output of the response corresponding to the input parameter $\theta^{(j)}$. Also let $r = \min(k, m)$. Using singular value decomposition,

$$[z]_{i,j} = [U_{kxr} D_{rxr} V'_{rxm}]_{i,j} = \sum_{p=1}^{r} \lambda_p \{\alpha_p\}_i \{w_p(\theta)\}_j, \tag{10}$$

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 (5).

```
> 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] 99.99614

> 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 (10). 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.

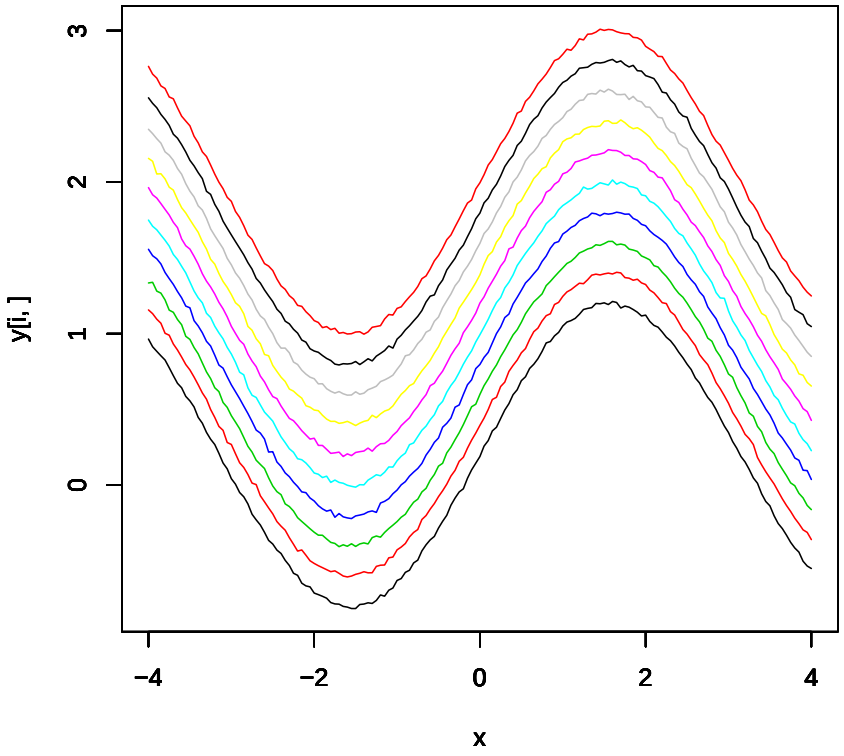

Figure 5: An example of functional responses where the design parameter determines the vertical offset

```
> 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 (6).

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

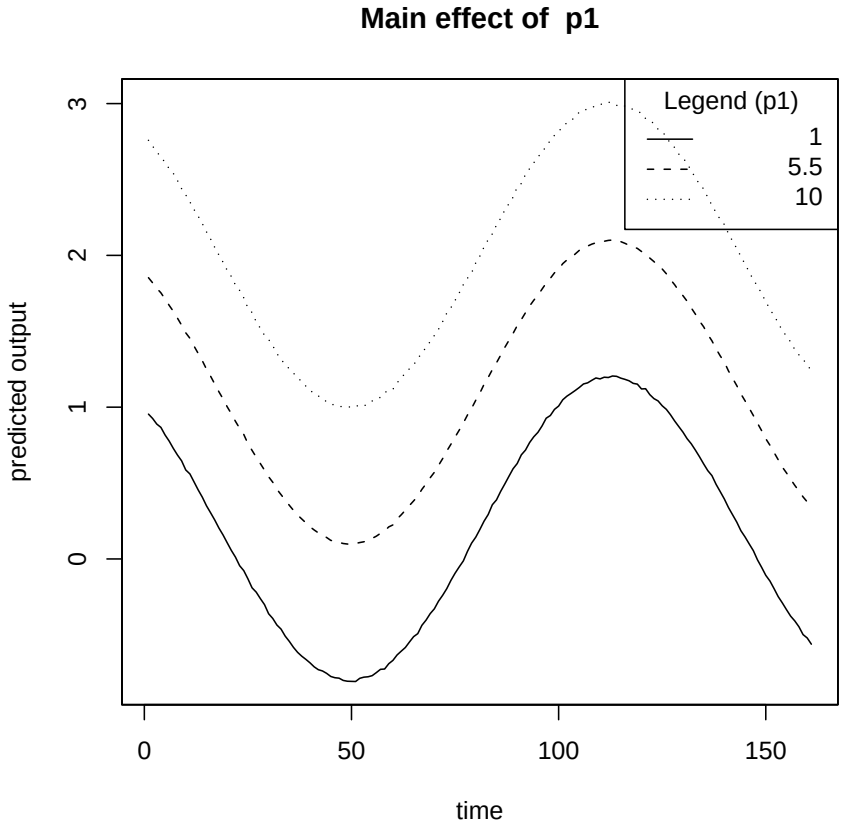

Figure 6: Main effect of $p1$ on functional output, based on Gaussian process modeling of the two most important principle component weights.

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 (7).

```
> 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)
+ }
```

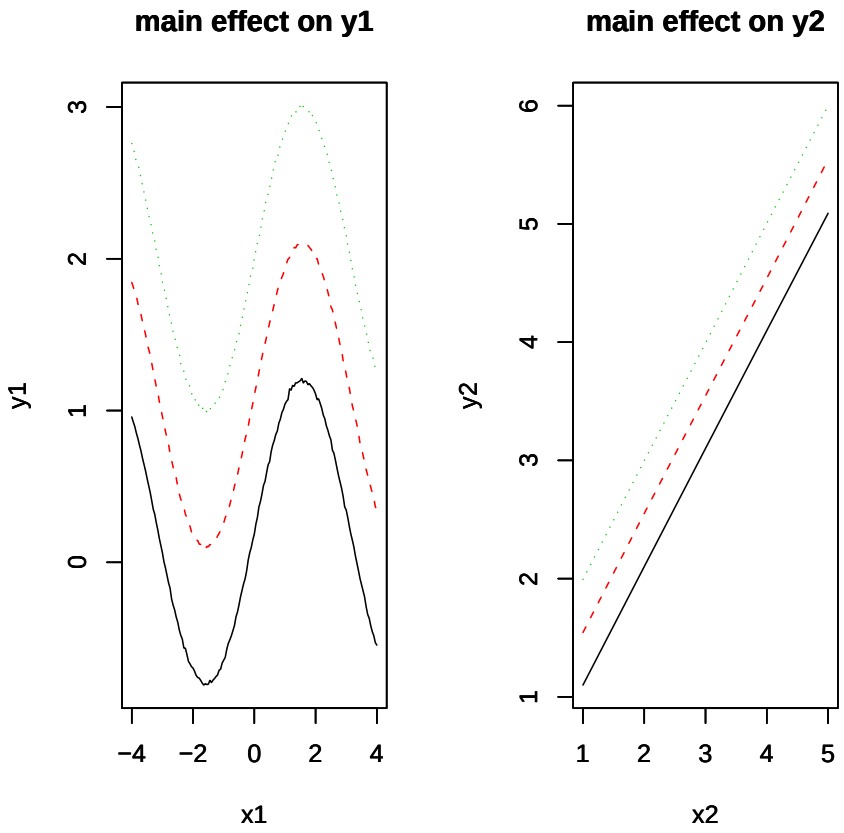

Figure 7: Main effects of the parameter p on functional responses $y1$ (left) and $y2$ (right). Values of p are 1 (black solid lines), 5.5 (red dashed lines), and 10 (green dotted lines).

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`)