GPfit: An R Package for Fitting a Gaussian Process Model to Deterministic Simulator Outputs

Blake MacDonald  
Acadia University

Pritam Ranjan  
Acadia University

Hugh Chipman  
Acadia University

Abstract

Gaussian process (GP) models are commonly used statistical metamodels for emulating expensive computer simulators. Fitting a GP model can be numerically unstable if any pair of design points in the input space are close together. Ranjan, Haynes, and Karsten (2011) proposed a computationally stable approach for fitting GP models to deterministic computer simulators. They used a genetic algorithm based approach that is robust but computationally intensive for maximizing the likelihood. This paper implements a slightly modified version of the model proposed by Ranjan et al. (2011) in the R package GPfit. A novel parameterization of the spatial correlation function and a clustering based multi-start gradient based optimization algorithm yield robust optimization that is typically faster than the genetic algorithm based approach. We present two examples with R codes to illustrate the usage of the main functions in GPfit. Several test functions are used for performance comparison with the popular R package mlegp. We also use GPfit for a real application, i.e., for emulating the tidal kinetic energy model for the Bay of Fundy, Nova Scotia, Canada. GPfit is free software and distributed under the General Public License and available from the Comprehensive R Archive Network.

Keywords: computer experiments, clustering, near-singularity, nugget.

1. Introduction

Computer simulators are often used to model complex physical and engineering processes that are either infeasible, too expensive or time consuming to observe. Examples include tracking the population for bowhead whales in Western Arctic (Poole and Raftery 2000), monitoring traffic control system (Medina, Moreno, and Royo 2005), dynamics of dark energy and dark matter in cosmological studies (Arbey 2006), and the tidal kinetic energy in the Bay of Fundy, Nova Scotia, Canada (Ranjan et al. 2011). Realistic computer simulators can still be computationally expensive to run, and they are often approximated (or emulated) using statistical models. Sacks, Welch, Mitchell, and Wynn (1989) proposed emulating such
an expensive deterministic simulator as a realization of a Gaussian stochastic process (GP). This paper presents the R (R Core Team 2014) package \texttt{GPfit} (MacDonald, Chipman, and Ranjan 2014) for robust and computationally efficient fitting of GP models to deterministic simulator outputs.

The computational stability of GP estimation algorithms can depend critically on the set of design points and corresponding simulator outputs that are used to build a GP model. If any pair of design points in the input space are close together, the spatial correlation matrix $R$ may become near-singular and hence the GP model fitting procedure computationally unstable. A popular approach to overcome this numerical instability is to introduce a small “nugget” parameter $\delta$ in the model, i.e., $R$ is replaced by $R_\delta = R + \delta I$, that is estimated along with the other model parameters (e.g., Neal 1997; Booker, Dennis, Frank, Serafini, Torczon, and Trosset 1999; Santner, Williams, and Notz 2003; Gramacy and Lee 2008). However, adding a nugget in the model introduces additional smoothing in the predictor and as a result the predictor is no longer an interpolator. Thus, it is challenging to choose an appropriate value of $\delta$ that maintains the delicate balance between the stabilization and minimizing the over-smoothing of the model predictions. Ranjan \textit{et al.} (2011) proposed a computationally stable approach by introducing a lower bound on the nugget, which minimizes unnecessary over-smoothing and improves the model accuracy. Assuming space-filling designs are used for fitting the GP models, the nugget based approach is useful in problems with large $n$ (size of the training data) and small $d$ (input dimension).

Instead of trying to interpolate the data, one may argue that all simulators are noisy and the statistical surrogates should always smooth the simulator data (e.g., Gramacy and Lee 2012). In spite of the recent interest in stochastic simulators (e.g., Poole and Raftery 2000; Arbey 2006), deterministic simulators are still being actively used. For instance, Medina \textit{et al.} (2005) demonstrate the preference of deterministic traffic simulators over their stochastic counterparts. The model considered in \texttt{GPfit} assumes that the computer simulator is deterministic and is very similar to the GP model proposed in Ranjan \textit{et al.} (2011).

The maximum likelihood approach for fitting the GP model requires optimizing the log-likelihood, which can often have multiple local optima (Yuan, Wang, Yu, and Fang 2008; Schirru, Pampuri, Nicolao, and McLoone 2011; Kalaitzis and Lawrence 2011; Petelin, Filipiš, and Kocijan 2011). This makes the model fitting procedure computationally challenging. Ranjan \textit{et al.} (2011) uses a genetic algorithm (GA) approach, which is robust but computationally intensive for likelihood optimization. \texttt{GPfit} uses a multi-start gradient based search algorithm that is robust and typically faster than the GA used in Ranjan \textit{et al.} (2011). A clustering based approach on a large space-filling design over the parameter space is used for the multiple starts, i.e., choosing the initial values of the gradient search algorithm. Furthermore, we propose a new parameterization of the spatial correlation function that simplifies likelihood optimization.

The remainder of the paper is organized as follows. Section 2 presents a brief review of the GP model in Ranjan \textit{et al.} (2011), the new parameterization of the correlation function and the proposed optimization algorithm implemented in \texttt{GPfit}. In Section 3, the main functions of \texttt{GPfit} and their arguments are discussed. Two examples illustrating the usage of \texttt{GPfit} are presented in Section 4. Section 5 compares \texttt{GPfit} with other popular R packages. This includes an empirical performance comparison with the popular R package \texttt{mlegp} (Dancik and Dorman 2008; Dancik 2013). Section 6 shows the usage of \texttt{GPfit} for emulating tidal energy model outputs. The paper concludes with a few remarks in Section 7.
2. Methodology

Section 2.1 reviews the GP model proposed in Ranjan et al. (2011) (for more details on GP models, see Santner et al. 2003 and Rasmussen and Williams 2006). We propose a new parameterization of the correlation function in Section 2.2 that facilitates optimization of the likelihood. The proposed optimization algorithm implemented in GPfit is presented in Section 2.3.

2.1. Gaussian process model

Let the \(i\)-th input and the corresponding output of the computer simulator be denoted by a \(d\)-dimensional vector, \(x_i = (x_{i1}, \ldots, x_{id})^\top\) and \(y_i = y(x_i)\) respectively. The experimental design \(D_0 = \{x_1, \ldots, x_n\}\) is the set of \(n\) input trials stored in an \(n \times d\) matrix \(X\). We assume \(x_i \in [0, 1]^d\). The outputs are held in the \(n \times 1\) vector \(Y = y(X) = (y_1, \ldots, y_n)^\top\). The simulator output, \(y(x_i)\), is modeled as

\[
y(x_i) = \mu + z(x_i); \quad i = 1, \ldots, n,
\]

where \(\mu\) is the overall mean, and \(z(x_i)\) is a GP with \(\mathbb{E}(z(x_i)) = 0\), \(\text{VAR}(z(x_i)) = \sigma^2\), and \(\text{Cov}(z(x_i), z(x_j)) = \sigma^2 R_{ij}\). In general, \(y(X)\) has a multivariate normal distribution, \(N_n(1_n \mu, \Sigma)\), where \(\Sigma = \sigma^2 R\) is formed with correlation matrix \(R\) having elements \(R_{ij}\), and \(1_n\) is a \(n \times 1\) vector of all ones. Although there are several choices for the correlation structure, we follow Ranjan et al. (2011) and use the Gaussian correlation function given by

\[
R_{ij} = \prod_{k=1}^d \exp\{-\theta_k|x_{ik} - x_{jk}|^2\}, \quad \text{for all } i, j,
\]

where \(\theta = (\theta_1, \ldots, \theta_d) \in [0, \infty)^d\) is a vector of hyper-parameters. The closed form estimators of \(\mu\) and \(\sigma^2\) given by

\[
\hat{\mu}(\theta) = (1_n^\top R^{-1} 1_n)^{-1} (1_n^\top R^{-1} Y) \quad \text{and} \quad \hat{\sigma}^2(\theta) = \frac{(Y - 1_n \hat{\mu}(\theta))^\top R^{-1} (Y - 1_n \hat{\mu}(\theta))}{n},
\]

are used to obtain the negative profile log-likelihood (from here onwards referred to as deviance)

\[-2 \log(L_0) \propto \log(|R|) + n \log[(Y - 1_n \hat{\mu}(\theta))^\top R^{-1} (Y - 1_n \hat{\mu}(\theta))]\]

for estimating the hyper-parameters \(\theta\), where \(|R|\) denotes the determinant of \(R\).

Following the maximum likelihood approach, the best linear unbiased predictor at \(x^*\) (as shown in Sacks et al. 1989) is

\[
\hat{y}(x^*) = \hat{\mu} + r^\top R^{-1} (Y - 1_n \hat{\mu}) = \left[\frac{(1 - r^\top R^{-1} 1_n)}{1_n^\top R^{-1} 1_n} 1_n^\top + r^\top\right] R^{-1} Y = C^\top Y,
\]

with mean squared error

\[
s^2(x^*) = \text{E}[(\hat{y}(x^*) - y(x^*))^2] = \sigma^2(1 - 2C^\top r + C^\top RC) = \sigma^2\left(1 - r^\top R^{-1} r + \frac{(1 - 1_n^\top R^{-1} 1_n)^2}{1_n^\top R^{-1} 1_n}\right),
\]
where \( r = (r_1(x^*), \ldots, r_n(x^*))^\top \), and \( r_i(x^*) = \text{COR}(z(x^*), z(x_i)) \). In practice, the parameters \( \mu, \sigma^2 \) and \( \theta \) are replaced with their respective estimates.

Fitting a GP model to \( n \) data points requires the repeated computation of the determinant and inverse of the \( n \times n \) correlation matrix \( R \). Such correlation matrices are positive definite by definition, however, the computation of \(|R|\) and \( R^{-1} \) can sometimes be unstable due to near-singularity. An \( n \times n \) matrix \( R \) is said to be near-singular (or, ill-conditioned) if its condition number \( \kappa(R) = \|R\| \cdot \|R^{-1}\| \) is too large, where \( \| \cdot \| \) denotes the \( L_2 \)-matrix norm (see Ranjan et al. 2011 for details). Near-singularity prohibits precise computation of the deviance and hence the parameter estimates. This is a common problem in fitting GP models which occurs if any pair of design points in the input space are close together (Neal 1997).

A popular approach to overcome near-singularity is to introduce a small nugget or jitter parameter, \( \delta \in (0, 1) \), in the model (i.e., \( R \) is replaced by \( R_\delta = R + \delta I \)) that is estimated along with the other model parameters. Replacing \( R \) with \( R_\delta \) in the GP model introduces additional smoothing of the simulator data that is undesirable for emulating a deterministic simulator. Ranjan et al. (2011) proposed a lower bound on \( \delta \) that minimizes the unnecessary over-smoothing. The lower bound given by Ranjan et al. (2011) is

\[
\delta_{lb} = \max \left\{ \frac{\lambda_n(\kappa(R) - e^a)}{\kappa(R)(e^a - 1)}, 0 \right\},
\]

where \( \lambda_n \) is the largest eigenvalue of \( R \) and \( e^a \) is the threshold of \( \kappa(R) \) that ensures a well conditioned \( R \). Ranjan et al. (2011) suggest \( a = 25 \) for space-filling Latin hypercube designs (LHDs; McKay, Beckman, and Conover 1979).

\textbf{GPfit} uses the GP model with \( R_{\delta_{lb}} = R + \delta_{lb} I \). The R package \texttt{mlegp}, used for performance comparison of \textbf{GPfit} in Section 5, implements the classical GP model with \( R \) replaced by \( R_\delta = R + \delta I \), and estimates \( \delta \) along with other hyper-parameters by minimizing the deviance. In both approaches the deviance function happens to be bumpy with multiple local optima. Next, we investigate a novel parameterization of the correlation function that makes the deviance easier to optimize.

### 2.2. Reparameterization of the correlation function

The key component of fitting the GP model described in Section 2.1 is the estimation of the correlation parameters by minimizing the deviance

\[
-2 \log(L_\theta) \propto \log(|R_{\delta_{lb}}|) + n \log(\|Y - 1_n \hat{\mu}(\theta)\|^\top R_{\delta_{lb}}^{-1}(Y - 1_n \hat{\mu}(\theta))).
\]

The deviance surface can be bumpy and have several local optima. For instance, the deviance functions for two examples in Section 4 are displayed in Figure 1. Figure 1 shows that the deviance function is bumpy near \( \theta = 0 \) and there are multiple local optima. Evolutionary algorithms like GA (used by Ranjan et al. 2011) are often robust for such objective functions, however, they can be computationally intensive (especially, because the computational cost of \( |R| \) and \( R^{-1} \) is \( O(n^3) \) and evolutionary algorithms often employ many evaluations of the objective function). Gradient-based optimization might be faster but will require careful selection of initial values to achieve the global minimum of the deviance function. It may be tempting to use a space-filling design over the parameter space for the starting points, however, such designs (e.g., maximin LHD) often tend to stay away from
Figure 1: The plots show the deviance (3) w.r.t. the GP parameter(s) \( \theta \). Panels (a) and (b) correspond to Example 1 (with \( d = 1, n = 10 \)), and (c) and (d) display deviance for Example 2 (with \( d = 2, n = 30 \)). Panels (b) and (d) are enlargements of (a) and (b) near 0, respectively.

the boundaries and corners. This is unfavorable because the deviance functions (Figure 1) fluctuate rapidly near \( \theta = 0 \).

To address the issue of a bumpy deviance surface near the boundaries of the parameter space, we propose a new parameterization of \( R \). Let \( \beta_k = \log_{10}(\theta_k) \) for \( k = 1, \ldots, d \), then

\[
R_{ij} = \prod_{k=1}^{d} \exp \left\{ -10^{\beta_k} |x_{ik} - x_{jk}|^2 \right\}, \quad \text{for all } i, j, \tag{4}
\]

where \( \beta_k \ll 0 \) implies a very high spatial correlation or a relatively flat surface in the \( k \)-th coordinate, and \( \beta_k \gg 0 \) implies low correlation, or a very wiggly surface with respect to the \( k \)-th input factor. One can also use automatic relevance determination approach to effectively remove irrelevant inputs (Neal 1997; Williams and Rasmussen 1996). Figure 2 displays the two
Figure 2: The plots show the deviance under the $\beta$-parameterization of $R$ given in (4), for the same examples and data as in Figure 1. Panels (a) and (b) correspond to Example 1 (with $d = 1, n = 10$), and (c) and (d) display the deviance for Example 2 (with $d = 2, n = 30$). Panels (b) and (d) are enlargements of (a) and (b) near 0, respectively.

deviance surfaces (shown in Figure 1) under the $\beta$-parameterization of $R$ given in (4). Though the new parameterization of $R$ in (4) results in an unbounded parameter space $\Omega = (-\infty, \infty)^d$, the peaks and dips of the deviance surface are now in the middle of the search space. This should facilitate a thorough search through the local optima and the choice of a set of initial values for a gradient based search.

**GPfit** uses a multi-start gradient based search algorithm for minimizing the deviance. The gradient based approach is often computationally fast, and careful selection of the multiple initial values of the search algorithm makes our implementation robust.
2.3. Optimization algorithm

A standard gradient based search algorithm like L-BFGS-B (Byrd, Lu, Nocedal, and Zhu 1995) finds the local optimum closest to the initial value, and thus often gets stuck in the wrong local optima. Our objective is to find the $\beta$ that corresponds to the global minimum of the deviance function. Kalaitzis and Lawrence (2011) argue that a slightly suboptimal solution of the deviance optimization problem may not always be a threat in the GP model setup, as alternative interpretations can be used to justify the model fit. However, suboptimal parameter estimates may lead to either overfitting or oversmoothing of the training data, which results in highly inaccurate prediction at unsampled input locations. In an attempt to obtain a good fit of the GP model, \texttt{GPfit} uses a multi-start L-BFGS-B algorithm for optimizing the deviance $-2\log(L_\beta)$. We first find a subregion $\Omega_0$ of the parameter space $\Omega = (-\infty, \infty)^d$ that is likely to contain the optimal parameter values. Then, a set of initial values for L-BFGS-B is carefully chosen to cover $\Omega_0$.

The structural form of the spatial correlation function (4) guarantees that its value lies in $[0, 1]$. That is, excluding the extreme cases of perfectly correlated and absolutely uncorrelated observations, $R_{ij}$ can be approximately bounded as:

$$\exp\{-5\} = 0.0067 \leq R_{ij} \leq 0.9999 = \exp\{-10^{-4}\},$$

or equivalently,

$$10^{-4} \leq \sum_{k=1}^{d} 10^{\beta_k}|x_{ik} - x_{jk}|^2 \leq 5. \quad (5)$$

To convert the bounds above into workable ranges for the $\beta_k$, we need to consider ranges for $|x_{ik} - x_{jk}|$. Assuming the objective is to approximate the overall simulator surface in $[0, 1]^d$, Loeppky, Sacks, and Welch (2009) argue that $n = 10 \cdot d$ is a good rule of thumb for determining the size of a space-filling design over the input locations of the simulator. In this case, the maximum value of the minimum inter-point distance along the $k$-th coordinate is $|x_{ik} - x_{jk}| \approx 1/10$. Furthermore, if we also make the simplifying assumption that the simulator is equally smooth in all directions, i.e., $\beta_k = \beta_0$, then (5) simplifies to

$$-2 \leq \log_{10}(d) \leq \beta_k \leq \log_{10}(500) - \log_{10}(d).$$

That is, $\Omega_0 = \{(\beta_1, \ldots, \beta_d) : -2 \leq \log_{10}(d) \leq \beta_k \leq \log_{10}(500) - \log_{10}(d), k = 1, \ldots, d\}$ is the set of $\beta = (\beta_1, \ldots, \beta_d)$ values that is likely to contain the likelihood optimizer. We use $\Omega_0$ for restricting the initial values of the L-BFGS-B algorithm to a manageable area, and the optimal solutions can be found outside this range.

The initial values for L-BFGS-B can be chosen using a large space-filling LHD on $\Omega_0$. However, Figure 2 shows that some parts of the likelihood surface are roughly flat, and multiple starts of L-BFGS-B in such regions might be unnecessary. We use a combination of $k$-means clustering applied to the design of parameter values, and evaluation of the deviance to reduce a large LHD to a more manageable set of initial values. Since the construction of $\Omega_0$ assumed the simplification $\beta_k = \beta_0$ for all $k$, and in some cases, for instance, in Figure 2(c), the deviance surface appears symmetric in the two coordinates, we enforce the inclusion of an additional initial value of L-BFGS-B on the main diagonal of $\Omega_0$. This diagonal point is the best of three L-BFGS-B runs only along the main diagonal, $\beta_k = \beta_0$ for all $k$.

The deviance optimization algorithm is summarized as follows:
1. Choose a 200d-point maximin LHD for $\beta = (\beta_1, \ldots, \beta_d)$ in the hyper-rectangle $\Omega_0$.
2. Choose the 80d values of $\beta$ that correspond to the smallest $-2\log(L_\beta)$ values.
3. Use the $k$-means clustering algorithm on these 80d points to find 2d groups. For robustness, we use five random restarts of $k$-means and choose the one with minimum total within-cluster sum-of-squares.
4. For $d \geq 2$, run the L-BFGS-B algorithm along the main diagonal of $\Omega_0$ starting at three equidistant points on the diagonal (i.e., at 25%, 50% and 75%). Choose the best of the three L-BFGS-B outputs, i.e., with smallest $-2\log(L_\beta)$ value.
5. These 2d + 1 (or 2 if $d = 1$) initial values, found in Steps 3 and 4, are then used in the L-BFGS-B routine to find the smallest $-2\log(L_\beta)$ and corresponding $\hat{\beta}_{mle} \in \Omega$.

The multi-start L-BFGS-B algorithm outlined above requires \( (200d + \sum_{i=1}^{2d+1} \eta_i + \sum_{j=1}^{3} \eta_j') \) deviance evaluations, where $\eta_i$ is the number of deviance evaluations for the $i$-th L-BFGS-B run in $\Omega$ space, and $\eta_j'$ is the number of deviance evaluations for the $j$-th L-BFGS-B run along the diagonal of the $\Omega_0$ space. For every iteration of L-BFGS-B, the algorithm computes one gradient (i.e., 2d deviance evaluations) and adaptively finds the location of the next step. That is, $\eta_i$ and $\eta_j'$ may vary, and the total number of deviance evaluations in the optimization process cannot be determined. Nonetheless, the empirical evidence based on the examples in Sections 4 and 5 suggest that the optimization algorithm used here is much faster than the GA in Ranjan et al. (2011) which uses 1000d^2 evaluations of (3) for fitting the GP model in $d$-dimensional input space. Both deviance minimization approaches have a few tunable parameters, for instance, the initial values and the maximum number of iterations (maxit) in L-BFGS-B, and the population size and number of generations in a GA, that can perhaps be adjusted to get better performance (i.e., fewer deviance calls to achieve the same accuracy in optimizing the deviance surface).

3. GPfit package

In this section, we discuss various GPfit functions that implement our proposed model, which is the computationally stable version of the GP model proposed by Ranjan et al. (2011) with the new parameterization of the correlation matrix $R$ (Section 2.2), and optimization algorithm described in Section 2.3.

The main functions for the users of GPfit are GP_fit(), predict() and (for $d \leq 2$) plot(). Both predict() and plot() methods for objects returned by GP_fit() allow for providing prediction and plots respectively. The code for fitting the GP model to $n$ data points in $d$-dimensional input space stored in an $n \times d$ matrix $X$ and an $n$-vector $Y$ is:

\[
\text{GP_fit}(X, Y, \text{control} = c(200 \ast d, 80 \ast d, 2 \ast d), \text{nug_thres} = 20, \\
\text{trace} = \text{FALSE}, \text{maxit} = 100)
\]

The default values of control, nug_thres, trace and maxit worked smoothly for all the examples implemented in this paper, however, they can be changed if necessary. In particular, control and maxit provide a balance between robustness and computational efficiency in optimizing the deviance.
Detailed description of the arguments are as follows:

- **control**: A vector of three tunable parameters used in the deviance optimization algorithm. The default values correspond to choosing $2 \times d$ clusters (using $k$-means clustering) based on $80 \times d$ best points (smallest deviance) from a $200 \times d$-point random maximin LHD in $\Omega_0$.

- **nug_thres**: A threshold parameter used in the calculation of the lower bound of the nugget, $\delta_{lb}$. Although Ranjan et al. (2011) suggest $nug\_thres = 25$ for space-filling designs, we use a conservative default value $nug\_thres = 20$. This value might change for different design schemes.

- **trace**: A flag that indicates whether or not to print the information on the final runs of the L-BFGS-B algorithm. The default $trace = FALSE$ implies no printing.

- **maxit**: An integer number that indicates the maximum number of iterations per L-BFGS-B run in the deviance optimization. We use the optim default $maxit = 100$.

$GP\_fit()$ returns the object of class ‘GP’ that contains the data set $X, Y$ and the estimated model parameters $\hat{\beta}, \hat{\sigma}^2$ and $\hat{\delta}_{lb}(\hat{\beta})$. Assuming $GPmodel$ is the ‘GP’ class object, $print(GPmodel, \ldots)$ presents the values of the object $GPmodel$, and options like $digits$ can be used for $\ldots$ As an alternative, one can use $summary(GPmodel)$ to get the same output.

$predict(GPmodel, xnew)$ returns the predicted response $\hat{y}(x^*)$ and the associated MSE $s^2(x^*)$ for every input $x^*$ in $xnew$. It also returns a data frame with the predictions combined with the $xnew$. Expressions for $\hat{y}(x^*)$ and $s^2(x^*)$ are shown in Section 2.1 subject to the replacement of $R$ with $R_{\delta_{lb}(\hat{\beta}_{mle})} = R + \delta_{lb}(\hat{\beta}_{mle})I$. The default value of $xnew$ is the design matrix $X$ used for model fitting.

The plotting function $plot()$ takes the ‘GP’ class object as input and depicts the model predictions and the associated MSES over a regular grid of the $d$-dimensional input space for $d = 1$ and 2. Various graphical options can be specified as additional arguments:

$$plot(GPmodel, range = c(0, 1), resolution = 50, colors = c("black", "blue", "red"), line_type = c(1, 1), pch = 1, cex = 2, surf_check = FALSE, response = TRUE, \ldots)$$

For $d = 1$, $plot()$ generates the predicted response $\hat{y}(x)$ and uncertainty bounds $\hat{y}(x) \pm 2s(x)$ over a regular grid of $resolution$ many points in the specified $range = c(0, 1)$. The graphical arguments $colors$, $line\_type$, $pch$ and $cex$ are only applicable for one-dimensional plots. One can also provide additional graphical argument in $\ldots$ for changing the plots (see $par$ in the base R function $plot()$).

For $d = 2$, the default arguments of $plot()$ with ‘GP’ class object produces a level plot of $\hat{y}(x^*)$. The plots are based on the model predictions using $predict()$ at a $resolution \times resolution$ regular grid over $[0, 1]^2$. The argument $surf\_check = TRUE$ can be used to generate a surface plot instead, and MSES can be plotted by using $response = FALSE$. Options like $shade$ and $drape$ from the $wireframe()$ function, $contour$ and $cuts$ from the $levelplot()$ function in $lattice$ (Sarkar 2008), and color specific arguments in package $colorspace$ (Zeileis, Hornik, and Murrell 2009; Ihaka, Murrell, Hornik, Fisher, and Zeileis 2015) can also be passed in for $\ldots$
4. Examples using GPfit

This section demonstrates the usage of GPfit functions and the interpretation of the outputs of the main functions. Two test functions are used as computer simulators to illustrate the functions of this package.

Example 1  Let \( x \in [0, 1] \), and the computer simulator output, \( y(x) \), be generated using the simple one-dimensional test function

\[
y(x) = \log(x + 0.1) + \sin(5\pi x),
\]

referred to as the function \texttt{computer_simulator} below. Suppose we wish to fit the GP model to a data set collected over a random maximin LHD of size \( n = 7 \). The design can be generated using the \texttt{maximinLHS} function in the \texttt{R} package \texttt{lhs} (Carnell 2012; Stein 1987). The following \texttt{R} code shows how to load the packages, generate the simulator outputs and then fit the GP model using \texttt{GP_fit()}.

\begin{verbatim}
R> library("GPfit")
R> library("lhs")
R> n <- 7
R> x <- maximinLHS(n, 1)
R> y <- matrix(0, n, 1)
R> for (i in 1:n) y[i] <- computer_simulator(x[i])
R> GPmodel <- GP_fit(x, y)
\end{verbatim}

The proposed optimization algorithm used only 227 deviance evaluations for fitting this GP model. The parameter estimates of the fitted GP model are obtained using \texttt{print(GPmodel)}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3}
\caption{The plot shows the model predictions and uncertainty bands for Example 1. The true simulator output curve is also displayed by the dash-dotted line.}
\end{figure}
Figure 4: The plots illustrate the usage of graphical parameters in `plot()` for Example 1. Panel (a) shows the model prediction and uncertainty plot with default graphical parameters, (b) illustrates the change due to `line_type`, (c) highlights the point size using `cex`, and (d) shows the usage of `pch` in changing the point character.

For printing only four significant decimal places, `digits = 4` can be used in `print()`.

Number Of Observations: n = 7
Input Dimensions: d = 1

Correlation: Exponential (power = 2)
Correlation Parameters:
  beta_hat
  [1]  1.977
12

**GPfit**: Gaussian Process Model Fitting in R

\[ \sigma^2_{\text{hat}}: [1] 0.7444 \]
\[ \delta_{\text{lb}}(\beta_{\text{hat}}): [1] 0 \]
nugget threshold parameter: 20

The `GPmodel` object can be used to predict and then plot the simulator outputs at a grid of inputs using `plot(GPmodel, ...)`. Figures 3 and 4 show the model prediction along with the uncertainty bounds \( \hat{y}(x^*) \pm 2s(x^*) \) on the uniform grid with `resolution = 100`. Figure 3 compares the predicted and the true simulator output. Figure 4 illustrates the usage of the graphical arguments of `plot()`. `predict(GPmodel, xnew)` can also be used to obtain model predictions at an arbitrary set of inputs, `xnew`, in the design space (i.e., not a grid).

**Example 2**

We now consider a two-dimensional test function to illustrate different functions of the `GPfit` package. Let \( x = (x_1, x_2) \in [-2,2]^2 \), and the simulator outputs be generated from the GoldPrice function (Andre, Siarry, and Dognon 2000)

\[
y(x) = \left[ 1 + (x_1 + x_2 + 1)^2\left\{ 19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2 \right\} \right] \times \left[ 30 + (2x_1 - 3x_2)^2\left\{ 18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2 \right\} \right].
\]

For convenience the inputs are scaled to \([0,1]^2\). The `GP_fit()` output from fitting the GP model to a data set based on a 20-point maximin LHD is as follows:

Number Of Observations: \( n = 20 \)
Input Dimensions: \( d = 2 \)
Correlation: Exponential (power = 2)
Correlation Parameters:
\[ \beta_{\text{hat}}.1 \beta_{\text{hat}}.2 \]
[1] 0.8578 1.442

\[ \sigma^2_{\text{hat}}: [1] 4.52e+09 \]
\[ \delta_{\text{lb}}(\beta_{\text{hat}}): [1] 0 \]
nugget threshold parameter: 20

For fitting this GP model, the proposed multi-start L-BFGS-B optimization procedure used only 808 deviance evaluations, whereas the GA based optimization in Ranjan et al. (2011) would have required 4000 deviance calls. The correlation hyper-parameter estimate \( \hat{\beta}_{\text{mle}} = (0.8578, 1.442) \) shows that the fitted simulator is slightly more active (or wiggly) in the \( X_2 \) variable. The nugget parameter \( \delta_{\text{lb}}(\beta_{\text{mle}}) = 0 \) implies that the correlation matrix with the chosen design points and \( \beta = \hat{\beta}_{\text{mle}} \) is well-behaved.

The following code illustrates the usage of `predict()` for obtaining predicted response and associated MSEs at a set of unobserved inputs.

R> `xnew` <- matrix(runif(20), ncol = 2)
R> `Model_pred` <- `predict(GPmodel, xnew)`

The model prediction outputs stored in object `Model_pred` are as follows:
Figure 5: The plots illustrate the usage of graphical parameters in `plot()` for Example 2. Panel (a) shows the default plot (the levelplot of $\hat{y}(x^*)$) with additional color specification, (b) presents the levelplot with contour lines of $s^2(x^*)$, (c) shows the surface plot of $\hat{y}(x^*)$, and (d) displays the surface plot of $s^2(x^*)$.

$\hat{Y}$

<table>
<thead>
<tr>
<th></th>
<th>561.3877</th>
<th>-372.5221</th>
<th>13287.0495</th>
<th>3148.5904</th>
<th>5129.1136</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td>8188.2805</td>
<td>3626.4985</td>
<td>14925.8142</td>
<td>2869.6225</td>
<td>217039.3229</td>
</tr>
</tbody>
</table>

$\text{MSE}$

<table>
<thead>
<tr>
<th></th>
<th>186119713</th>
<th>21523832</th>
<th>86391757</th>
<th>8022989</th>
<th>562589770</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13698589</td>
<td>123121468</td>
<td>1167409027</td>
<td>1483924477</td>
<td>264176788</td>
</tr>
</tbody>
</table>

Panel (a): Default arguments
Panel (b): $\text{response} = \text{FALSE}$, $\text{contour} = \text{TRUE}$
Panel (c): $\text{surf_check} = \text{TRUE}$
Panel (d): $\text{response} = \text{FALSE}$, $\text{surf_check} = \text{TRUE}$
The `GPfit` function `plot()` calls `predict()` for computing $\hat{y}(x^*)$ and $s^2(x^*)$ at a regular resolution $\times$ resolution grid in the input space defined by the `range` parameter. Recall from Section 3 that `colors`, `line_type`, `pch` and `cex` are only applicable for one dimensional plots. For $d = 2$, the following code can be used to draw the level/contour and surface plots of $\hat{y}(x)$ and $s^2(x)$ over a specified grid resolution.

```R
plot(GPmodel, range = c(0, 1), resolution = 50, surf_check = FALSE,
     response = TRUE, ...)
```

Additional graphical arguments, for instance, from `lattice` and `colorspace`, can also be passed in for ... to enhance the plotting features. Figure 5 shows the model predictions and the MSEs on the uniform 50 $\times$ 50 grid. Figures 5(a) and 5(b) use the additional argument `col.regions = sequential_hcl(51, power = 2.2)` (from the `colorspace` package) to change the default color palettes. Different panels of Figure 5 highlight the usage of `surf_check` and `response` for obtaining a level plot and surface plot of $\hat{y}(x)$ and $s^2(x)$.

## 5. Comparison with other packages

In the last two decades, a few different software programs (e.g., for R, MATLAB, C, C++, Python, etc.) have been produced for fitting GP models in computer experiments. The Gaussian process website (Rasmussen 2011) presents an extensive (though incomplete) list of such programs. Since R is a free software environment, packages like `tgp` and `mlegp` have gained popularity among the practitioners in computer experiments.

The `tgp` package (Gramacy 2007; Gramacy and Lee 2008), originally developed for building surrogates of both stationary and non-stationary noisy simulators, uses a GP model for emulating the stationary components of the process. The GP model here includes a nugget parameter that is estimated along with other parameters. The recent version of the `tgp` package facilitates the emulation of deterministic simulators by removing the nugget parameter from the model. Most importantly, `tgp` is implemented using Bayesian techniques like the Metropolis-Hastings algorithm, whereas, `GPfit` follows the maximum likelihood approach for fitting GP models and includes the smallest possible nugget required for computational stability.

Dancik and Dorman (2008) developed an R package called `mlegp` that uses maximum likelihood for fitting the GP model with Gaussian correlation structure. Though not relevant for

<table>
<thead>
<tr>
<th>xnew.1</th>
<th>xnew.2</th>
<th>Y_hat</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2002145</td>
<td>0.2732849</td>
<td>561.3877</td>
<td>186119713</td>
</tr>
<tr>
<td>0.6852186</td>
<td>0.4905132</td>
<td>-372.5221</td>
<td>21523832</td>
</tr>
<tr>
<td>0.9168758</td>
<td>0.3184040</td>
<td>13287.0495</td>
<td>86391757</td>
</tr>
<tr>
<td>0.2843995</td>
<td>0.5591728</td>
<td>3148.5904</td>
<td>8022989</td>
</tr>
<tr>
<td>0.1046501</td>
<td>0.2625931</td>
<td>5129.1136</td>
<td>562589770</td>
</tr>
<tr>
<td>0.7010575</td>
<td>0.2018752</td>
<td>8188.2805</td>
<td>13698589</td>
</tr>
<tr>
<td>0.5279600</td>
<td>0.3875257</td>
<td>3626.4985</td>
<td>123121468</td>
</tr>
<tr>
<td>0.8079352</td>
<td>0.8878698</td>
<td>14925.8142</td>
<td>1167409027</td>
</tr>
<tr>
<td>0.9565001</td>
<td>0.5549226</td>
<td>2869.6225</td>
<td>1483924477</td>
</tr>
<tr>
<td>0.1104530</td>
<td>0.8421794</td>
<td>217039.3229</td>
<td>264176788</td>
</tr>
</tbody>
</table>
this paper, \texttt{mlegp} can fit GP models with multivariate response, non-constant mean function and non-constant variance that can be specified exactly or up to a multiplicative constant. The simple GP model in \texttt{mlegp} is the same as described in Section 2.1 except that the nugget parameter is estimated along with other hyper-parameters. Hence, we use \texttt{mlegp} for the performance comparison of \texttt{GPfit}.

We now use several test functions to compare the performance of the two packages \texttt{mlegp} and \texttt{GPfit}. The test functions used here are commonly used in computer experiments for comparing competing methodologies (Santner et al. 2003). Since the two packages minimize slightly different deviance functions, one cannot directly compare the parameter estimates or the minimized deviance. Consequently, we compared the discrepancy between the predicted and the true simulator response. The performance measure is the standardized/scaled root mean squared error (sRMSE) given by

$$1 \frac{y_{\max} - y_{\min}}{\sqrt{\frac{1}{N} \sum_{i=1}^{N} \left[\hat{y}(x_i^*) - y(x_i^*)\right]^2}},$$

where $y_{\max}$ and $y_{\min}$ are the global maximum and minimum of the true simulator, $y(x_i^*)$ and $\hat{y}(x_i^*)$ are the true and predicted simulator output at $x_i^*$ in the test data, and $N$ is the size of the test data set. The results are averaged over 50 simulations. Each simulation starts with choosing two random $n \times d$ maximin LHDs ($D_0$ and $D_1$) for the training data and test data respectively (i.e., $N = n$). The average and standard error of the sRMSE values of the GP fits obtained from \texttt{mlegp} and \texttt{GPfit} are compared for several design sizes.

We found that \texttt{mlegp} occasionally crashes due to near-singularity of the spatial correlation matrix. In \texttt{mlegp}, the nugget parameter in $R_\delta = R + \delta I$ is estimated using the maximum likelihood procedure along with the other model parameters. If any candidate $\delta \in (0, 1)$ in the optimization procedure is not large enough to overcome the ill-conditioning of $R_\delta$, the likelihood computation fails and the \texttt{mlegp} package crashes with the following error message:

```
Error in solve.default(gp$invVarMatrix):
  system is computationally singular: reciprocal condition number = 2.11e-16.
```

This is not a problem in the \texttt{GPfit} implementation, because the nugget parameter is set at the smallest $\delta$ required to make $R_\delta$ well-conditioned. As a result, \texttt{GPfit} outperforms \texttt{mlegp} in terms of computational stability. When \texttt{mlegp} runs are computationally stable, \texttt{GPfit} usually has lower sRMSE values.

\textbf{Example 1 (contd.)} Suppose we wish to compare the prediction accuracy of the GP model fits from the two packages for the one dimensional test function in Example 1. Table 1 summarizes the sRMSE values and standard errors for a range of sample sizes. The results are based on 50 simulations.

It is clear from Table 1 that the sRMSE values decrease in both methods as $n$ increases. More importantly, \texttt{GPfit} significantly outperforms \texttt{mlegp}, especially, for larger $n$. This is expected as the numerical instability of the GP model increases with $n$. The smallest nugget $\delta_{lb}$ in the GP model of \texttt{GPfit} minimizes unnecessary over-smoothing giving a smaller sRMSE than \texttt{mlegp}, where $\hat{\delta}_{mle}$ might be relatively large to ensure computationally stable GP model fits (i.e., without any crashes).
GPfit: Gaussian Process Model Fitting in R

Table 1: The summary of sRMSE values (with standard errors) for the one dimensional simulator in Example 1.

<table>
<thead>
<tr>
<th>Sample size</th>
<th>GPfit</th>
<th>mlegp</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>30979.95 (5914.4)</td>
<td>31178.82 (6093.5)</td>
</tr>
<tr>
<td>n = 25</td>
<td>131.90 (16.160)</td>
<td>177.44 (24.724)</td>
</tr>
<tr>
<td>n = 50</td>
<td>26.74 (2.4935)</td>
<td>112.89 (17.749)</td>
</tr>
<tr>
<td>n = 75</td>
<td>18.14 (1.5203)</td>
<td>146.24 (23.989)</td>
</tr>
<tr>
<td>n = 100</td>
<td>12.99 (0.9882)</td>
<td>111.83 (17.089)</td>
</tr>
</tbody>
</table>

Table 2: The summary of sRMSE values (with standard errors) and the number of crashes for the GoldPrice function.

<table>
<thead>
<tr>
<th>Sample size</th>
<th>GPfit</th>
<th>mlegp</th>
<th>Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 25</td>
<td>415.00 (37.282)</td>
<td>483.3 (58.092)</td>
<td>0</td>
</tr>
<tr>
<td>n = 50</td>
<td>98.66 (6.8743)</td>
<td>129.2 (31.426)</td>
<td>0</td>
</tr>
<tr>
<td>n = 75</td>
<td>25.27 (1.6211)</td>
<td>19.2 (1.6546)</td>
<td>0</td>
</tr>
<tr>
<td>n = 100</td>
<td>12.84 (0.7839)</td>
<td>9081.5 (2633.4)</td>
<td>15</td>
</tr>
</tbody>
</table>

Example 2 (contd.) We now revisit the two-dimensional GoldPrice function illustrated in Example 2. Table 2 presents the averages and standard errors of sRMSE values for GP model fits obtained from mlegp and GPfit.

It is important to note that mlegp crashed 15 times out of 50 simulations for the $n = 100$ case. The summary statistics for $n = 100$ case in the mlegp column are calculated from the remaining 35 successful runs. The average and standard error of the sRMSE values in the successful runs of mlegp generate unreliable predictions. For the remaining cases, the results show that the sRMSE values decrease in both methods as $n$ increases. For $n = 25$ and 50, GPfit produces better GP fits with smaller sRMSE values. Interestingly, for $n = 75$, the average sRMSE value in GPfit is slightly larger as compared to that in mlegp.

Example 3 Suppose the four-dimensional Colville function is used as the computer simulator. Let $x = (x_1, x_2, x_3, x_4) \in [-10, 10]^4$, and the outputs be generated from

$$y(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1).$$

For implementation purpose, the inputs are rescaled to the unit-hypercube $[0, 1]^4$. Table 3 summarizes the averages and standard errors of the sRMSE values from 50 simulations.

Similar to Example 2, a few runs from mlegp crashed due to near-singularity, and the successful runs in these cases ($n = 50, 75$ and 100) yield unreliable summary statistics (i.e., unrealistically large sRMSE values). In contrast, GPfit provides stable and good predictions. Similar to Examples 1 and 2, the average sRMSE values decrease as $n$ increases.

It is worth noting that for the $n = 100$ case in this example, mlegp crashed 8 times in 50 simulations, whereas for the GoldPrice function example (Table 2), mlegp crashed 14 times.
Table 3: The summary of sRMSE values (with standard errors) and the number of crashes for Colville function.

<table>
<thead>
<tr>
<th>Sample size</th>
<th>GPfit sRMSE ($\times 10^{-4}$)</th>
<th>mlegp sRMSE ($\times 10^{-4}$)</th>
<th>Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 25</td>
<td>759.07 (27.72)</td>
<td>783.7 (29.41267)</td>
<td>0</td>
</tr>
<tr>
<td>n = 50</td>
<td>78.02 (4.046)</td>
<td>348851.2 (85861.61)</td>
<td>5</td>
</tr>
<tr>
<td>n = 75</td>
<td>41.88 (1.358)</td>
<td>235887.0 (32320.62)</td>
<td>14</td>
</tr>
<tr>
<td>n = 100</td>
<td>26.59 (1.245)</td>
<td>326456.8 (132317.6)</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 4: The summary of sRMSE values (with standard errors) for the six-dimensional Hartmann function.

<table>
<thead>
<tr>
<th>Sample size</th>
<th>GPfit sRMSE ($\times 10^{-3}$)</th>
<th>mlegp sRMSE ($\times 10^{-3}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 25</td>
<td>135.23 (5.546)</td>
<td>125.09 (5.022)</td>
</tr>
<tr>
<td>n = 50</td>
<td>102.10 (4.605)</td>
<td>94.89 (4.322)</td>
</tr>
<tr>
<td>n = 75</td>
<td>78.04 (2.573)</td>
<td>77.34 (2.546)</td>
</tr>
<tr>
<td>n = 100</td>
<td>66.87 (1.728)</td>
<td>66.87 (1.728)</td>
</tr>
<tr>
<td>n = 125</td>
<td>60.23 (1.415)</td>
<td>60.21 (1.416)</td>
</tr>
</tbody>
</table>

Though the number of simulations considered here is not large enough to accurately estimate the proportion of crashes in each case, it is expected that the occurrence of near-singular cases becomes less frequent with the increase in the input dimension (see Ranjan et al. 2011 for more details).

**Example 4** Consider the six-dimensional Hartmann function for generating simulator outputs. Since the input dimension is reasonably large, all mlegp runs turned out to be successful, and both packages lead to similar model predictions. Table 4 presents the averages and standard errors of the sRMSE values.

Overall in Examples 1 to 4, mlegp crashed only 42 times out of 900 simulations. However, the successful runs in the cases with any crash ($n = 100$ in Example 2 and $n = 50, 75$ and 100 in Example 3) lead to unreliable model fits. Furthermore, GPfit either outperforms or gives comparable GP model fits as compared to mlegp.

### 6. Real application

The Bay of Fundy (Figure 6(a)), located between New Brunswick and Nova Scotia, Canada, with a small portion touching Maine, USA, is world famous for its high tides. In Minas Passage (the upper portion of the Bay of Fundy), the difference in water level between high tide and low tide can be as much as 17 meters. Karsten, McMillan, Lickley, and Haynes (2008) (hereafter KMLH) considered a version of finite volume community ocean model (FVCOM) in the Minas Passage of the Bay of Fundy for simulating the average kinetic energy of the tidal flow (which in turn can be used for producing electricity). Figure 6(b) depicts the simulator output on a relatively coarse grid ($13 \times 41$) in Minas Passage.

According to KMLH, an individual tidal turbine can generate up to 1MW of power, and
Figure 6: Figure (a) shows the Bay of Fundy with the triangular grid used in the FVCOM model for simulating tides. The small box in the center surrounds the Minas Passage shown in (b). Figure (b) depicts KMLH simulator outputs in the Minas Passage.

approximately 2.5GW of power can be harnessed from the tidal kinetic energy by placing large collections of turbines in the Minas Passage. Optimal locations of such turbines can efficiently generate the much needed green energy at the minimal cost. One could use an expected improvement based sequential design scheme (Jones, Schonlau, and Welch 1998) for finding these optimal locations; the key component in such a sequential optimization is to efficiently emulate (i.e., fit a GP model to) the simulator response after every iteration of this sequential procedure. In this paper, we focus on this first step of fitting a GP model-based surrogate to the simulator output.

For fitting an emulator, we choose a 30-point space-filling design from the grid of 13 × 41 points using cover.design() in the R package fields (Nychka, Furrer, and Sain 2014), and evaluated the goodness-of-fit criterion sRMSE based on the remaining points from the 13 × 41 grid. In general, one can use a test/validation dataset to compute sRMSE values for goodness-of-fit comparison. Figures 7(a) and 7(b) show the level/contour plots of the fitted surface obtained via GPfit and mlegp respectively. Perhaps, the two fits cannot easily be ranked based on these contour plots, however, GPfit yields slightly smaller sRMSE (0.05493) than mlegp (0.06813).

By further increasing the design size to 50 from the same 13 × 41 grid, the fitted emulators obtained from GPfit and mlegp are reasonably comparable. Contour plots of the fitted surfaces (Figure 8) are visually indistinguishable. The sRMSE values computed using the remaining grid points are 0.05631008 and 0.05631197 for GPfit and mlegp respectively. According to Loepky et al. (2009), n = 10d-point space-filling design should be sufficient for building a good overall emulator for a reasonably smooth simulator. However, if the underlying simulator is more active, a larger design might be needed for building a decent overall emulator. Since the sample sizes used here are reasonably small near-singularity is not expected and the correlation matrices should be well-behaved, i.e., $\delta_{\text{mle}}(\hat{\theta}_{\text{mle}})$ (in GPfit) and $\delta_{\text{mle}}$ (in mlegp) should be zero. Subsequently, it is expected that both methods should result in comparable fits for this application.
Figure 7: GP emulator based on a 30-point design from the $13 \times 41$ grid shown in Figure 6(b). Figures (a) and (b) show the emulator obtained using \texttt{GPfit} and \texttt{mlegp} respectively.

Figure 8: GP emulator based on a 50-point design from the $13 \times 41$ grid shown in Figure 6(b). Figures (a) and (b) show the emulator obtained using \texttt{GPfit} and \texttt{mlegp} respectively.

7. Concluding remarks

This paper presents the R package \texttt{GPfit} for fitting GP models to scalar valued deterministic simulators, which is available from the Comprehensive R Archive Network (CRAN) at \url{http://CRAN.R-project.org/package=GPfit}. \texttt{GPfit} implements a slightly modified version of the GP model proposed by Ranjan \textit{et al.} (2011), which uses the new \(\beta\)-parameterization given in (4) of the spatial correlation function to facilitate optimization. The deviance optimization is achieved through a multi-start L-BFGS-B algorithm.

The proposed optimization algorithms makes $200d + \sum_{i=1}^{2d+1} \eta_i + \sum_{j=1}^{3} \eta'_j$ calls of the deviance function, whereas the GA implemented by Ranjan \textit{et al.} (2011) uses $1000d^2$ deviance eval-
uations. Though $\eta_i$ and $\eta'_j$ are non-deterministic, and vary with the complexity and input dimension of the deviance surface, for the simulations in Section 5 $\eta'_j \approx 30$ for all examples, however, the average $\eta_i$ are approximately 40, 75, 300 and interestingly 150 for Examples 1, 2, 3 and 4 respectively. Of course, neither of the two implementations have been optimally tuned for the most efficient deviance optimization. The best choice of options will of course vary from problem to problem, and so we encourage users to experiment with the available options.

The mlegp package is written in pre-compiled C code, whereas GPfit is implemented solely in R. This difference in the programming environment makes mlegp substantially faster than GPfit. The plots and results obtained in this paper used GPfit version 0.1-0, which allowed only Gaussian correlation. The current version of GPfit (version 0.2-0, same as version 1.0-0) is more flexible and stable. For instance, one can now specify the correlation structure as Matérn or power exponential with different power (the default is power exponential with power=1.95). Note that all results presented here can be produced in GPfit version 0.2-0 by using GP_fit(x,y,corr=list(type="exponential",power=2)) (see the stand alone R-script). The default values of a few plotting characters like line_type, cex and pch have also been updated for better visuals.

Acknowledgments

We thank the editorial board for many useful comments and suggestions that led to significant improvement of the article. This work of was supported in part by Discovery grants (for Ranjan and Chipman) and undergraduate student research award (for MacDonald) from the Natural Sciences and Engineering Research Council of Canada.

References


Affiliation:
Pritam Ranjan, Hugh Chipman
Department of Mathematics and Statistics
Acadia University

15 University Avenue
Wolfville, NS, Canada
E-mail: pritam.ranjan@acadiau.ca, hugh.chipman@acadiau.ca
URL: http://acadiau.ca/~pranjan/, http://math.acadiau.ca/chipmanh/