mlegp: an R package for Gaussian process modeling and sensitivity analysis

Garrett Dancik

November 4, 2009

1 *mlegp*: an overview

Gaussian processes (GPs) are commonly used as surrogate statistical models for predicting output of computer experiments (Santner *et al.*, 2003). Generally, GPs are both interpolators and smoothers of data and are effective predictors when the response surface of interest is a smooth function of the parameter space. The package *mlegp* finds *m*aximum *l*ikelihood *e*stimates of *G*aussian *p*rocesses for univariate and multi-dimensional responses, for Gaussian processes with Gaussian correlation structures; constant or linear regression mean functions; and for responses with either constant or non-constant variance that can be specified exactly or up to a multiplicative constant. Unlike traditional GP models, GP models implemented in *mlegp* are appropriate for modelling heteroscedastic responses where variance is known or accurately estimated. Diagnostic plotting functions, and the sensitivity analysis tools of Functional Analysis of Variance (FANOVA) decomposition, and plotting of main and two-way factor interaction effects are implemented. Multi-dimensional output can be modelled by fitting independent GPs to each dimension of output, or to the most important principle component weights following singular value decomposition of the output. Plotting of main effects for functional output is also implemented. From within R, a complete list of functions and vignettes can be obtained by calling 'library(help = "mlegp")'.

2 Gaussian process modeling and diagnostics

2.1 Gaussian processes

Let $z_{\text{known}} = \left[\text{z}(\theta^{(1)}), \ldots, \text{z}(\theta^{(m)})\right]$ be a vector of *observed* responses, where $z(\theta^{(i)})$ is the response at the input vector $\theta^{(i)} = \left[\theta_1^{(i)}, \ldots, \theta_p^{(i)}\right]$, and we are interested in predicting output $z(\theta^{(\text{new})})$ at the untried input $\theta^{(\text{new})}$. The correlation between any two *unobserved* responses is assumed to have the form

$$C(\beta)_{i,t} \equiv \text{cor}\left(z(\theta^{(i)}), z(\theta^{(t)})\right) = \exp\left\{\sum_{k=1}^{p}\left(-\beta_k\left(\theta_k^{(i)} - \theta_k^{(t)}\right)^2\right)\right\}. \quad (1)$$

The correlation matrix $\text{C}(\beta) = [\text{C}(\beta)]_{i,t}$, and depends on the correlation parameters $\beta = [\beta_1, \ldots, \beta_p]$

Let $\mu(\cdot)$ be the mean function for the unconditional mean of any observation, and the mean matrix of z_{known} be

$$M \equiv \left[\mu\left(\theta^{(1)}\right), \ldots, \mu\left(\theta^{(m)}\right)\right]. \quad (2)$$

The vector of observed responses, z_{known}, is distributed according to

$$z_{\text{known}} \sim MVN_m(M, V), \quad (3)$$

where V is the variance-covariance matrix defined as

$$V \equiv \sigma_{GP}^2 C(\beta) + N, \tag{4}$$

where σ_{GP}^2 is the unconditional variance of an expected response and N is a diagonal *nugget matrix* with the i^{th} diagonal element equal to $\sigma_{\mathrm{e}}^2(\theta^{(i)})$, which is variance due to the stochasticity of the response (e.g., random noise) that may depend on θ. If output is *deterministic*, the nugget is not present so that $\sigma_{\mathrm{e}}^2(\theta) \equiv 0$. For *stochastic* responses, variance is traditionally taken to be constant so that $\sigma_{\mathrm{e}}^2(\theta) \equiv \sigma_{\mathrm{e}}^2$ and $N = \sigma_{\mathrm{e}}^2 I$. The package *mlegp* extends the traditional GP model by allowing the user to specify N exactly or N up to a multiplicative constant.

Define $r_i = \mathrm{cor}(z(\theta^{(new)}), z(\theta^{(i)}))$, following equation (1), and $r = [r_1, \ldots, r_m]'$. Under the GP assumption, the predictive distribution of $z(\theta^{(new)})$ is normal with mean

$$\widehat{z}\left(\theta^{(i)}\right) = \mathrm{E}[z(\theta^{(new)})|z_{\mathrm{known}}] = \mu(\theta^{(new)}) + \sigma_{GP}^2 r' V^{-1}(z_{\mathrm{known}} - M) \tag{5}$$

and variance

$$\mathrm{Var}[z(\theta^{(new)})|z_{\mathrm{known}}] = \sigma_{\mathrm{GP}}^2 + \sigma_{\mathrm{e}}^2(\theta) - \sigma_{GP}^4 r' V^{-1} r. \tag{6}$$

For more details, see Santner *et al.* (2003).

2.2 Maximum likelihood estimation

We first need some additional notation. Mean functions that are constant or linear in design parameters have the form $\mu(\theta) = x(\theta)F$, where $x(\theta)$ is a row vector of regression parameters, and F is a column vector of regression coefficients. Note that for a constant mean function, $x(\cdot) \equiv 1$ and F is a single value corresponding to the constant mean. The mean matrix M defined in equation (2) has the form $M = XF$, where the i^{th} row of X is equal to $x\left(\theta^{(i)}\right)$.

Let us also rewrite the variance-covariance matrix V from equation (4) to be

$$V \equiv \sigma_{\mathrm{GP}}^2(C(\beta) + aN_s) \equiv \sigma_{\mathrm{GP}}^2 W(\beta, a), \tag{7}$$

where N_s is the nugget matrix specified up to a multiplicative constant, with $N = \sigma_{GP}^2 a N_s$ and the matrix W depends on the correlation parameters $\beta = [\beta_1, \ldots, \beta_p]$ and a proportionality constant a.

When the matrix W is fully specified, maximum likelihood estimates of the mean regression parameters and σ_{GP}^2 exist in closed form and are

$$\widehat{F} = (X^T W^{-1} X)^{-1} X^T W^{-1} z_{\mathrm{known}} \tag{8}$$

and

$$\widehat{\sigma}_{\mathrm{GP}}^2 = \frac{1}{m}(z_{\mathrm{known}} - \widehat{M})^T W^{-1}(z_{\mathrm{known}} - \widehat{M}), \tag{9}$$

where $\widehat{M} = X\widehat{F}$.

2.3 Diagnostics

The cross-validated prediction $\widehat{z}_{-i}(\theta^{(i)})$ is the predicted response obtained using equation (5) after removing all responses at input vector $\theta^{(i)}$ from z_{known} to produce $z_{\mathrm{known,-i}}$. Note that it is possible for multiple $\theta^{(i)}$'s, for various i's, to be identical, in which case all corresponding observations are removed. The cross-validated residual for this observations is

$$\frac{z(\theta^{(i)}) - z_{-i}(\theta^{(i)})}{\sqrt{\mathrm{Var}(z(\theta^{(i)})|z_{\mathrm{known,-i}})}}. \tag{10}$$

2.4 What does *mlegp* do?

The package *mlegp* extends the standard GP model of (3), which assumes that $N = \sigma_e^2 I$, 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 $\text{cor}(\overline{z_{i.}}, \overline{z_{t.}})$, following Eq. (1). and the i^{th} element of the nugget matrix N given by $\frac{\sigma_e^2(\theta)}{n_i}$. 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 $N = \sigma_e^2 I$ 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)
```

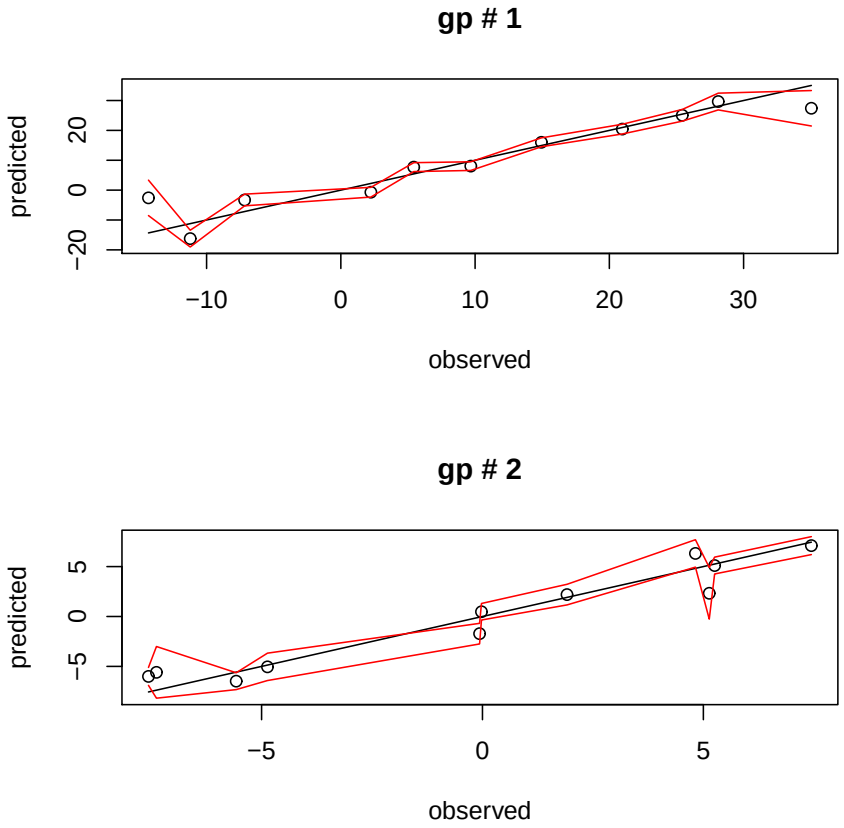

Figure 1: Gaussian process diagnostic plots. Open circles, cross-validated predictions; solid black lines, observed values; solid red lines, confidence bands corresponding to cross-validated predictions ± standard deviation.

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) $\pi(\theta_{-k})$ 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 $\pi(\theta_{-k,-l})$. 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'.

4.2 Examples

4.2.1 FANOVA decomposition

The function *FANOVADecomposition* is used to perform FANOVA decomposition on a single Gaussian processes, or on (a subset) of all Gaussian processes in a list. The function returns a table that reports the % contribution of each effect to the total functional variance of the Gaussian process predictor (or each Gaussian process predictor, in the case of a list). The code below demonstrates the use of the *FANOVADecomposition* function on a Gaussian process with two design parameters.

```
> x1 = kronecker(seq(0, 1, by = 0.25), rep(1, 5))
> x2 = rep(seq(0, 1, by = 0.25), 5)
> y = 4 * x1 - 2 * x2 + x1 * x2 + rnorm(length(x1), sd = 0.001)
> fit = mlegp(cbind(x1, x2), y, param.names = c("x1", "x2"))

> FANOVADecomposition(fit, verbose = F)

  param % contribution
1    x1     89.7624120
2    x2      9.8685623
3 x1:x2      0.3689804
```

4.2.2 Graphical plots for main and interaction effects

The function *plotMainEffects* is a generic function for plotting multiple main effects for a single Gaussian process; comparing a main effect across multiple Gaussian processes; and visualizing main effects of a single parameter on functional output. The first two uses of *plotMainEffects* are demonstrated below; an example of the latter can be found in Section (5).

First, we use *plotMainEffects* to plot the main effects for all input parameters on the Gaussian process created above. By default, all main effects are plotted, but a subset of effects can be specified by either name or number through the argument 'effects'. Setting 'FANOVA = TRUE' will calculate, for each main effect, the percentage contribution of that effect to the total functional variance of the GP predictor, and this will be reported in the legend. The function *plotInteractionEffect* is used to create a contour plot which visualizes interaction effects, and this is also demonstrated below. Output from the code can be seen in Figure (2).

```
> par(mfrow = c(1, 2))
> plotMainEffects(fit, graphStyle = 1, FANOVA = TRUE)
> plotInteractionEffect(fit, effects = c(1, 2))
```

It is also possible to use *plotMainEffects* to compare a main effect of a single parameter across multiple responses. We first create a Gaussian process list object that contains three GPs, each with the design parameter x. We then plot the main effect of x on the three GPs. This example also illustrates how the main effect can be referred to by name instead of by number. The main effects plot produced by the code is displayed in Figure (3).

```
> x = -5:5
> z1 = 10 - 5 * x + rnorm(length(x))
> z2 = 4 - 5 * x + rnorm(length(x))
> z3 = 7 * sin(x) + rnorm(length(x))
> fitMulti = mlegp(x, cbind(z1, z2, z3), param.names = "x")
```

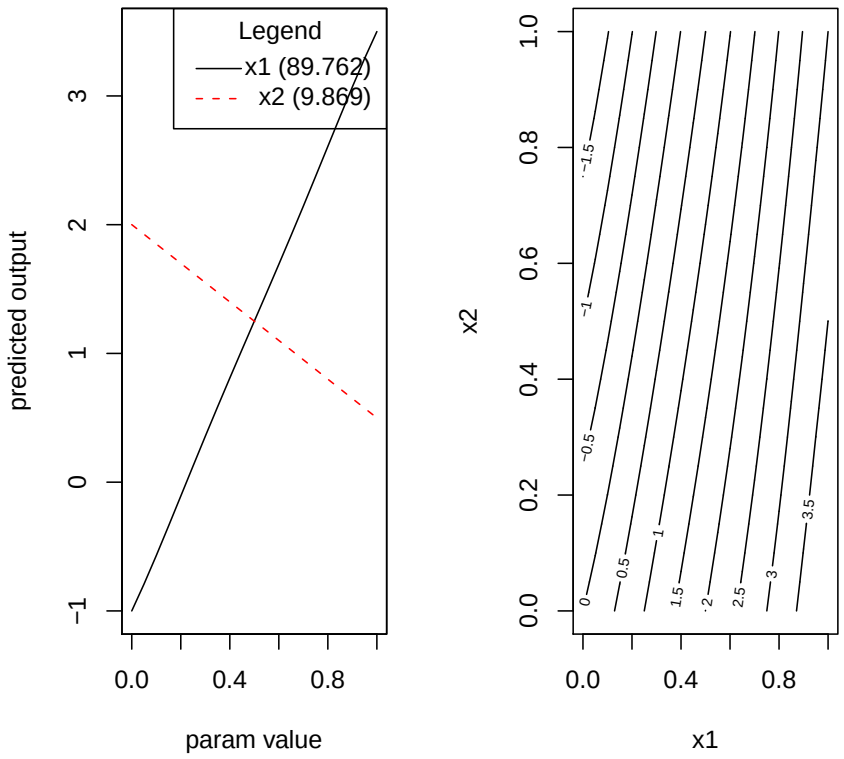

Figure 2: Main and Two-Way Interaction Effect Plots. For each main effect, the percent contribution of that effect to the total functional variance of the Gaussian process is also reported.

```
> plotMainEffects(fitMulti, effects = "x", graphStyle = 1)
```

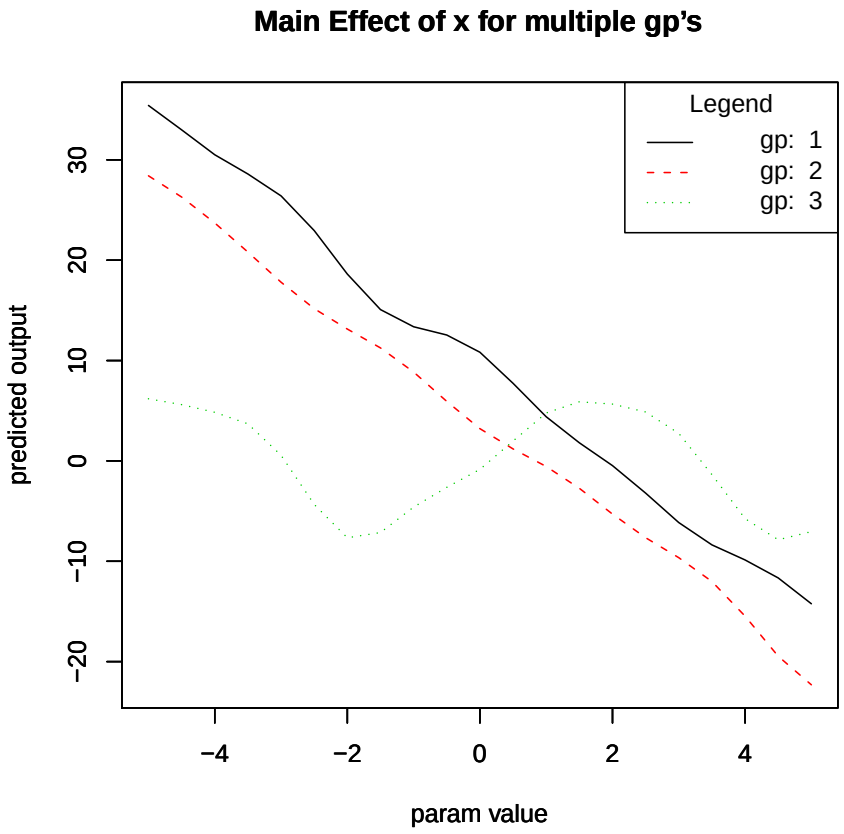

Figure 3: Main effects of the parameter x on a Gaussian process list that models three responses.

5 Multivariate Output and Dimension Reduction

5.1 Background

For multivariate or functional output, singular value decomposition can be used to reduce the dimensionality of the response (Heitmann *et al.*, 2006). Let $[z]_{i,j}$, $i = 1, \ldots, k$, $j = 1, \ldots, m$ be a matrix of m multivariate responses, where column j of the matrix contains the k-dimensional output of the response corresponding to the input parameter $\theta^{(j)}$. Also let $r = \min(k, m)$. Using singular value decomposition,

$$[z]_{i,j} = [U_{kxr} D_{rxr} V'_{rxm}]_{i,j} = \sum_{p=1}^{r} \lambda_p \{\alpha_p\}_i \{w_p(\theta)\}_j, \tag{11}$$

where λ_p is the p^{th} singluar value, α_p is the p^{th} column of U, and $w_p(\theta)$ is the p^{th} row of V'. We will refer to the j^{th} column of V', which contains the elements $\{w_p(\theta)\}_j$, $p = 1, \ldots, r$, as a vector of *principle component weights* corresponding to the j^{th} observation. The output z is approximated by keeping the $l < r$ most important principle component weights, corresponding to the l largest singular values. For a response matrix z as described above, *mlegp* fits independent Gaussian processes to the most important principle component weights. The number of principle component weights to be kept is specified through the argument 'PC.num'; alternatively, setting the argument 'PC.percent' will keep the most important principle component weights that account for 'PC.percent' of the variation in the response.

5.2 Examples

5.2.1 Basics: Modeling functional output

The first example demonstrates the use of *mlegp* to fit GPs to principle component weights in order to model functional output. The functional responses are sinusoidal, consisting of 161 points, with a vertical offset determined by the design parameter p. We first create the functional responses and plot them. This output is displayed in Figure (4).

```
> x = seq(-4, 4, by = 0.05)
> p = 1:10
> y = matrix(0, length(p), length(x))
> for (i in 1:length(p)) {
+     y[i, ] = sin(x) + 0.2 * i + rnorm(length(x), sd = 0.01)
+ }

> for (i in p) {
+     plot(x, y[i, ], type = "l", col = i, ylim = c(min(y), max(y)))
+     par(new = TRUE)
+ }
```

For functional output such as this, it is possible to fit separate GPs to each dimension. However, with 161 dimensions, this is not reasonable. In the code below, we first use the function *singularValueImportance* and see that the two most important principle component weights explain more than 99.99% of the variation in the response. Then, we fit the GPs to these two principle component weights. Note that in the call to *mlegp* we take the transpose of the response matrix, so that columns correspond to the functional responses.

```
> numPCs = 2
> singularValueImportance(t(y))[numPCs]

[1] 99.99604
```

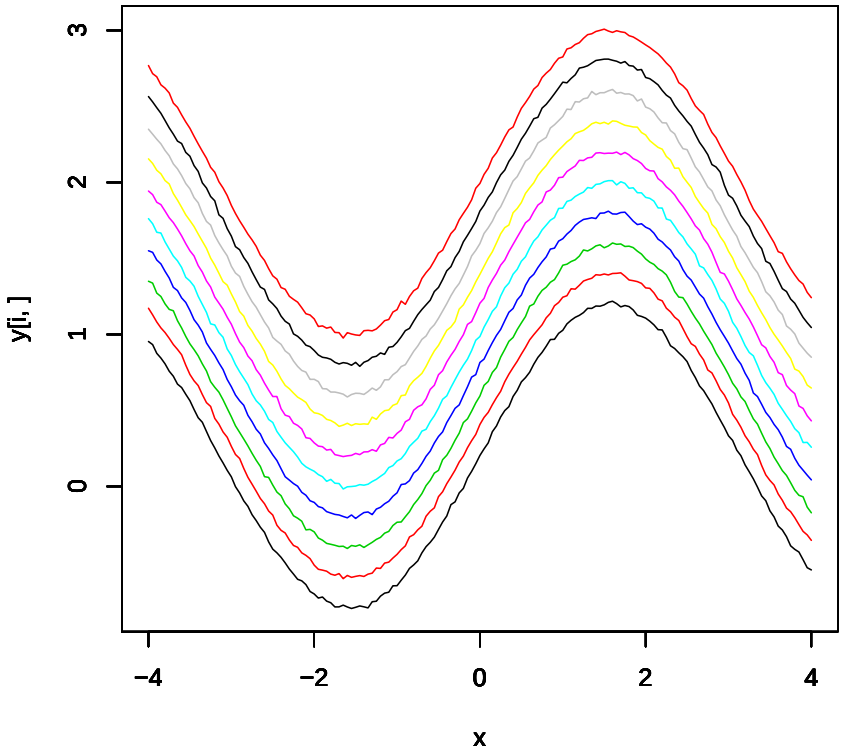

Figure 4: An example of functional responses where the design parameter determines the vertical offset

```
> fitPC = mlegp(p, t(y), PC.num = numPCs)
```

The GPs, which model principle component weights, can now be used to predict and analyze the functional response, based on the UDV' matrix of equation (11). The UD matrix corresponding to the principle component weights that are kept is saved as a component of the Gaussian process list object. The R code below demonstrates use of the *predict* method to reconstruct (approximately) the original functional output.

```
> Vprime = matrix(0, numPCs, length(p))
> Vprime[1, ] = predict(fitPC[[1]])
> Vprime[2, ] = predict(fitPC[[2]])
> predY = fitPC$UD %*% Vprime
```

5.2.2 Calculating main effects

The function *plotMainEffects* visualizes main effects of design parameters on functional output for GPs fit to principle component weights. This is demonstrated below, for the Gaussian processes that were fit above. The graphical plot is displayed in Figure (5).

```
> plotMainEffects(fitPC, effect = 1, graphStyle = 1)
```

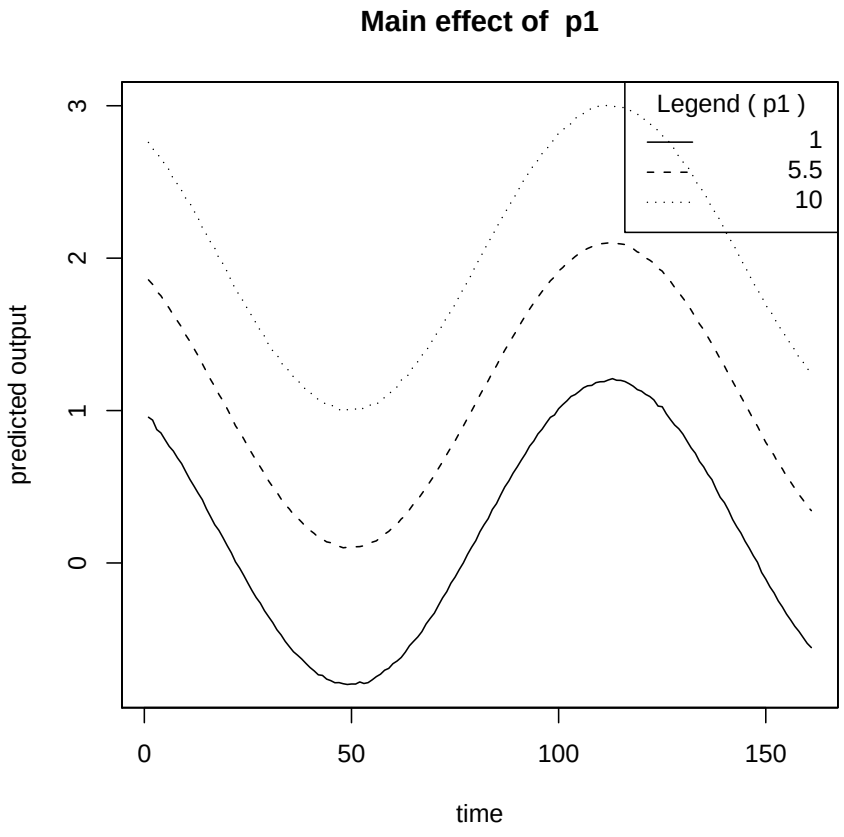

Figure 5: Main effect of $p1$ on functional output, based on Gaussian process modeling of the two most important principle component weights.

5.2.3 Modeling high dimensional (and not necessarily functional) data

Although the above example involves functional output, the singular value decomposition technique can applied generally to any multi-dimensional response. One can also use *mlegp* to analyze non-functional data of high-dimension, as well as multiple functional responses, by manipulating the output of *predict* and calling *plotMainEffects* functions with 'no.plot'= 'TRUE', which returns the main effects without plotting them. In our final example, we consider *two* sets of functional outputs: the sinusoidal response $y1$ and the linear response $y2$. Both responses depend on the design parameter p. After combining the output vectors $y1$ and $y2$, we fit Gaussian processes to the most important principle component weights that explain at least 99.999% of the variation in output.

```
> p = 1:10
> x1 = seq(-4, 4, by = 0.05)
> x2 = 1:5
> y1 = matrix(0, length(p), length(x1))
> y2 = matrix(0, length(p), length(x2))
> for (i in 1:length(p)) {
+     y1[i, ] = sin(x1) + 0.2 * p[i] + rnorm(length(x1), sd = 0.01)
+     y2[i, ] = 0.1 * p[i] + x2 + rnorm(length(x2), sd = 0.01)
+ }
> y = cbind(y1, y2)
> fitPC = mlegp(p, t(y), PC.percent = 99.999)
```

Plotting main effects for the different types of responses is now a matter of retrieving all of the main effects from *plotMainEffects*, and breaking this predicted effect into the separate main effects for the two responses of interest. Main effects for both responses are displayed in Figure (6).

```
> main = plotMainEffects(fitPC, effect = 1, no.plot = TRUE)
> preds = main$preds
> beg1 = 1
> end1 = length(x1)
> beg2 = end1 + 1
> end2 = beg2 + length(x2) - 1
> par(mfrow = c(1, 2))
> for (i in 1:dim(preds)[1]) {
+     plot(x1, preds[i, beg1:end1], ylim = c(min(preds[, beg1:end1]),
+         max(preds[, beg1:end1])), type = "l", col = i, xlab = "x1",
+         ylab = "y1", main = "main effect on y1")
+     par(new = TRUE)
+ }
> par(new = FALSE)
> for (i in 1:dim(preds)[1]) {
+     plot(x2, preds[i, beg2:end2], ylim = c(min(preds[, beg2:end2]),
+         max(preds[, beg2:end2])), type = "l", col = i, xlab = "x2",
+         ylab = "y2", main = "main effect on y2")
+     par(new = TRUE)
+ }
```

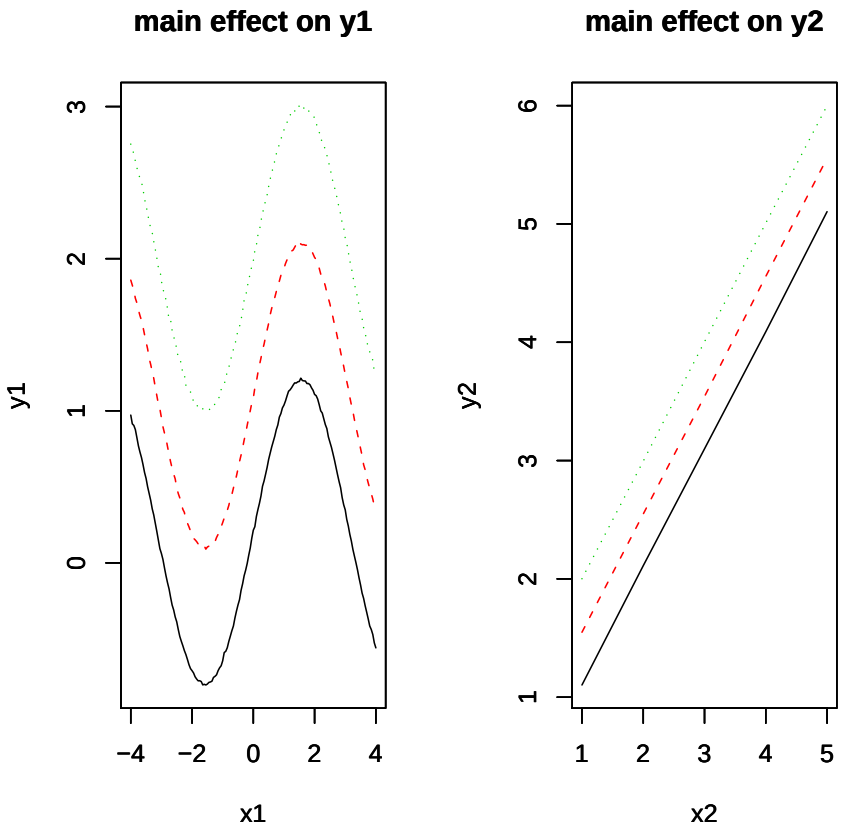

Figure 6: Main effects of the parameter p on functional responses $y1$ (left) and $y2$ (right). Values of p are 1 (black solid lines), 5.5 (red dashed lines), and 10 (green dotted lines).

References

Heitmann, K., Higdon, D., Nakhleh, C., Habib, S., 2006. Cosmic Calibrat ion, *The Astrophysical Journal*, **646**, 2, L1-L4.

Saltelli, A., Chan, K., Scott, E.M., 2000. Sensitivity analysis. (Chichester; New York: Wiley).

Santner, T.J., Williams, B.J., Notz, W., 2003. The Design and Analysis of Computer Experiments (New York: Springer).

Schonlau, M. and Welch, W., 2006. Screening the Input Variables to a Comp uter Model Via Analysis of Variance and Visualization, in Screening: Methods for Experimentation in Industry, Drug Discovery, and Genetics. A. Dean and S. Lewis, eds. (New York: Springer).

Programming Acknowledgements

- C code for random number generation provided by Mutsuo Saito, Makoto Matsumoto and Hiroshima University (`http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT`)
- C code for L-BFGS algorithm provided by Naoaki Okazaki (`http://www.chokkan.org/software/liblbfgs`)