GPfit: An R package for Gaussian Process Model Fitting using a New Optimization Algorithm

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), as the new R package GPfit. A novel parameterization of the spatial correlation function and a new multi-start gradient based optimization algorithm yield optimization that is robust and 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 a popular R package mlegp. GPfit is a free software and distributed under the general public license, as part of the R software project (R Development Core Team 2012).

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), and dynamics of dark energy and dark matter in cosmological studies (Arbey 2006). 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 a new R package **GPfit** 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 δ 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, Jr., 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 δ that maintains the delicate balance between the stabilization and minimizing the over-smoothing of the model predictions. Ranjan *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.

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 et al. (2005) demonstrate the preference of deterministic traffic simulators over their stochastic counterparts. The model considered in **GPfit** assumes that the computer simulator is deterministic and is very similar to the GP model proposed in Ranjan 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, Filipič, and Kocijan 2011). This makes the model fitting procedure computationally challenging. Ranjan et al. (2011) uses a genetic algorithm (GA) approach, which is robust but computationally intensive for likelihood optimization. **GPfit** uses a multi-start gradient based search algorithm that is robust and typically faster than the GA used in Ranjan et al. (2011). A clustering based approach on a large space-filling design over the parameter space is used for choosing the initial values of the gradient search. Furthermore, we proposed a new parameterization of the spatial correlation function for the ease of likelihood optimization.

The remainder of the paper is organized as follows. Section 2 presents a brief review of the GP model in Ranjan *et al.* (2011), the new parameterization of the correlation function and the new optimization algorithm implemented in **GPfit**. In Section 3, the main functions of **GPfit** and their arguments are discussed. Two examples illustrating the usage of **GPfit** are presented in Section 4. Section 5 compares **GPfit** with other popular R packages. This includes an empirical performance comparison with the popular R package **mlegp**. The paper concludes with a few remarks in Section 6.

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 new 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, $x_i = (x_{i1}, ..., x_{id})'$ and $y_i = y(x_i)$ respectively. The experimental design $D_0 = \{x_1, ..., x_n\}$ 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 $Y = y(X) = (y_1, ..., y_n)'$. The simulator output, $y(x_i)$, is modeled as

$$y(x_i) = \mu + z(x_i);$$
 $i = 1, ..., n,$

where μ is the overall mean, and $z(x_i)$ is a GP with $E(z(x_i)) = 0$, $Var(z(x_i)) = \sigma^2$, and $Cov(z(x_i), z(x_j)) = \sigma^2 R_{ij}$. In general, y(X) has a multivariate normal distribution, $N_n(\mathbf{1_n}\mu, \Sigma)$, where $\Sigma = \sigma^2 R$ is formed with correlation matrix R having elements R_{ij} , and $\mathbf{1_n}$ 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

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

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

$$\hat{\mu}(\theta) = (\mathbf{1_n}'R^{-1}\mathbf{1_n})^{-1}(\mathbf{1_n}'R^{-1}Y) \text{ and } \hat{\sigma}^2(\theta) = \frac{(Y - \mathbf{1_n}\hat{\mu}(\theta))'R^{-1}(Y - \mathbf{1_n}\hat{\mu}(\theta))}{n},$$

are used to obtain the negative profile log-likelihood (hereonwards, referred to as deviance)

$$-2\log(L_{\theta}) \propto \log(|R|) + n\log[(Y - \mathbf{1_n}\hat{\mu}(\theta))'R^{-1}(Y - \mathbf{1_n}\hat{\mu}(\theta))],$$

for estimating the hyper-parameters θ , 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

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

with mean squared error

$$\begin{split} s^2(x^*) &= E\left[(\hat{y}(x^*) - y(x^*))^2\right] \\ &= \sigma^2(1 - 2C'r + C'RC) = \sigma^2\left(1 - r'R^{-1}r + \frac{(1 - \mathbf{1_n}'R^{-1}r)^2}{\mathbf{1_n}R^{-1}\mathbf{1_n}}\right), \end{split}$$

where $r = (r_1(x^*), ..., r_n(x^*))$, and $r_i(x^*) = corr(z(x^*), z(x_i))$. In practice, the parameters μ , σ^2 and θ 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_{δ} 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 computationally stable approach to choosing the nugget parameter δ . They introduced a lower bound on δ that minimizes the unnecessary over-smoothing. The lower bound given by Ranjan *et al.* (2011) is

$$\delta_{lb} = \max \left\{ \frac{\lambda_n(\kappa(R) - e^a)}{\kappa(R)(e^a - 1)}, 0 \right\},\tag{2}$$

where λ_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).

GPfit uses the GP model with $R_{\delta_{lb}} = R + \delta_{lb}I$. The R package **mlegp**, used for performance comparison of **GPfit** in Section 5, implements the classical GP model with R replaced by $R_{\delta} = R + \delta I$, and estimates δ 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

$$-2\log(L_{\theta}) \propto \log(|R_{\delta_{lb}}|) + n\log[(Y - \mathbf{1}_{\mathbf{n}}\hat{\mu}(\theta))'R_{\delta_{lb}}^{-1}(Y - \mathbf{1}_{\mathbf{n}}\hat{\mu}(\theta))]. \tag{3}$$

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 stating points, however, such designs (e.g., maximin LHD) often tend to stay away from the boundaries and corners. This is unfavourable because the deviance functions (Figure 1) are very active near $\theta = 0$.

To address the issue of a bumpy deviance surface near the boundaries of the parameter space,

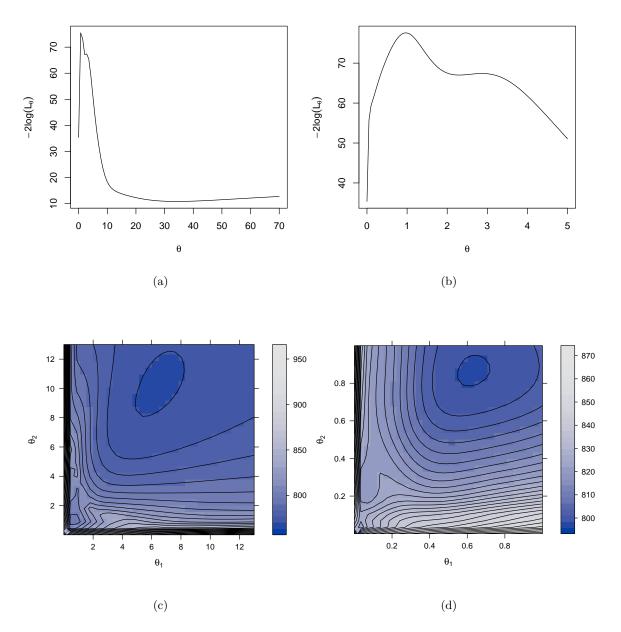


Figure 1: The plots show deviance (3) w.r.t. the GP parameter(s) θ . Panels (a) and (b) correspond to Example 1 (with d = 1, n = 10), and (c) and (d) display deviance for Example 2 (with d = 2, n = 30). Panels (b) and (d) are enlargements of (a) and (b) near 0, respectively.

we propose a new parameterization of R. Let $\beta_k = \log_{10}(\theta_k)$ for k = 1, ..., d, then

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

where a small value of β_k implies a very high spatial correlation or a relatively flat surface in the k-th coordinate, and the large values of β_k imply low correlation, or a very wiggly surface

with respect to the k-th input factor. Figure 2 displays the two deviance surfaces (shown in Figure 1) under the β - parameterization of R (4). Though the new parameterization of R (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.

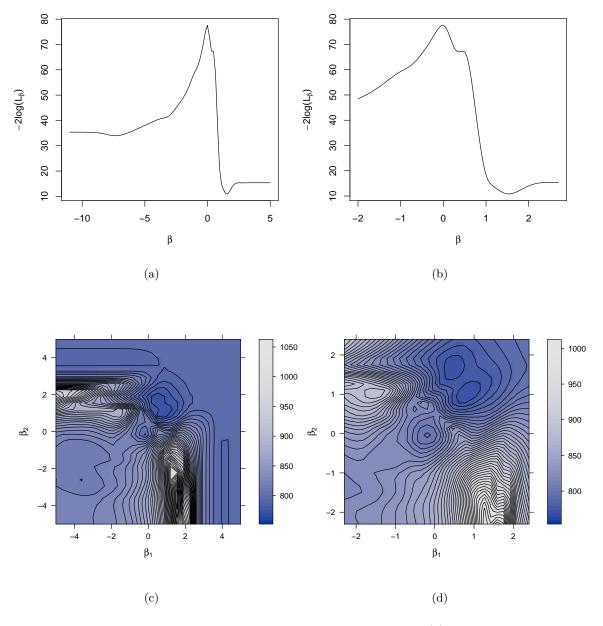


Figure 2: The plots show deviance under β parameterization of R (4), for the same examples and data as in Figure 1. Panels (a) and (b) correspond to Example 1 (with d=1, n=10), and (c) and (d) display deviance for Example 2 (with d=2, n=30). Panels (b) and (d) are enlargements of (a) and (b) near 0, respectively.

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 β that minimizes 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, the prediction accuracy at unsampled locations may suffer from suboptimal parameter estimates. In an attempt to obtain a good fit of the GP model, **GPfit** uses a multi-start L-BFGS-B algorithm for optimizing the deviance $-2 \log(L_{\beta})$. We first find a subregion Ω_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 Ω_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:

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

or equivalently,

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

To convert the bounds above into workable ranges for the β_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 k-th coordinate is $|x_{ik} - x_{jk}| \approx 1/10$. Furthermore, if we also make a simplifying assumption that the simulator is equally smooth in all directions, i.e., $\beta_k = \beta_0$, then the inequality simplifies to

$$-2 - \log_{10}(d) \le \beta_k \le \log_{10}(500) - \log_{10}(d). \tag{5}$$

That is, $\Omega_0 = \{(\beta_1, ..., \beta_d) : -2 - \log_{10}(d) \le \beta_k \le \log_{10}(500) - \log_{10}(d), k = 1, ..., d\}$ is the set of $\beta = (\beta_1, ..., \beta_d)$ values that is likely to contain the likelihood optimizer. We use Ω_0 for restricting the initial values of 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 Ω_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 Ω_0 assumed the simplification $\beta_k = \beta_0$ for all k, and in some cases, for instance, in Figure 2(d), 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 Ω_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 200*d*-point maximin LHD for $\beta = (\beta_1, ..., \beta_d)$ in the hyper-rectangle Ω_0 .
- 2. Choose the 80d values of β that correspond to the smallest $-2\log(L_{\beta})$ values.
- 3. Use k-means clustering algorithm on these 80d points to find 2d groups. To improve the quality of the clusters, five random restarts of k-means are used.
- 4. For $d \geq 2$, run L-BFGS-B algorithm along the main diagonal of Ω_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 $\left(200d + \sum_{i=1}^{2d+1} \eta_i + \sum_{j=1}^{3} \eta_j'\right)$ deviance evaluations, where η_i is the number of deviance evaluations for the *i*-th L-BFGS-B run in Ω space, and η_j' is the number of deviance evaluations for the *j*-th L-BFGS-B run along the diagonal of the Ω_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, η_i and η_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 different functions of **GPfit** that implements our proposed model, which is the computationally stable version of the GP model proposed by Ranjan $et\ al.$ (2011) with the new parameterization of 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 $d \le 2$) plot(). Both predict() and plot() use GP_{fit} () class objects 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:

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.

- control: A vector of three tunable parameters used in the deviance optimization algorithm. The default values correspond to choosing 2*d clusters (using k-means clustering algorithm) based on 80*d best points (smallest deviance) from a 200*d point random maximin LHD in Ω₀.
- nug_thres: A threshold parameter used in the calculation of the lower bound of the nugget, δ_{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: is the maximum number of iterations per L-BFGS-B run in the deviance optimization. We use the optim package 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 $\delta_{lb}(\hat{\beta})$. Assuming GPmodel is the GP class object, print(GPmodel,...) presents the values of the object GPmodel, and options like digits can be used for "...". As an alternative, one can use summary(GPmodel) to get the same output.

If xnew contains the set of unobserved inputs, `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. The expressions of $\hat{y}(x^*)$ 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 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, ...)
```

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 "..." for changing the plots (see `par' in the base R function `plot()').

For d=2, the default arguments of plot() with GP object produces a level plot of $\hat{y}(x^*)$. The plots are based on the model prediction 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 wireframe() function, contour and cuts from levelplot() function in lattice (Sarkar 2008), and color specific arguments in colorspace (??) can also be passed in for "...".

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 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 maximinLHS function in the R package lhs (Carnell 2009; Stein 1987). The following R code shows how to load the packages, generate the simulator outputs and then fit the GP model using GP_fit().

```
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)
```

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 print(GPmodel). 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

sigma^2_hat: [1]    0.7444
delta_lb(beta_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).

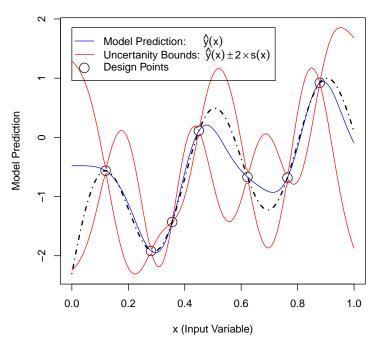


Figure 3: The plot shows the model predictions and uncertainty bands for Example 1. The true simulator output curve is also displayed by the dash-dotted line.

Example 2 We now consider a two-dimensional test function to illustrate different functions of **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)

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

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 = 20Input Dimensions: d = 2

Correlation: Exponential (power = 2)
Correlation Parameters:

beta_hat.1 beta_hat.2
[1] 0.8578 1.442

sigma^2_hat: [1] 4.52e+09
delta_lb(beta_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)

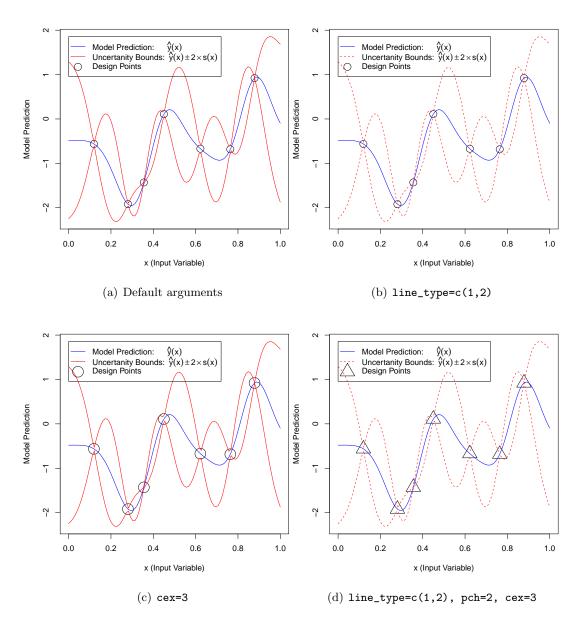


Figure 4: The plots illustrate the usage of graphical parameters in plot() for Example 1. Panel (a) shows the model prediction and uncertainty plot with default graphical parameters, (b) illustrates the change due to line_type, (c) highlights the point size using cex, and (d) shows the usage of pch in changing the point character.

would have required 4000 deviance calls. The correlation hyper-parameter estimate $\hat{\beta}_{mle} = (0.8578, 1.442)$ shows that the fitted simulator is slightly more active (or wiggly) in the X_2 variable. The nugget parameter $\delta_{lb}(\hat{\beta}_{mle}) = 0$ implies that the correlation matrix with the chosen design points and $\beta = \hat{\beta}_{mle}$ 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 predict object Model_pred are as follows:

```
$Y_hat
 [1]
        561.3877
                    -372.5221
                               13287.0495
                                             3148.5904
                                                          5129.1136
 [6]
       8188.2805
                    3626.4985
                               14925.8142
                                             2869.6225 217039.3229
$MSE
                   21523832
                              86391757
 [1]
      186119713
                                           8022989
                                                     562589770
 [6]
                 123121468 1167409027 1483924477
       13698589
                                                     264176788
$complete_data
         xnew.1
                    xnew.2
                                 Y_hat
                                               MSE
 [1,] 0.2002145 0.2732849
                              561.3877
                                         186119713
 [2,] 0.6852186 0.4905132
                             -372.5221
                                          21523832
 [3,] 0.9168758 0.3184040
                            13287.0495
                                          86391757
 [4,] 0.2843995 0.5591728
                             3148.5904
                                           8022989
 [5,] 0.1046501 0.2625931
                             5129.1136
                                         562589770
 [6,] 0.7010575 0.2018752
                             8188.2805
                                          13698589
 [7,] 0.5279600 0.3875257
                             3626.4985
                                         123121468
 [8,] 0.8079352 0.8878698
                            14925.8142 1167409027
 [9,] 0.9565001 0.5549226
                             2869.6225 1483924477
[10,] 0.1104530 0.8421794 217039.3229
                                         264176788
```

The **GPfit** function plot() calls predict() for computing $\hat{y}(x^*)$ and $s^2(x^*)$ at a regular resolution x 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.

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 50×50 grid. Figures 5(a) and 5(b) used additional argument 'col.regions=sequential_hcl(51, power=2.2)' (from 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 programs (in R, Matlab, C, C++, Python, and so on) 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.

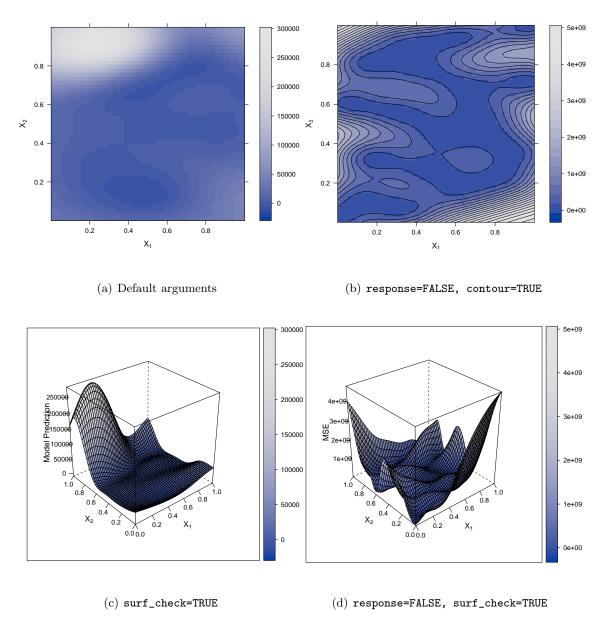


Figure 5: The plots illustrate the usage of graphical parameters in plot() for Example 2. Panel (a) shows the default plot (the levelplot of $\hat{y}(x^*)$) with additional color specification, (b) presents levelplot with contour lines of $s^2(x^*)$, (c) shows the surface plot of $\hat{y}(x^*)$, and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

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 stochastic (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 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 this paper, **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 **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 **mlegp** for the performance comparison of **GPfit**.

We now use several test functions to compare the performance of the two packages **mlegp** and **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

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

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 $(D_0 \text{ and } D_1)$ 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 **mlegp** and **GPfit** are compared for several design sizes.

We found that **mlegp** occasionally crashes due to near-singularity of the spatial correlation matrix in the GP model. In **mlegp**, the nugget parameter in $R_{\delta} = R + \delta I$ is estimated using 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_{δ} , the likelihood computation fails and the **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 **GPfit** implementation, because the nugget parameter is set at the smallest δ required to make R_{δ} well-conditioned. As a result, **GPfit** outperforms **mlegp** in terms of computational stability. Whenever **mlegp** runs are computationally stable, then also **GPfit** appears to have lower sRMSE values in most cases.

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 for a range of sample sizes in the format: average (standard error). The results are based on 50 simulations.

	GPfit	mlegp
Sample size	sRMSE (×10 ⁻⁶)	sRMSE ($\times 10^{-6}$)
n = 10	32958 (4948.8)	37282 (7153.4)
n=25	139.21 (13.768)	158.07 (15.662)
n = 50	$28.81 \ (2.5977)$	113.49 (16.139)
n = 75	18.29 (1.6297)	105.84 (16.251)
n = 100	12.36 (0.7320)	101.25 (14.254)

Table 1: The summary of sRMSE values for the one dimensional simulator in Example 1.

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

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**.

	GPfit	mlegp	
Sample size	sRMSE ($\times 10^{-4}$)	sRMSE ($\times 10^{-4}$)	Crashes
n=25	381.23 (43.85)	424.07 (56.92)	0
n = 50	88.120 (8.114)	105.95 (18.93)	0
n = 75	23.282 (1.499)	17.379 (2.271)	0
n = 100	12.747 (0.875)	1601.5 (188.6)	14

Table 2: The summary of sRMSE values and the number of crashes for GoldPrice function.

It is important to note that the **mlegp** crashed 14 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 26 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

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

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.

Sample size	GPfit	mlegp	
Dample Size	sRMSE ($\times 10^{-6}$)	sRMSE ($\times 10^{-6}$)	Crashes
n=25	103.3 (5.401)	109.58 (6.120)	0
n = 50	11.77 (0.771)	10334 (3344)	2
n = 75	7.169 (0.472)	3251 (1109)	5
n = 100	5.786 (1.839)	63.10 (25.39)	1

Table 3: The summary of sRMSE values and the number of crashes for Colville function.

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 only once in 50 simulations, whereas for the GoldPrice function example (Table 2), **mlegp** crashed 14 times. 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 the packages lead to similar model predictions. Table 4 presents the averages and standard errors of the sRMSE values.

Sample size	GPfit	mlegp	
	sRMSE ($\times 10^{-3}$)	sRMSE ($\times 10^{-3}$)	
n=25	118.44 (4.837)	116.64 (4.655)	
n = 50	105.24 (4.649)	105.56 (4.500)	
n = 75	82.587 (2.536)	84.819 (3.090)	
n = 100	75.169 (2.645)	75.402 (2.738)	
n = 125	63.014 (1.652)	63.223 (1.653)	

Table 4: The summary of sRMSE values for the six-dimensional Hartmann function.

Overall in Examples 1 to 4, **mlegp** crashed only 22 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. Concluding remarks

This paper presents a new R package **GPfit** for fitting GP models to scalar valued deterministic simulators. **GPfit** implements a slightly modified version of the GP model proposed by Ranjan

et al. (2011), which uses the new β parameterization (4) of the spatial correlation function for the ease of 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 et~al.~(2011) uses $1000d^2$ deviance evaluations. Though η_i and η'_j are non-deterministic, and vary with the complexity and input dimension of the deviance surface, the simulations in Section 5 show that $\eta'_j \approx 30$ for all examples, however, the average η_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 current version of **GPfit** package uses only Gaussian correlation. We intend to include other popular correlation functions like Matérn in our R package.

References

- Andre J, Siarry P, Dognon T (2000). "An Improvement of the Standard Genetic Algorithm Fighting Premature Convergence." Advances in Engineering Software, 32, 49–60.
- Arbey A (2006). "Dark fluid: A Complex Scalar Field to Unify Dark Energy and Dark Matter." Phys. Rev. D, 74, 043516. doi:10.1103/PhysRevD.74.043516.
- Booker AJ, Jr JED, Frank PD, Serafini DB, Torczon V, Trosset MW (1999). "A Rigorous Framework for Optimization of Expensive Functions by Surrogates." *Structural and Multidisciplinary Optimization*, **17**, 1–13.
- Byrd RH, Lu P, Nocedal J, Zhu C (1995). "A Limited Memory Algorithm for Bound Constrained Optimization." SIAM Journal of Scientific Computing, 16, 1190–1208.
- Carnell R (2009). *lhs: Latin Hypercube Samples*. R package version 0.5.
- Dancik GM, Dorman KS (2008). "mlegp: Statistical Analysis for Computer Models of Biological Systems using R." *Bioinformatics*, **24**, 1966–1967. R package version 3.1.2, URL http://CRAN.R-project.org/package=mlegp.
- 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.
- Gramacy RB, Lee HKH (2008). "Bayesian Treed Gaussian Process Models with an Application to Computer Modeling." *Journal of the American Statistical Association*, **103**(483), 1119–1130.
- Gramacy RB, Lee HKH (2012). "Cases for the Nugget in Modeling Computer Experiments." Statistics and Computing, 22(3), 713–722.

- Kalaitzis AA, Lawrence ND (2011). "A Simple Approach to Ranking Differentially Expressed Gene Expression Time Courses through Gaussian Process Regression." *BMC Bioinformatics*, **12**, 180.
- Loeppky JL, Sacks J, Welch WJ (2009). "Choosing the Sample Size of a Computer Experiment: A Practical Guide." *Technometrics*, **51**(4), 366–376.
- McKay MD, Beckman RJ, Conover WJ (1979). "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code." *Technometrics*, **21**(2), 239–245.
- Medina JS, Moreno MG, Royo ER (2005). "Stochastic Vs Deterministic Traffic Simulator. Comparative Study for Its Use Within a Traffic Light Cycles Optimization Architecture." In *Proceedings of the IWINAC* (2), pp. 622–631. Berlin: Springer-Verlag.
- Neal RM (1997). "Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification." Tech Rep. 9702, Dept. of Statistics, Univ. of Toronto, Canada.
- Petelin D, Filipič B, Kocijan J (2011). "Optimization of Gaussian Process Models with Evolutionary Algorithms." In *Proceedings of the 10th international conference on Adaptive and natural computing algorithms Volume Part I*, ICANNGA'11, pp. 420–429. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-642-20281-0. URL http://dl.acm.org/citation.cfm?id=1997052.1997098.
- Poole D, Raftery AE (2000). "Inference for Deterministic Simulation Models: The Bayesian Melding Approach." *Journal of the American Statistical Association*, **95**(452), 1244–1255.
- Ranjan P, Haynes R, Karsten R (2011). "A Computationally Stable Approach to Gaussian Process Interpolation of Deterministic Computer Simulation Data." *Technometrics*, **53**(4), 366–378.
- Rasmussen CE (2011). "The Gaussian Process Website." URL http://www.gaussianprocess.org/.
- Rasmussen CE, Williams CKI (2006). Gaussian Processes for Machine Learning. The MIT Press.
- R Development Core Team (2012). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.
- Sacks J, Welch W, Mitchell T, Wynn H (1989). "Design and Analysis of Computer Experiments." Statistical Science, 4(4), 409–435.
- Santner TJ, Williams B, Notz W (2003). The Design and Analysis of Computer Experiments. Springer-Verlag, New York.
- Sarkar D (2008). *Lattice:* Multivariate Data Visualization with R. Springer-Verlag, New York. ISBN 978-0-387-75968-5, URL http://lmdvr.r-forge.r-project.org.

- Schirru A, Pampuri S, Nicolao GD, McLoone S (2011). "Efficient Marginal Likelihood Computation for Gaussian Processes and Kernel Ridge Regression." ArXiv:1110.6546v1.
- Stein M (1987). "Large Sample Properties of Simulations Using Latin Hypercube Sampling." *Technometrics*, **29**, 143–151.
- Yuan J, Wang K, Yu T, Fang M (2008). "Reliable Multi-objective Optimization of High-speed WEDM Process based on Gaussian Process Regression." International Journal of Machine Tools and Manufacture, 48(1), 47 60. ISSN 0890-6955. doi:10.1016/j.ijmachtools.2007.07.011. URL http://www.sciencedirect.com/science/article/pii/S0890695507001265.

Affiliation:

Pritam Ranjan Department of Mathematics and Statistics Acadia University 15 University Avenue, Wolfville, NS, Canada

E-mail: pritam.ranjan@acadiau.ca URL: http://acadiau.ca/~pranjan/

published by
Volume VV, Issue II
MMMMMM YYYY

http://www..org/ http://www..org/

Submitted: yyyy-mm-dd Accepted: yyyy-mm-dd