2.4 What does *mlegp* do?

The package *mlegp* extends the standard GP model of (3), which assumes that $N = \sigma_{\mathrm{e}}^2 I$, by allowing the user to specify the diagonal nugget matrix N exactly or up to a multiplicative constant (i.e., N_s). This extension provides some flexibility for modeling heteroscedastic responses. The user also has the option of fitting a GP with a constant mean (i.e., $\mu(\theta) \equiv \mu_0$) or mean functions that are linear regression functions in all elements of θ (plus an intercept term). For multi-dimensional output, the user has the option of fitting independent GPs to each dimension (i.e., each type of observation), or to the most important principle component weights following singular value decomposition. The latter is ideal for data rich situations, such as functional output, and is explained further in Section (5). GP accuracy is analyzed through diagnostic plots of cross-validated predictions and cross-validated residuals, which were described in Section (2.3). Sensitivity analysis tools including FANOVA decomposition, and plotting of main and two-way factor interactions are described in Section (4).

The package *mlegp* employs two general approaches to GP fitting. In the standard approach, *mlegp* uses numerical methods in conjunction with equations (8) and (9) to find maximum likelihood estimates (MLEs) of all GP parameters. However, when replicate runs are available, it is usually more accurate and computationallly more efficient to fit a GP to a collection of *sample means* while using a plug-in estimate for the nugget (matrix).

Let $z_{ij} \equiv z_j\left(\theta^{(i)}\right)$ be the j^{th} replicate output from the computer model evaluated at the input vector $\theta^{(i)}$, $i = 1, \ldots k, j = 1, \ldots n_i$, so that the computer model is evaluated n_i times at the input vector $\theta^{(i)}$. Let $\overline{z} = (\overline{z_{1.}}, \ldots \overline{z_{k.}})$ be a collection of k sample mean computer model outputs, where

$$\overline{z_{i.}} = \frac{1}{n_i} \sum_{j=1}^{n_i} z_{ij}$$

is the sample mean output when the computer model is evaluated at θ.

The GP model of $\overline{z}$ is similar to the GP model of z_{known} described above, with the $(i,t)^{th}$ element of the matrix $C(\beta)$ given by $\mathrm{cor}(\overline{z_{i.}}, \overline{z_{t.}})$, following Eq. (1). and the i^{th} element of the nugget matrix N given by $\frac{\sigma_{\mathrm{e}}^2(\theta)}{n_i}$. The covariance matrix V has the same form as Eq. (4). Predicted means and variances have the same form as Eqs. (5 - 6), but with the vector z_{known} replaced by $\overline{z}$. For a fixed nugget term or nugget matrix, the package *mlegp* can fit a GP to a set of sample means by using numerical methods in combination with Eq. (8) to find the MLE of all remaining GP parameters. The user may specify a value for the constant nugget or nugget matrix to use. Alternatively, if replicate runs are available and a nugget term is not specified, *mlegp* will automatically take $N = \sigma_{\mathrm{e}}^2 I$ and estimate the nugget as

$$\widehat{\sigma_e^2} = \frac{1}{N-k} \sum_{i=1}^{k} (n_i - 1) s_i^2,$$

where, s_i^2 is the sample variance for design point i and $N = \sum_{i=1}^{k} n_i$. This estimate is the best linear unbiased estimate (BLUE) of σ_e^2 (which is linear in s_i^2).

The above *means* approach is computationally more efficient when replicate runs are available. If the nugget term or nugget matrix is well known or can be accurately estimated, the *means* approach is also more accurate than the standard approach.

3 Examples: Gaussian process fitting and diagnostics

3.1 A simple example

The function *mlegp* is used to fit one or more Gaussian processes (GPs) to a vector or matrix of responses observed under the same set of inputs. Data can be input from within R or read from a text file using the command *read.table* (type '?read.table' from within R for more information).

The example below shows how to fit multiple GPs to multiple outputs $z1$ and $z2$ for the design matrix x. Diagnostic plots are obtained using the *plot* function, which graphs observed values vs. cross-validated predicted values for each GP. The plot obtained from the code below appears in Figure (1).

```
> x = -5:5
> z1 = 10 - 5 * x + rnorm(length(x))
> z2 = 7 * sin(x) + rnorm(length(x))
> fitMulti = mlegp(x, cbind(z1, z2))
> plot(fitMulti)
```

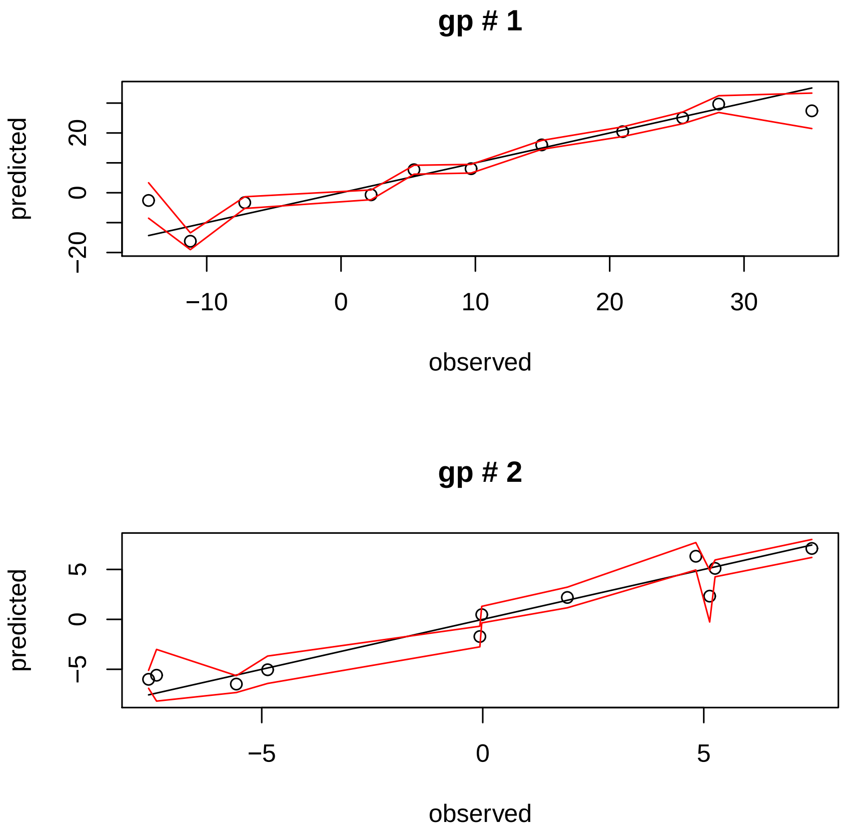

Figure 1: Gaussian process diagnostic plots. Open circles, cross-validated predictions; solid black lines, observed values; solid red lines, confidence bands corresponding to cross-validated predictions ± standard deviation.

After the GPs are fit, simply typing the name of the object (e.g., $fitMulti$) will return basic summary information.

```
> fitMulti

num GPs: 2
Total observations (per GP): 11
Dimensions: 1
```

We can also access individual Gaussian processes by specifying the index. The code below, for examples, displays summary information for the first Gaussian process, including diagnostic statistics of cross-validated root mean squared error (CV RMSE) and cross-validated root max squared error (CV RMaxSE), where squared error corresponds to the squared difference between cross-validated predictions and observed values.

```
> fitMulti[[1]]

Total observations = 11
Dimensions = 1

mu = 9.486534
sig2:        223.9967
nugget:        0

Correlation parameters:

      beta a
1 0.263693 2

Log likelihood = -37.19833

CV RMSE: 4.831457
CV RMaxSE: 137.8616
```

3.2 An example with replicate observations

When replicate observations are available, and the nugget term (or matrix) is known or can be accurately estimated, it is computationally more efficient and more accurate to use a plug-in estimate for the nugget term (or matrix) and to fit a GP to a set of sample means. This is done by setting 'nugget.known = 1' in the call to *mlegp*, while still passing in a vector or matrix of all observations. A nugget value can be specified exactly by setting the 'nugget' argument to the (estimated) value of σ_e^2 as in the code below.

```
> x = c(1:10, 1:10, 1:10)
> y = x + rnorm(length(x), sd = 1)
> fit = mlegp(x, y, nugget = 1, nugget.known = 1)
```

If the argument 'nugget' is not specified, a weighted average of sample variances will be used.

```
> fit = mlegp(x, y, nugget.known = 1)

> fit$nugget

[1] 1.572979
```

3.3 Heteroscedastic responses and the nugget matrix

In cases where the responses are heteroscedastic (have non-constant variance), it is possible to specify the diagonal nugget matrix exactly or up to a multiplicative constant. Currently, we recommend specifying the nugget matrix based on sample variances for replicate design points (which is easily obtained using the function *varPerReps*), based on the results of a separate statistical model, or based on prior information.

In the example below, we demonstrate how to fit a GP with a constant nugget term, a GP where the diagonal nugget matrix is specified up to a multiplicative constant, and a GP where the nugget matrix is specified exactly. First we generate heteroscedastic data, with variance related to the design parameter.

```
> x = seq(0, 1, length.out = 20)
> z = x + rnorm(length(x), sd = 0.1 * x)
```

By default, a nugget term is automatically estimated if there are replicates in the design matrix, and is not estimated otherwise. However, one can estimate a nugget term by specifying an initial scalar value for the 'nugget' argument during the call to *mlegp*. This is done in the code below.

```
> fit1 = mlegp(x, z, nugget = mean((0.1 * x)^2))
```

Alternatively, one can set 'nugget' equal N_s, which specifies the nugget matrix up to a multiplicative constant, and is demonstrated in the code below.

```
> fit2 = mlegp(x, z, nugget = (0.1 * x)^2)
```

Finally, we completely and *exactly* specify the nugget matrix N by also setting 'nugget.known = 1'.

```
> fit3 = mlegp(x, z, nugget.known = 1, nugget = (0.1 * x)^2)
```

We demonstrate the advantage of using a non-constant nugget term by comparing the root mean squared error (RMSE) between the true response and predictions from each fitted GP. Importantly, predictions are less accurate (have higher root mean squared errors) and can have far from nominal coverage probabilities when a constant nugget is incorrectly assumed.

```
> sqrt(mean((x - predict(fit1))^2))

[1] 0.05602172

> sqrt(mean((x - predict(fit2))^2))

[1] 0.05253949

> sqrt(mean((x - predict(fit3))^2))

[1] 0.05249836
```

4 Sensitivity Analysis

4.1 Background

For a response $y = f(x)$, where x can be multidimensional, sensitivity analysis (SA) is used to (a) quantify the extent in which uncertainty in the response y can be attributed to uncertainty in the design parameters x, and (b) characterize how the response changes as one or more design parameters are varied. General SA methods can be found in Saltelli *et al.* (2000). We briefly describe SA using Gaussian process models, which is described in Schonlau and Welch (2006).

For independent marginal priors on the components of θ, the total variance of the GP predictor can be decomposed into variance contributions from main and higher order interaction effects, a technique known as Functional Analysis of Variance (FANOVA) decomposition. The percentage of the total functional variance accounted for by a particular effect provides a measure of the importance of that effect.

The main effect of parameter θ_k, defined as $E[z(\theta)|z_{\text{known}}, \theta_k]$, predicts output for a fixed value of θ_k, averaged over the remaining parameters according to a prior (or weight function) $\pi(\theta_{\text{-}k})$ on all components of θ except for the k^{th}. The two-way interaction effect for parameters θ_k and θ_l, defined as $E[z(\theta)|z_{\text{known}}, \theta_k, \theta_l]$, predicts output for jointly fixed values of θ_k and θ_l, averaged over the remaining parameters according to a prior $\pi(\theta_{\text{-}k,\text{-}l})$. Main effects plots and contour plots conveniently illustrate main effects and two-factor interactions.

In *mlegp*, we implement FANOVA decomposition and the plotting of main and two-way factor interactions using indepedent uniform priors on all components of θ. By default, the range of each component is taken to be the range of that component in the design matrix, but these ranges can be overwritten via the arguments 'lower'and 'upper'.