part of that discretization process.

4.2. Contour finding and optimization

So far we have focused on constructing a globally accurate surrogate model, but it is also common to target specific regions of interest. Examples include global minima or level sets (Bogunovic, Scarlett, Krause, and Cevher 2016).

Bayesian optimization: Starting with optimization, namely, finding $\mathbf{x}^* \in \text{argmin}_{\mathbf{x}} Y(\mathbf{x})$, we follow the canonical efficient global optimization framework (Jones, Schonlau, and Welch 1998) with the expected improvement (EI) criterion (Mockus, Tiesis, and Zilinskas 1978). For a review of many variants of this and other so-called Bayesian optimization (BO) methods, see, for example, Shahriari, Swersky, Wang, Adams, and de Freitas (2016) and Frazier (2018). Picheny, Wagner, and Ginsbourger (2012) provide benchmarks specifically for the noisy objective case. In that setting, where more evaluations are needed to separate signal from noise, EI is appealing since it does not need extra tuning parameters to balance exploitation and exploration, does not require numerical approximation, and has a closed-form derivative.

For a deterministic function, the improvement $I_N : \mathbf{x} \in \mathbb{R}^d \mapsto \mathbb{R}$ is defined as

$$I(\mathbf{x}) = \max\left[\min_{i\in\{1,\dots,n\}} Y(\mathbf{x}_i) - Y(\mathbf{x})\right],$$

a random variable. EI is the expectation of the improvement conditioned on the observations, which has a closed-form expression:

$$\mathrm{EI} \equiv \mathsf{E}(I_N(\mathbf{x}) \mid D_N) = (y^* - \mu_n(\mathbf{x}))\Phi\left(\frac{y^* - \mu_n(\mathbf{x})}{\sigma_n(\mathbf{x})}\right) + \sigma_n(\mathbf{x})\phi\left(\frac{y^* - \mu_n(\mathbf{x})}{\sigma_n(\mathbf{x})}\right),$$

where $y^* = \min_{i\in\{1,\dots,n\}} Y(\mathbf{x}_i)$ and ϕ and Φ are the pdf and cdf of the standard normal distribution, respectively. In the noisy case, y^* defined in this way is no longer a viable option, but it can be replaced for instance with $\min_{i\in\{1,\dots,n\}} \mu_n(\mathbf{x}_i)$ as recommended by Vazquez, Villemonteix, Sidorkiewicz, and Walter (2008) or $\min_{\mathbf{x}} \mu(\mathbf{x})$ as described by Gramacy and Lee (2011).

Lookahead versions of EI have been studied (see, e.g., Ginsbourger and Le Riche 2010; Gonzalez, Osborne, and Lawrence 2016; Lam, Willcox, and Wolpert 2016; Huan and Marzouk 2016), but a closed-form expression exists only for one-step-ahead versions. The reason is that, unlike IMPSE, future values of the criterion depend on future function values. Inspired by (Lam *et al.* 2016) and our IMSPE-analog (Section 4.1), we introduce here a replication-biased lookahead for EI. To circumvent the unknown future function values issue, our simple implementation uses $y_{n+1} \leftarrow \mu_n(\mathbf{x}_{n+1})$, a kriging "believer" type of approach (Ginsbourger, Le Riche, and Carraro 2010).

Going back to the 1D example, the code below implements this replication-biased lookahead scheme in an optimization context. Notice that we initialize with a small amount of replication. This is to guard against initial (and incorrect) noiseless surrogate fits that cause numerical issues. Initial designs for Bayesian optimization of noisy functions is still an open area of research. Also, we fix a lookahead horizon of $h = 5$; developing an analog *target* and *adapt* scheme is left for future research as well.

First, we reinitialize the design and fit, but this time with a small degree of replication to stabilize the behavior of this example across `knitr` builds.

```
R> X <- seq(0, 1, length = 10)
R> X <- c(X, X)
R> Y <- fr(X)
R> mod <- mleHetGP(X = X, Z = Y, lower = 0.0001, upper = 10)
```

We have 500 sequential design iterations as before, but this time with EI.

```
R> for(i in 1:480) {
+    opt <- crit_optim(mod, crit = "crit_EI", h = 5)
+    X <- c(X, opt$par)
+    Ynew <- fr(opt$par)
+    Y <- c(Y, Ynew)
+    mod <- update(mod, Xnew = opt$par, Znew = Ynew, ginit = mod$g * 1.01)
+
+    if(i %% 25 == 0){
+      mod2 <- mleHetGP(X = list(X0 = mod$X0, Z0 = mod$Z0, mult = mod$mult),
+        Z = mod$Z, lower = 0.0001, upper = 1)
+      if(mod2$ll > mod$ll) mod <- mod2
+    }
+  }
```

Next, we obtain predictions on the grid.

```
R> p <- predict(mod, matrix(xgrid, ncol = 1))
R> pvar <- p$sd2 + p$nugs
```

Figure 12, built with the following R code, provides a visualization similar to our IMSPE-based ones in Section 4.1.

```
R> plot(xgrid, f1d(xgrid), type = "l", xlab = "x", ylab = "y",
+    ylim = c(-6, 17), main = "Forrester example with EI, h = 5")
R> lines(xgrid, qnorm(0.05, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> lines(xgrid, qnorm(0.95, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> points(X, Y)
R> segments(mod$X0, rep(0, nrow(mod$X0)) - 6, mod$X0, (mod$mult - 6) * 0.5,
+    col = "gray")
R> lines(xgrid, p$mean, col = 2)
R> lines(xgrid, qnorm(0.05, p$mean, sqrt(pvar)), col = 2, lty = 2)
R> lines(xgrid, qnorm(0.95, p$mean, sqrt(pvar)), col = 2, lty = 2)
R> legend("top", c("truth", "estimate"), col = 1:2, lty = 1:2)
```

Observe that this lookahead-biased EI approach leads to substantial replication in the areas of interest, judiciously balancing exploration and exploitation in order to pin down the global minima of the mean surface. The actual number of unique locations is extracted as follows, which is a small fraction of the total budget of $N = 500$.

```
R> nrow(mod$X0)
```

```
[1] 60
```

Most of the replication is focused in the areas of primary interest from an optimization perspective, around $x \approx 0.1$, where we observe an inferior local minimum with large noise and, more intensively, around the true global optima located at the other end of the input space. EI is also available for TPs, and the corresponding modifications are provided in Appendix A.

Contour finding. A related problem is contour finding, also known as level-set estimation. The objective is to identify a region of inputs of interest: $\Gamma_f = \left\{\mathbf{x} \in \mathbb{R}^d : Y(\mathbf{x}) > T\right\}$ with T a given threshold, often zero without loss of generality. We describe briefly here the criteria defined by Lyu *et al.* (2018) for both GPs and TPs that are implemented in **hetGP**. Several

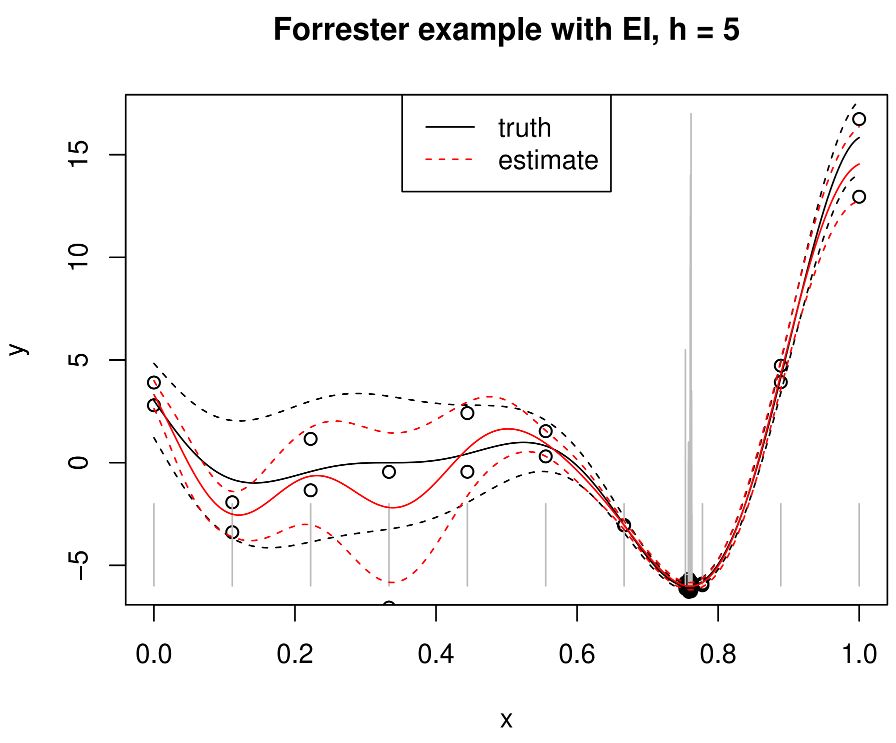

Figure 12: Sequential optimization with horizon $h = 5$. The truth is in black and the predictive distribution in red .

others can be found in the literature (Chevalier, Ginsbourger, Bect, and Molchanov 2013; Bogunovic *et al.* 2016; Azzimonti, Ginsbourger, Chevalier, Bect, and Richet 2016), with a selection of implementations provided by the **KrigInv** package (Chevalier, Picheny, and Ginsbourger 2014b; Chevalier, Picheny, Ginsbourger, and Azzimonti 2018).

The simplest criterion is maximum contour uncertainty (MCU), implemented by `crit_MCU` in the **hetGP** package. MCU is based on the local probability of misclassification, namely, that f is incorrectly predicted to be below or above the threshold (see also, e.g., Bichon, Eldred, Swiler, Mahadevan, and McFarland 2008; Ranjan, Bingham, and Michailidis 2008). A second criterion, contour stepwise uncertainty reduction (cSUR), implemented by `crit_cSUR` in **hetGP**, amounts to calculating a one-step-ahead reduction of MCU. A more computationally intensive, but more global, alternative involves integrating cSUR over the domain in a manner similar to IMSPE for variance reduction or so-called integrated expected conditional improvement (IECI, Gramacy and Lee 2011) for Bayesian optimization. In practice, the integral is approximated by a finite sum, which is the approach taken by `crit_ICU` in **hetGP**. The final criterion available, targeted mean square error (tMSE) (Picheny, Ginsbourger, Roustant, Haftka, and Kim 2010), is implemented in `crit_tMSE` and is designed to reduce the variance close to the limiting contour via a weight function.

For illustration, let us consider finding the zero contour in our simple 1D example. The code below illustrates the `crit_cSUR` criteria, the **hetGP** package's generic `crit_optim` interface.

Otherwise the setup and `for` loop are identical to those for our IMSPE and EI-based examples.

```
R> X <- seq(0, 1, length = 10)
R> X <- c(X, X)
R> Y <- fr(X)
R> mod <- mleHetGP(X = X, Z = Y, lower = 0.0001, upper = 10)
R>
R> for(i in 1:480) {
+    opt <- crit_optim(mod, crit = "crit_cSUR", h = 5)
+    X <- c(X, opt$par)
+    Ynew <- fr(opt$par)
+    Y <- c(Y, Ynew)
+    mod <- update(mod, Xnew = opt$par, Znew = Ynew, ginit = mod$g * 1.01)
+
+    if(i %% 25 == 0){
+      mod2 <- mleHetGP(X = list(X0 = mod$X0, Z0 = mod$Z0, mult = mod$mult),
+        Z = mod$Z, lower = 0.0001, upper = 1)
+      if(mod2$ll > mod$ll) mod <- mod2
+    }
+ }
R>
R> p <- predict(mod, matrix(xgrid, ncol = 1))
R> pvar <- p$sd2 + p$nugs
```

Again, we see that using the biasing lookahead procedure reduces the number of unique designs, which is beneficial in terms of speed as well as accuracy in general.

```
R> nrow(mod$X0)

[1] 135
```

In Figure 13, the repartition of unique designs is nontrivial around locations of crossing of the threshold, with larger spread (and less replication) where the crossing is in noisy areas. As with EI, this preliminary setup is likely to be improved upon after further research.

```
R> plot(xgrid, f1d(xgrid), type = "l", xlab = "x", ylab = "y",
+    ylim = c(-6, 17), main="Forrester example with cSUR, h = 5")
R> lines(xgrid, qnorm(0.05, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> lines(xgrid, qnorm(0.95, f1d(xgrid), fn(xgrid)), col = 1, lty = 2)
R> points(X, Y)
R> segments(mod$X0, rep(0, nrow(mod$X0)) - 6, mod$X0, (mod$mult - 6) * 0.5,
+    col="gray")
R> lines(xgrid, p$mean, col = 2)
R> lines(xgrid, qnorm(0.05, p$mean, sqrt(pvar)), col = 2, lty = 2)
R> lines(xgrid, qnorm(0.95, p$mean, sqrt(pvar)), col = 2, lty = 2)
R> legend("top", c("truth", "estimate"), col = 1:2, lty = 1:2)
R> abline(h = 0, col = "blue")
```

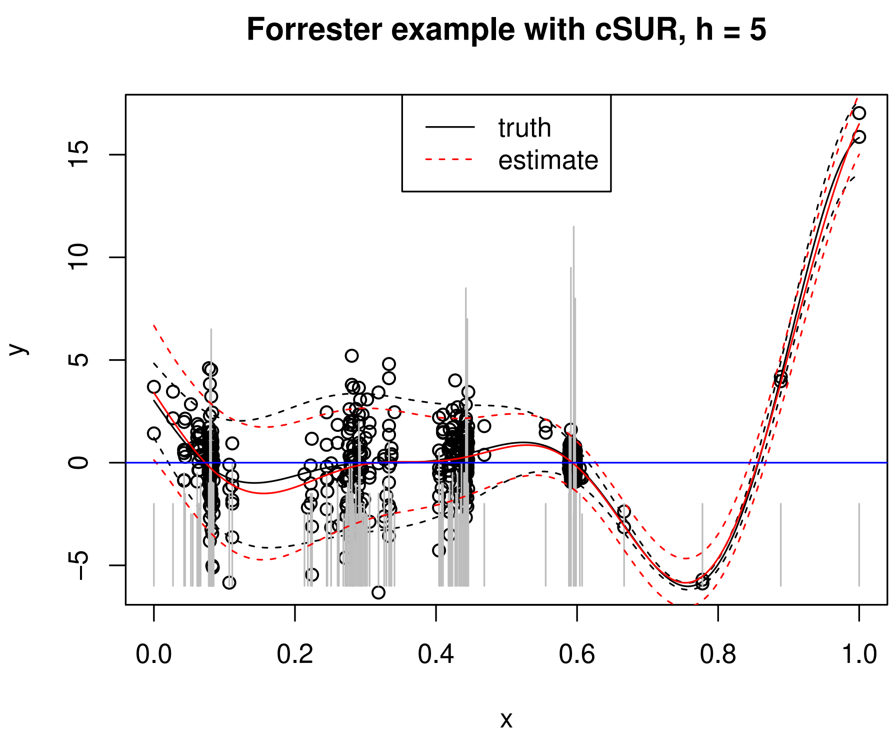

Figure 13: Sequential contour finding with horizon $h = 5$. The truth is in black and the predictive distribution in red.

5. Summary and discussion

We have introduced the R package **hetGP** for heteroskedastic Gaussian process regression, sequential experimental design, and Bayesian optimization. Although the package is designed for dealing with noise and changing signal-to-noise ratios in the setting of Gaussian process regression, it offers a full-featured GP regression approach. Ordinary homoskedastic (and noise-free) GP modeling is supported. When the data is observed with noise, the package implements a Woodbury trick to make inference more efficient, decomposing matrices sized by the number of unique input sites, rather than ones sized by the full data set. This leads to dramatically faster inference compared with other GP packages on CRAN when the level of replication is high.

Working only with unique inputs has other advantages, particularly it comes to coupled-GP inference for nonlinearly changing (latent) variance variables along with the usual mean-based analysis. By creating a unifying likelihood-based inferential framework, complete with closed-form derivative expressions, library-based optimization methods (e.g., `optim` in R) can be deployed for efficient heteroskedastic modeling. Whereas the earlier MCMC-based approaches could at best handle dozens of observations, **hetGP** can handle thousands in a reasonable amount of computing time.

Although relevant to machine learning and spatial data applications, the methods in the **hetGP** package target stochastic computer modeling applications, which often exhibit heterogeneous noise effects. Agent-based models are a good example. In that setting, the design of the experiment is at least as important as modeling. Here we introduced several sequential design schemes that work with **hetGP** model objects to organically grow the design toward accurate (low-variance) prediction, optimization, and the search for level sets. In all three scenarios, the scheme is able to balance exploration, exploitation, and replication as a means of obtaining efficient predictions for those targets. Extensions are provided to accommodate new sequential design acquisition strategies toward novel surrogate modeling and prediction applications.

Acknowledgments

The work of MB is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Contract No. DE-AC02-06CH11357. RBG gratefully acknowledges funding from a DOE LAB 17-1697 via subaward from Argonne National Laboratory for SciDAC/DOE Office of Science ASCR and High Energy Physics, and partial support from National Science Foundation grants DMS-1849794, DMS-1821258 and DMS-1621746. Many thanks to D. Austin Cole for comments on early drafts and to Gail Pieper for her useful language editing.

References

Anagnostopoulos C, Gramacy R (2013). “Information-Theoretic Data Discarding for Dynamic Trees on Data Streams.” *Entropy*, **15**(12), 5510–5535. ArXiv:1201.5568.

Ankenman B, Nelson BL, Staum J (2010). “Stochastic Kriging for Simulation Metamodeling.” *Operations Research*, **58**(2), 371–382.

Azzimonti D, Ginsbourger D, Chevalier C, Bect J, Richet Y (2016). “Adaptive Design of Experiments for Conservative Estimation of Excursion Sets.” *arXiv preprint arXiv:1611.07256.*

Banerjee S, Carlin BP, Gelfand AE (2004). *Hierarchical Modeling and Analysis for Spatial Data.* Chapman and Hall/CRC.

Banerjee S, Gelfand AE, Finley AO, Sang H (2008). “Gaussian Predictive Process Models for Large Spatial Data Sets.” *Journal of the Royal Statistical Society B*, **70**(4), 825–848.

Bichon BJ, Eldred MS, Swiler LP, Mahadevan S, McFarland JM (2008). “Efficient Global Reliability Analysis for Nonlinear Implicit Performance Functions.” *AIAA Journal*, **46**(10), 2459–2468.

Binois M, Gramacy RB, Ludkovski M (2018a). “Practical Heteroscedastic Gaussian Process Modeling for Large Simulation Experiments.” *Journal of Computational and Graphical Statistics*, **27**(4), 808–821. `doi:10.1080/10618600.2018.1458625`. `https://doi.org/10.1080/10618600.2018.1458625`, URL `https://doi.org/10.1080/10618600.2018.1458625`.

Binois M, Huang J, Gramacy RB, Ludkovski M (2018b). "Replication or Exploration? Sequential Design for Stochastic Simulation Experiments." *Technometrics*, **0**(ja), 1–43. doi:10.1080/00401706.2018.1469433. https://doi.org/10.1080/00401706.2018.1469433, URL https://doi.org/10.1080/00401706.2018.1469433.

Bogunovic I, Scarlett J, Krause A, Cevher V (2016). "Truncated Variance Reduction: A Unified Approach to Bayesian Optimization and Level-Set Estimation." In *Advances in Neural Information Processing Systems*, pp. 1507–1515.

Burnaev E, Panov M (2015). "Adaptive Design of Experiments Based on Gaussian Processes." In *Statistical Learning and Data Sciences*, pp. 116–125. Springer-Verlag.

Carnell R (2018). ***lhs**: Latin Hypercube Samples.* R package version 0.16, URL https://CRAN.R-project.org/package=lhs.

Chevalier C, Ginsbourger D, Bect J, Molchanov I (2013). "Estimating and Quantifying Uncertainties on Level Sets Using the Vorob'ev Expectation and Deviation with Gaussian Process Models." In D Ucinski, AC Atkinson, M Patan (eds.), *mODa 10 - Advances in Model-Oriented Design and Analysis*, Contributions to Statistics, pp. 35–43. Springer-Verlag. ISBN 978-3-319-00217-0. doi:10.1007/978-3-319-00218-7_5. URL http://dx.doi.org/10.1007/978-3-319-00218-7_5.

Chevalier C, Ginsbourger D, Emery X (2014a). "Corrected Kriging Update Formulae for Batch-Sequential Data Assimilation." In *Mathematics of Planet Earth*, pp. 119–122. Springer-Verlag.

Chevalier C, Picheny V, Ginsbourger D (2014b). "**KrigInv**: An Efficient and User-Friendly Implementation of Batch-Sequential Inversion Strategies Based on Kriging." *Computational Statistics & Data Analysis*, **71**, 1021–1034.

Chevalier C, Picheny V, Ginsbourger D, Azzimonti D (2018). ***KrigInv**: Kriging-Based Inversion for Deterministic and Noisy Computer Experiments.* R package version 1.4.1, URL https://CRAN.R-project.org/package=KrigInv.

Chung M, Binois M, Gramacy RB, Moquin DJ, Smith AP, Smith AM (2018). "Parameter and Uncertainty Estimation for Dynamical Systems Using Surrogate Stochastic Processes." *arXiv preprint arXiv:1802.00852*.

Dancik GM, Dorman KS (2008). "**mlegp**: Statistical Analysis for Computer Models of Biological Systems using R." *Bioinformatics*, **24**(17).

Deville Y, Ginsbourger D, Durrande ORCN (2018). ***kergp**: Gaussian Process Laboratory.* R package version 0.4.0, URL https://CRAN.R-project.org/package=kergp.

Erickson CB, Ankenman BE, Sanchez SM (2017). "Comparison of Gaussian Process Modeling Software." *European Journal of Operational Research*. ISSN 0377-2217. doi:https://doi.org/10.1016/j.ejor.2017.10.002. URL http://www.sciencedirect.com/science/article/pii/S0377221717308962.

Forrester A, Sobester A, Keane A (2008). *Engineering Design via Surrogate Modelling: A Practical Guide*. John Wiley & Sons.

Franck CT, Gramacy RB (2018). "Assessing Bayes Factor Surfaces Using Interactive Visualization and Computer Surrogate Modeling." *arXiv preprint arXiv:1809.05580*.

Frazier PI (2018). "A Tutorial on Bayesian Optimization." *arXiv preprint arXiv:1807.02811*.

Gauthier B, Pronzato L (2014). "Spectral Approximation of the IMSE Criterion for Optimal Designs in Kernel-Based Interpolation Models." *SIAM/ASA Journal on Uncertainty Quantification*, **2**(1), 805–825.

Ginsbourger D, Le Riche R (2010). "Towards Gaussian Process-Based Optimization with Finite Time Horizon." In *mODa 9–Advances in Model-Oriented Design and Analysis*, pp. 89–96. Springer-Verlag.

Ginsbourger D, Le Riche R, Carraro L (2010). "Kriging Is Well-Suited to Parallelize Optimization." In *Computational Intelligence in Expensive Optimization Problems*, pp. 131–162. Springer-Verlag.

Gneiting T, Raftery AE (2007). "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association*, **102**(477), 359–378. doi:10.1198/016214506000001437.

Goldberg PW, Williams CK, Bishop CM (1998). "Regression with Input-Dependent Noise: A Gaussian Process Treatment." In *Advances in Neural Information Processing Systems*, volume 10, pp. 493–499. MIT press, Cambridge, MA.

Gonzalez J, Osborne M, Lawrence N (2016). "GLASSES: Relieving the Myopia of Bayesian Optimisation." In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 790–799.

Gorodetsky A, Marzouk Y (2016). "Mercer Kernels and Integrated Variance Experimental Design: Connections between Gaussian Process Regression and Polynomial Approximation." *SIAM/ASA Journal on Uncertainty Quantification*, **4**(1), 796–828.

Gramacy R, Lee H (2012). "Cases for the Nugget in Modeling Computer Experiments." *Statistics and Computing*, **22**(3).

Gramacy R, Polson N (2011). "Particle Learning of Gaussian Process Models for Sequential Design and Optimization." *Journal of Computational and Graphical Statistics*, **20**(1), 102–118. doi:10.1198/jcgs.2010.09171.

Gramacy RB (2007). "**tgp**: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. URL http://www.jstatsoft.org/v19/i09/.

Gramacy RB (2016). "**laGP**: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R." *Journal of Statistical Software*, **72**(1), 1–46. doi:10.18637/jss.v072.i01.

Gramacy RB (2017). ***monomvn**: Estimation for Multivariate Normal and Student-t Data with Monotone Missingness*. R package version 1.9-7, URL https://CRAN.R-project.org/package=monomvn.

Gramacy RB, Apley DW (2015). "Local Gaussian Process Approximation for Large Computer Experiments." *Journal of Computational and Graphical Statistics*, **24**(2), 561–578. See arXiv:1303.0383.

Gramacy RB, Lee HKH (2009). "Adaptive Design and Analysis of Supercomputer Experiment." *Technometrics*, **51**(2), 130–145.

Gramacy RB, Lee HKH (2011). "Optimization under Unknown Constraints." In J Bernardo, S Bayarri, JO Berger, AP Dawid, D Heckerman, AFM Smith, M West (eds.), *Bayesian Statistics 9*, pp. 229–256. Oxford University Press.

Gramacy RB, Pantaleo E (2010). "Shrinkage Regression for Multivariate Inference with Missing Data, and an Application to Portfolio Balancing." *Bayesian Anal.*, **5**(2), 237–262. doi:10.1214/10-BA602. URL https://doi.org/10.1214/10-BA602.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with **tgp** Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. URL http://www.jstatsoft.org/v33/i06/.

Hong L, Nelson B (2006). "Discrete Optimization via Simulation Using COMPASS." *Operations Research*, **54**(1), 115–129.

Hu R, Ludkovski M (2017). "Sequential Design for Ranking Response Surfaces." *SIAM/ASA Journal on Uncertainty Quantification*, **5**(1), 212–239.

Huan X, Marzouk YM (2016). "Sequential Bayesian Optimal Experimental Design via Approximate Dynamic Programming." *arXiv preprint arXiv:1604.08320*.

Jacquier E, Polson N, Rossi PE (2004). "Bayesian Analysis of Stochastic Volatility Models with Fat-Tails and Correlated Errors." *J. of Econometrics*, **122**, 185–212.

Jones D, Schonlau M, Welch W (1998). "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global Optimization*, **13**(4), 455–492. URL http://www.springerlink.com/index/M5878111M101017P.pdf.

Kersting K, Plagemann C, Pfaff P, Burgard W (2007). "Most Likely Heteroscedastic Gaussian Process Regression." In *Proceedings of the International Conference on Machine Learning*, pp. 393–400. ACM, New York, NY.

Lam R, Willcox K, Wolpert DH (2016). "Bayesian Optimization with a Finite Budget: An Approximate Dynamic Programming Approach." In *Advances In Neural Information Processing Systems*, pp. 883–891.

Lazaro-Gredilla M, Titsias M (2011). "Variational Heteroscedastic Gaussian Process Regression." In *Proceedings of the International Conference on Machine Learning*, pp. 841–848. ACM, New York, NY.

Leatherman ER, Santner TJ, Dean AM (2017). "Computer Experiment Designs for Accurate Prediction." *Statistics and Computing*, pp. 1–13.

Lyu X, Binois M, Ludkovski M (2018). "Evaluating Gaussian Process Metamodels and Sequential Designs for Noisy Level Set Estimation." *arXiv preprint arXiv:1807.06712*.

MacDonald B, Ranjan P, Chipman H (2015). "**GPfit**: An R Package for Fitting a Gaussian Process Model to Deterministic Simulator Outputs." *Journal of Statistical Software*, **64**(12), 1–23. URL `http://www.jstatsoft.org/v64/i12/`.

Mockus J, Tiesis V, Zilinskas A (1978). "The Application of Bayesian Methods for Seeking the Extremum." *Towards Global Optimization*, **2**(117-129), 2.

Ng SH, Yin J (2012). "Bayesian Kriging Analysis and Design for Stochastic Systems." *ACM Transations on Modeling and Computer Simulation (TOMACS)*, **22**(3), article no. 17.

Opsomer J, Ruppert D, Wand W, Holst U, Hossler O (1999). "Kriging with Nonparametric Variance Function Estimation." *Biometrics*, **55**, 704–710.

Picheny V, Ginsbourger D, Roustant O, Haftka RT, Kim NH (2010). "Adaptive Designs of Experiments for Accurate Approximation of a Target Region." *Journal of Mechanical Design*, **132**(7), 071008.

Picheny V, Wagner T, Ginsbourger D (2012). "A Benchmark of Kriging-Based Infill Criteria for Noisy Optimization." *Structural and Multidisciplinary Optimization*, pp. 1–20.

Plumlee M, Tuo R (2014). "Building Accurate Emulators for Stochastic Simulations via Quantile Kriging." *Technometrics*, **56**(4), 466–473.

Pratola M, Chipman H, George E, McCulloch R (2017a). "Heteroscedastic BART Using Multiplicative Regression Trees." *arXiv preprint arXiv:1709.07542*.

Pratola MT, Harari O, Bingham D, Flowers GE (2017b). "Design and Analysis of Experiments on Nonconvex Regions." *Technometrics*, pp. 1–12.

Ranjan P, Bingham D, Michailidis G (2008). "Sequential Experiment Design for Contour Estimation from Complex Computer Codes." *Technometrics*, **50**(4), 527–541.

Rasmussen CE, Williams C (2006). *Gaussian Processes for Machine Learning*. MIT Press. URL `http://www.gaussianprocess.org/gpml/`.

Roustant O, Ginsbourger D, 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. URL `http://www.jstatsoft.org/v51/i01/`.

Sacks J, Welch WJ, Mitchell TJ, Wynn HP (1989). "Design and Analysis of Computer Experiments." *Statistical Science*, **4**(4), 409–423.

Seo S, Wallat M, Graepel T, Obermayer K (2000). "Gaussian Process Regression: Active Data Selection and Test Point Rejection." In *Proceedings of the International Joint Conference on Neural Networks*, volume III, pp. 241–246. IEEE.

Shah A, Wilson A, Ghahramani Z (2014). "Student-t Processes as Alternatives to Gaussian Processes." In S Kaski, J Corander (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine*

Learning Research, pp. 877–885. PMLR, Reykjavik, Iceland. URL http://proceedings.mlr.press/v33/shah14.html.

Shahriari B, Swersky K, Wang Z, Adams RP, de Freitas N (2016). "Taking the Human out of the Loop: A Review of Bayesian Optimization." *Proceedings of the IEEE*, **104**(1), 148–175.

Snelson E, Ghahramani Z (2005). "Sparse Gaussian Processes Using Pseudo-Inputs." In *Advances in Neural Information Processing Systems*, pp. 1257–1264.

Snoek J, Larochelle H, Adams RP (2012). "Bayesian Optimization of Machine Learning Algorithms." In *Neural Information Processing Systems (NIPS)*.

Vazquez E, Villemonteix J, Sidorkiewicz M, Walter E (2008). "Global Optimization Based on Noisy Evaluations: An Empirical Study of Two Statistical Approaches." In *Journal of Physics: Conference Series*, volume 135, p. 012100. IOP Publishing.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer-Verlag, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Wang Z, Shi JQ, Lee Y (2017). "Extended T-Process Regression Models." *Journal of Statistical Planning and Inference*, **189**, 38–60.

Xie G, Chen X (2017). "A Heteroscedastic T-Process Simulation Metamodeling Approach and its Application in Inventory Control and Optimization." In *Simulation Conference (WSC), 2017 Winter*, pp. 3242–3253. IEEE.

Xie J, Frazier P, Chick S (2012). "Assemble to Order Simulator." URL http://simopt.org/wiki/index.php?title=Assemble_to_Order&oldid=447.

A. TP modifications

The expression for EI with TPs is given, for example, in the work of Shah *et al.* (2014):

$$\mathrm{EI}_{\mathrm{TP}} = z(\mathbf{x})\sigma(\mathbf{x})\Lambda_{\alpha+N}(z(\mathbf{x})) + \sigma(\mathbf{x})\left(1 + \frac{z(\mathbf{x})^2 - 1}{\alpha + N - 1}\right)\lambda_{\nu+N}(z(\mathbf{x})),$$

with $z(\mathbf{x}) = \frac{y^* - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$ and λ, Λ the pdf and cdf of the Student-t distribution.
We provide the expression for the corresponding gradient:

$$\nabla \mathrm{EI}_{\mathrm{TP}}(\mathbf{x}) = \nabla\left\{z(\mathbf{x})\sigma(\mathbf{x})\Lambda_{\alpha+N}(z(\mathbf{x}))\right\} + \nabla\left\{\sigma(\mathbf{x})\left(1 + \frac{z(\mathbf{x})^2 - 1}{\alpha + N - 1}\right)\lambda_{\nu+N}(z(\mathbf{x}))\right\}$$

where $\nabla\left\{z(\mathbf{x})\sigma(\mathbf{x})\Lambda_{\alpha+N}(z(\mathbf{x}))\right\} = -\nabla m(\mathbf{x})\lambda(z(\mathbf{x})) + s(\mathbf{x})z(\mathbf{x})\nabla z(\mathbf{x})\lambda(z(\mathbf{x}))$ and

$$\begin{aligned}\nabla\left\{\sigma(\mathbf{x})\left(1 + \frac{z(\mathbf{x})^2 - 1}{b}\right)\lambda_{\nu+N}(z(\mathbf{x}))\right\} = &\left(\nabla s(\mathbf{x})(1 + \frac{z(\mathbf{x})^2 - 1}{b}) + 2s(\mathbf{x})z(\mathbf{x})\nabla z(\mathbf{x})/b\right)\lambda(z(\mathbf{x})) \\ &+ s(\mathbf{x})(1 + \frac{z(\mathbf{x})^2 - 1}{b})\nabla z(\mathbf{x})\lambda'_\alpha(z(\mathbf{x}))\end{aligned}$$

with $b = \alpha + N - 1$, and

$$\lambda'_\alpha(z) = \frac{(-\alpha - 1)z\Gamma(\frac{\alpha+1}{2})(\frac{\alpha+z^2}{\alpha})^{-\alpha/2-3/2}}{\sqrt{\pi}\alpha^{3/2}\Gamma(\frac{\alpha}{2})}.$$

Affiliation:

Mickaël Binois
Mathematics and Computer Science Division
Argonne National Laboratory
9700 Cass Ave.
Lemont, IL 60439, United States of America
E-mail: mbinois@mcs.anl.gov
URL: https://sites.google.com/site/mickaelbinoishomepage/

Robert B. Gramacy
Department of Statistics
Virginia Tech
250 Drillfield Drive
Blacksburg, VA 24061, United States of America
E-mail: rbg@vt.edu
URL: http://bobby.gramacy.com/