GaSP: Train and Apply a Gaussian Stochastic Process Model

Yilin Yang and William J. Welch

2023-12-07

1. Introduction

Controlled physical experiments for complex phenomena may be expensive and time-consuming or, in some cases, impossible. Thus, the need for computer models to emulate such physical systems arises. Generally, computer experiments using statistical models will take a vector of d inputs $\mathbf{x}$ and produce a corresponding scalar output response $y(\mathbf{x})$. One distinction from physical experiments is that computer experiments may be deterministic: the same set of inputs will generate the same results. Hence there is need for special statistical methods.

Among these models is the popular model archetype called a Gaussian Stochastic Process (GaSP) or simply a Gaussian Process (GP). The core components are the mean function $\mu(\mathbf{x})$, the zero-mean stochastic process $Z(\mathbf{x})$, and an optional random error ϵ (absent if $y(\mathbf{x})$ is deterministic). The stochastic process will have a correlation function denoted as $R(\mathbf{x}, \mathbf{x}')$, it quantifies the relationship of the two response variables y and y' from the inputs $\mathbf{x}$ and $\mathbf{x}'$. The objective of package `GaSP` is to train a GaSP model via maximum likelihood estimation (MLE) or maximum a posteriori (MAP) estimation, run model diagnostics, and make predictions following Sacks et al. (1989). It can also perform sensitivity analysis and visualize low-order effects following Schonlau and Welch (2006).

The regression model is fairly flexible as defined by a model formula, and `GaSP` implements two popular correlation function families for the stochastic process: the Matérn and the power-exponential families. These families will be detailed in Section 3.3, but we note that the smoothness can be optimized in both cases. Following the prediction method described in Sacks et al. (1989), `GaSP` uses the "plug-in" estimated parameters obtained from the fitting method to calculate the best linear unbiased predictor of the response at any untried input vector $\mathbf{x}$ along with a standard error. Leave-one-out cross-validation (CV) predictions are also available along with many model diagnostic plots such as residual plots, standardized residual plots, and normal quantile-quantile (Q-Q) plots for both CV and out-of-sample predictions.

For sensitivity analysis and visualization of low-order effects, `GaSP` has the capability to perform functional analysis of variance (FANOVA) decomposition and plot estimated main and two-factor joint effects. This will be detailed in Section 9.

The package has little `R` overhead: all computationally intense matrix calculations are coded in `C` for efficiency.

This vignette aims to explain how to utilize `GaSP` in a research setting and help users to interpret the results. The authors have inevitably made implementation choices, some different from other packages, and those details need to be emphasized. The vignette will be divided into sections that follow the standard workflow order. Each section will feature code examples, interpretation of the results, and implementation details. Use `help(package = GaSP)` in R to access the full documentation.

2. Data Setup

`GaSP` uses training data passed in by two arguments:

- `x` is a dataframe containing n runs in the rows and the values of d input variables in the columns;

- `y` is a vector or dataframe containing the corresponding n values of a single output variable in the rows.

A `matrix` type is also allowed instead of a `dataframe`.

We will use the borehole data provided in `GaSP` for the data setup as follows:

```
library(GaSP)
x <- borehole$x
y <- borehole$y
x_pred <- borehole$x_pred
y_true <- borehole$y_true
```

If we look at the first three rows of inputs in `x` and the corresponding outputs in `y`

```
head(borehole$x, n = 3)
         rw        r        Tu       Hu        Tl       Hl        L       Kw
1 0.0730769 14174.34  64416.92 1057.692 116.00000 733.8461 1191.795 12045.00
2 0.0961539  6497.43 112906.16 1048.461  98.36668 813.8462 1421.539 11595.77
3 0.0935897 38484.63 107518.47 1066.923 106.50514 798.4616 1148.718 11090.39
head(borehole$y, n = 3)
         y
1 54.75499
2 55.35180
3 70.95713
```

we see that the variables in `x` are on the original scales of the application. For the user's convenience and scientific interpretability (especially in plots), the columns of `x` need not be scaled by the user to $[0, 1]$. Similarly, `y` need not be rescaled to have, say, mean zero or standard deviation 1. Where necessary, `GaSP` will perform scaling internally but report back results on the original scales.

For brevity, unless otherwise stated, `x` and `y` will be refer to `borehole$x` and `borehole$y` in further examples of this vignette.

Similarly, `x_pred` is a dataset of the inputs for untried runs where we want to predict y, and `y_true` contains true values for benchmarking of prediction accuracy; `x_pred` and `y_true` are set up analogously to `x` and `y`. For more details of the borehole function, please refer to the website in the citations by Surjanovic and Bingham (2013).

Currently `GaSP` does not handle missing values. If one or more inputs or the response is missing from an observation, that observation should be deleted.

3. `GaSP` Model Formulation

`GaSP` functions communicate through what we call a `GaSPModel` object, because it can be generated by the function `GaSPModel`. It contains the model formulation as well as quantities computed by `GaSP`. The function `GaSPModel` is described further in Section 4, but often the user will rely on `Fit` in Section 5 to implicitly create the same object from the model specification, along with parameter estimates. Either way, the components of the model need to be defined; their descriptions in this section will also help interpretation of the model parameters and their estimates.

3.1 `GaSP` model components

Following the approach of Sacks et al. (1989), `GaSP` treats observations of an unknown function $Y(\mathbf{x})$ as arising from the data model

$$Y(\mathbf{x}) = \mu(\mathbf{x}) + Z(\mathbf{x}) + \epsilon.$$

It comprises a mean function $\mu(\mathbf{x})$, a Gaussian stochastic process $Z(\mathbf{x})$ and an optional random error ϵ. The random error term is in principle absent for evaluations of a deterministic function, as from a computer experiment, but even here we need to think of a random function $Y(\mathbf{x})$ to provide a framework for quantifying uncertainty about the value of the function where it has not been observed. We now explain the details of the model components; those details will aid the interpretation of parameter estimates, etc.

3.2 Mean (regression) function

We can write the mean (regression) function as:

$$\mu(\mathbf{x}) = \sum_{j=1}^{k} \beta_j f_j(\mathbf{x})$$

Here, the β_j are unknown linear model regression coefficients to be estimated by `GaSP`, and the $f_j(\mathbf{x})$ are user-defined functions. In a `GaSPModel`, the mean function formula is specified by parameter `reg_model` using syntax that is similar to the `formula` parameter in say `lm()`. There is a restriction, however, to only polynomial models, i.e., powers of the inputs and interaction terms.

The simplest model is a constant regression $\mu(\mathbf{x}) = \beta_1 1 = \beta_1$, i.e., $k = 1$ and $f_1(\mathbf{x}) = 1$, where `GaSP` functions would have the argument

```
reg_model = ~ 1
```

Note first that the formula has no left-hand side, like in `y ~ 1`: the response variable is always the single variable appearing in the training data, say `borehole$y`. Secondly, while this simple model often works well, as demonstrated by Chen et al. (2016), and is very widely used, it is not a default. The regression model must always be specified.

As a slightly more complicated illustration, a regression model with first-order terms in the first three `borehole` inputs, plus a constant or intercept, is given by

```
reg_model_first = ~ 1 + r + rw + Tu
```

and passed to functions via `reg_model = reg_model_first`. Mathematically, we can express this model as:

$$\mu(\mathbf{x}) = \beta_1 + \beta_2 x_{\mathrm{r}} + \beta_3 x_{\mathrm{rw}} + \beta_4 x_{\mathrm{Tu}}$$

A more complicated model such as

```
reg_model_bizarre <- ~ 1 + (r + rw + Tu)^2 + I(Hu^2)
```

could also be passed to the `reg_model` argument. Mathematically, the regression model is

$$\mu(\mathbf{x}) = \beta_1 + \beta_2 x_{\mathrm{r}} + \beta_3 x_{\mathrm{rw}} + \beta_4 x_{\mathrm{Tu}} + \beta_5 x_{\mathrm{r}}^2 + \beta_6 x_{\mathrm{rw}}^2 + \beta_7 x_{\mathrm{Tu}}^2 + \beta_8 x_{\mathrm{r}} x_{\mathrm{rw}} + \beta_9 x_{\mathrm{r}} x_{\mathrm{Tu}} + \beta_{10} x_{\mathrm{rw}} x_{\mathrm{Tu}} + \beta_{11} x_{\mathrm{Hu}}^2$$

which is bizarre and definitely not recommended but demonstrates the flexibility. As usual, `Hu^2` has to be protected by `I()`.

3.3 Stochastic process component

The random process $Z(\mathbf{x})$ is assumed to have mean zero, variance σ_Z^2, and correlation function $R(\mathbf{x}, \mathbf{x}')$ for the correlation between $Z(\mathbf{x})$ and $Z(\mathbf{x}')$ at any two input vectors $\mathbf{x}$ and $\mathbf{x}'$. The correlation structure is important in predicting at untried inputs not in the training data.

Mathematically, `GaSP` uses a so-called product correlation structure,

$$R(\mathbf{x}, \mathbf{x}') = \prod_{j=1}^{d} R_j(h_j),$$

where the product is over the d inputs, $h_j = |x_j - x'_j|$ is a distance in the input j dimension, and $R_j(h_j)$ is a correlation function on $[0, 1]$

We next show that the variables called x_j here can be derived from the original inputs, before moving on to the important topic of the choice of correlation function.

The variables in the stochastic process component are specified by the parameter `sp_model`. The default `sp_model = NULL` uses in the above product all of the orginal inputs x_j appearing in the training data, and the user does not need to do anything in this typical case. If for some reason in the `borehole` application the `sp_model` argument is say

```
sp_model = ~ r + rw + Tu
```

then the product would only be over the inputs `r`, `rw` and `Tu`. Note there is no constant term in `sp_model` as the correlation function works on differences, and a constant cancels; a warning would be generated but there is no need for alarm. Note also that different variables or derived variables can appear in the regression and stochastic-process components.

Similar to `reg_model`, powers and interaction terms are allowed. For example, a set of variables in the stochastic process component of the model could be specified by

```
sp_model_bizarre = ~ (r + rw + Tu)^2 + I(Hu^2)
```

whereupon the variables

$$x_{\mathrm{r}},\ x_{\mathrm{rw}},\ x_{\mathrm{Tu}},\ x_{\mathrm{r}}^2,\ x_{\mathrm{rw}}^2,\ x_{\mathrm{Tu}}^2,\ x_{\mathrm{r}}x_{\mathrm{rw}},\ x_{\mathrm{r}}x_{\mathrm{Tu}},\ x_{\mathrm{rw}}x_{\mathrm{Tu}},\ x_{\mathrm{Hu}}^2$$

are the 10 "inputs" used in the correlation function. Again, the choice here is not recommended but shows the flexibility.

Next we describe the two families of correlation functions implemented by `GaSP` for the $R_j(\cdot)$ in the product correlation structure.

- **Power-exponential.** We parameterize the power-exponential correlation function as

  $$R_j(h_j) = \exp(-\theta_j h_j^{2-\alpha_j}).$$

  Here $R_j(h_j)$ depends on a distance-scale parameter $\theta_j \geq 0$, controlling the rate of correlation decay as the distance h_j (from x_j) increases. Different from some other packages, θ_j is in the numerator. Thus $\theta_j = 0$ implies perfect correlation, making x_j an inactive input. The smoothness parameter $0 \leq \alpha_j \leq 1$ defines the power as $2 - \alpha_j$, and again the specification is different from some other implementations. Hence, $\alpha_j = 0$ gives the extremely smooth squared-exponential (Gaussian) special case. Similarly, $\alpha_j = 1$ gives the special case known as the exponential correlation. Power-exponential is a generalization of these special cases and is much more flexible, though we will also describle later how to impose such special cases. Subject to any user constraints, `GaSP` will estimate the θ_j and α_j parameters separately for each input for a so-called anisotropic correlation function.

- **Matérn**. The parameterization of the Matérn correlation function follows Chen et al. (2016) and allows four discrete levels of smoothness controlled by the parameter δ_j, which is the number of derivatives:

  $$R_j(h_j) = \begin{cases} \exp(-\theta_j h_j) \text{ for } \delta_j = 0 \text{ (the exponential correlation)} \\ \exp(-\theta_j h_j)(\theta_j h_j + 1) \text{ for } \delta_j = 1 \\ \exp(-\theta_j h_j)((\theta_j h_j)^2/3 + \theta_j h_j + 1) \text{ for } \delta_j = 2 \\ \exp(-\theta_j h_j^2) \text{ for } \delta_j \to \infty \text{ (the squared-exponential correlation)} \end{cases}$$

In GaSP, δ is called Derivatives taking values 0, 1, 2, or 3, with $\delta_j \to \infty$ coded as 3. The setup here implies that θ_j has the same interpretation for the exponential and squared-exponential special cases common to the power-expontential and Matérn families. Similar to the power-exponential case, GaSP will fit the θ_j and δ_j parameters separately for each input, though the user is again allowed to constrain their ranges, for example restricting to exactly one or two derivatives, as is sometimes done.

GaSP stores the values of the correlation parameters in a dataframe named cor_par with one row for each term in the stochastic process model and two columns. The first column is named Theta, and the second is either Alpha for the power-exponential case or Derivatives for the Matérn case.

3.4 Random error component

The random error term ϵ is independent "white noise" with variance σ^2_ϵ. Its main purpose is to represent genuine measurement error, but it is also sometimes used for computational stability as a so-called "nugget" term even when the input-output relationship is deterministic.

To understand the (possibly confusing) interplay of the two distinct roles of the random error term, it is helpful to know how GaSP handles variance parameters internally. Let $\sigma^2 = \sigma^2_Z + \sigma^2_\epsilon$ be the total variance, and let $\gamma = \sigma^2_Z/\sigma^2$ be the proportion of the total variance due to the stochastic process. Internally GaSP optimizes σ^2 and γ but reports back $\sigma^2_Z = \gamma\sigma^2$ and $\sigma^2_\epsilon = (1-\gamma)\sigma^2$ as sp_var and error_var, respectively.

For computational stability, the user argument nugget taking values in $[0, 1]$ is available to provide a ceiling on γ: for example, the default nugget = 1e-9 means that γ cannot exceed $1 - 10^{-9}$. Thus, no correlation can exceed that ceiling, ruling out correlations of 1 and singular matrices. (In practice, GaSP has few problems with ill-conditioning even with zero nugget: warnings may be issued but the final result is rarely an error flag.)

The boolean argument random_error indicates whether GaSP should optimize γ and hence estimate a genuine $\sigma^2_\epsilon = (1-\gamma)\sigma^2$. If the user passes random_error = TRUE, γ is optimized subject to not exceeding the complement of nugget as above.

Thus, we can think of four combinations of random_error (TRUE / FALSE) and nugget (zero and non-zero).

- random_error = FALSE and nugget = 0: the estimate of the proportion of the total variance due to the stochastic process will be 1, i.e., the pure deterministic model with only $\sigma^2 = \sigma^2_Z$ to be estimated.
- random_error = FALSE and nugget > 0: the proportion of the total variance due to the stochastic process is fixed at 1− nugget, usually to represent a deterministic model with a small amount of random error for numerical stability.
- random_error = TRUE and nugget = 0: the estimate of the proportion of the total variance due to the stochastic process versus random error will be unbounded between 0 and 1 during optimization, i.e., the 'noisy data' model where σ^2_Z and σ^2_ϵ are both estimated without any restriction.
- random_error = TRUE and nugget > 0: again both variances are estimated but the proportion of the total variance due to the stochastic process will be upper bounded by 1− nugget during optimization.

It should be noted that when random_error = FALSE and nugget > 0, the resulting error variance error_var will be greater than 0, and we have a contradiction as random_error = FALSE assumes error_var = 0. For consistency with Fit, functions such as Predict that require a GaSPModel object as input will generate a warning that GaSP is assuming random_error = TRUE in these cases. There is no need to be alarmed, as these functions simply will use random_error = TRUE instead; the values of sp_var and error_var passed in will not be changed by the function.

4. GaSPModel object

After setting up our data and deciding on a model, GaSP provides two options to obtain a GaSPModel object: Fit and GaSPModel. Fit returns such an object with the model parameters trained via the MLE or MAP methods detailed in Section 5. Alternatively, the user can specify the model's parameters directly, perhaps

taking the results of another package, with the function `GaSPModel`. Either way the object is needed as input to the `GaSP` functions `Predict`, `CrossValidate`, and `Visualize`.

A `GaSPModel` object must have the following components:

- `x` and `y`: the input (explanatory variable) training data and output (response) training data.
- `reg_model` and `sp_model`: respectively the regression model and an optional stochastic process model, as specified in Sections 3.2 and 3.3.
- `cor_family` and `cor_par`: respectively the correlation family and the data frame containing the correlation parameters, as specified in Section 3.3.
- `random_error`, `sp_var` and `error_var`: respectively the boolean to indicate presence of a random error term, the stochastic process variance, and the random error variance, as described in Section 3.4.

The following additional components are optional (and generated by `Fit`) in the sense that `Predict`, `CrossValidate` and `Visualize` will ignore the user's values (and compute them from scratch if necessary):

- `beta`: the regression parameters β_j for the mean function in Section 3.2.
- `objective`, `cond_num` and `CVRMSE`: diagnostics from `Fit`, which are respectively the maximum objective found, the condition number of the matrix calculations, and the model's cross-validation root mean squared error (see Section 5).

To illustrate how the user can directly create a `GaSPModel` object:

```
theta <- c(
  5.767699e+01, 0.000000e+00, 0.000000e+00, 1.433571e-06,
  0.000000e+00, 2.366557e-06, 1.695619e-07, 2.454376e-09
)
alpha <- c(
  1.110223e-16, 0.000000e+00, 0.000000e+00, 0.000000e+00,
  0.000000e+00, 0.000000e+00, 2.494862e-03, 0.000000e+00
)
cor_par <- data.frame(Theta = theta, Alpha = alpha)
rownames(cor_par) <- colnames(borehole$x)
sp_var <- 38783.7
borehole_gasp <- GaSPModel(
  x = x, y = y,
  reg_model = ~1, cor_family = "PowerExponential",
  cor_par = cor_par, random_error = FALSE,
  sp_var = sp_var
)
```

The behavior of defaults is the same as with `Fit` in Section 5. Here the `cor_family` will default to `"PowerExponential"`, and `sp_model` will by default use all variables in `x`. However, it should be noted that `sp_var` needs to be specified, and `error_var` will also require specification when `random_error = FALSE`. Note also that `cor_par` takes its row names for the variables in the stochastic process model from the column names of `x`, because of the use of default stochastic process model here. In general the row names of `cor_par` should match the terms implied by `sp_mod`, so that the results are easy to interpret. `GaSP` makes several compatibility checks of this kind, and issues helpful warning/error messages if there is a problem. Despite the plethora of parameter checks, `GaSP` has no way of knowing if the parameters will generate a good, stable result, and it is up to the user to ensure the values generate the desired model.

5. Fit

Following the methods introduced by Sacks et al. (1989), in `GaSP` we specify the objective that `Fit` attempts to maximize by setting `fit_objective` to `"Likelihood"` for maximum likelihood estimation (MLE). Alternatively, `fit_objective = "Posterior"` for Bayesian maximum a posteriori (MAP) estimation.

5.1 Maximum likelihood estimation

To train a model via MLE we can run `Fit` as in the following example:

```
borehole_fit <- Fit(
  reg_model = ~1, x = x, y = y, cor_family = "PowerExponential",
  random_error = TRUE, fit_objective = "Likelihood", model_comparison = "Objective"
)
```

Here `x = x` refers to the borehole `x` in Section 2, and similarly for `y`. We choose the power-exponential correlation family with random error and the default nugget value. The parameter `model_comparison` is the criterion used to select from multiple solutions when there are multiple optimization tries from different starting points: either the objective function used in optimization `"Objective"` or leave-one-out cross validation `"CV"`. With MLE the objective function is the profile log likelihood:

$$-\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\widehat{\sigma^2}) - \frac{1}{2}\ln(\det(\mathbf{R})) - \frac{n}{2}$$

We operate on the log-scale for numerical stability.

In the profile log likelihood,

$$\widehat{\sigma^2} = \frac{1}{n}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})$$

is the generalized least squares estimate of the total variance σ^2 in Section 3.4, which is available in closed form. The correlation parameters have to be numerically optimized, however. As well as the parameters in `cor_par`, if `random_error = TRUE` then γ in Section 3.4 is another correlation parameter to be simultaneously optimized. When `Fit` terminates it reports back

$$\widehat{\sigma_Z^2} = \frac{\gamma}{n}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})$$

and

$$\widehat{\sigma_\epsilon^2} = \frac{(1-\gamma)}{n}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})$$

as `sp_var` and `error_var`, respectively.

`Fit` returns a `GaSPModel` class object, to be used in further calculations.

5.2 Maximum a posteriori (MAP) estimation

MAP estimation is chosen via `fit_objective = "Posterior"`, e.g.,

```
borehole_fit <- Fit(
  reg_model = ~1, x = x, y = y, cor_family = "Matern",
  random_error = FALSE, nugget = 0, fit_objective = "Posterior"
)
```

Again, it is actually the log of the profile objective, now the posterior, that is numerically maximized:

$$\lambda\sum_j \theta_j - \frac{n-k}{2}\ln(\widehat{\sigma^2}) - \frac{1}{2}\ln(\det(\mathbf{R})) - \frac{1}{2}\ln(\det((\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F}))).$$

This expression differs from the profile log likelihood in several ways. First, independent exponential priors with rate parameter λ on each correlation parameter θ_j give the term $\lambda\sum_j \theta_j$. The user can set λ via `lambda_prior`, which takes the default 0.1 here. (The same λ applies to all correlation parameters θ_j, and this is the only place where the scaling of the inputs and hence the θ_j matters.) Secondly, independent

limiting constant priors for the linear model regression coefficients β_j and a Jeffery's prior for σ^2 proportional to $1/\sigma^2$ generate a degrees of freedom adjustment

$$\widehat{\sigma^2} = \frac{1}{n-k}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})^T \mathbf{R}^{-1}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})$$

in the MAP estimate of σ^2 as well as the multiplier $(n-k)/2$ in the log profile; see Handcock and Stein (1993).

The example also specifies the Matérn correlation family via `cor_family = "Matern"`, a choice that could also be made for MLE.

5.3 Further details of Fit

In both estimation methods we did not address two parameters `log_obj_tol` and `log_obj_diff` as they relate to details of the optimization process. Here are their definitions:

- `log_obj_tol`: An absolute tolerance for terminating the maximization of the log of the objective. It is the stopping criterion for `Fit`.
- `log_obj_diff`: The default value is 0 and will have no effect on the maximization. However, if set to a value greater than 0, during iterations of the optimization an informal hypothesis test of $\theta_j = 0$ is repeatedly carried out to try to simplify the model. The test compares the difference in the log objective with and without the hypothesis imposed against `log_obj_diff`.

In summary, here are all the parameters for `Fit` that always need to be specified.

- `x` and `y`: the user has to specify the training data.
- `reg_model`: the user has to specify the mean function.
- `random_error`: the user has to specify whether there is random error.

In contrast, most parameters for `Fit` have defaults.

- `sp_model`: the default is `NULL` and all variables in `x` will be used.
- `cor_family`: the default is `"PowerExponential"`.
- `cor_par`: user-specified starting values for the first try to maximize the objective; the default empty dataframe simply communicates to the optimizer that `Fit` always generates its own starting values.
- `sp_var` and `error_var`: user-specified starting values for the first try to maximize the objective; the defaults of -1, which are clearly illegal for variances, communicate to the optimizer that `Fit` always generates its own starting values. Note the values are not used at all if `random_error = FALSE`.
- `nugget`: the default for the random error nugget is `1e-9`.
- `tries`: the number of tries to optimize the objective from different starts; the default is `10`.
- `seed`: the default for the random seed number is `500`.
- `fit_objective`: the default is `"Likelihood"`.
- `theta_standardized_min` and `theta_standardized_max`: the lower and upper bound on each θ_j, standardized for the scale of the corresponding stochastic-process term. They take the defaults 0 and ∞, respectively.
- `alpha_min` and `alpha_max`: the lower and upper bound on each α_j in the power-exponential correlation family, with defaults 0 and 1.
- `derivatives_min` and `derivatives_max`: These are the ranges for the number of derivative in the Matérn correlation family, with defaults 0 (no derivatives) and 3 (a code for the infinitely differentiable squared-exponential).
- `log_obj_tol`: the default is `1e-5`.
- `log_obj_diff`: the default is `0` (no attempt to simplify the model).
- `lambda_prior`: the default is `0.1`.
- `model_comparison`: the default is `"Objective"`.

One parameter often manipulated is `alpha_max`: setting it to zero will give the squared-exponential correlation family. Similarly, the Matérn correlation function with 2 derivatives, another popular choice, is specified by `cor_family = "Matern"`, `derivatives_min = 2` and `derivatives_max = 2`.

This list is just to get a general idea of why many variables are not specified in our examples. Furthermore, `GaSP` performs thorough parameter checks, for example:

```
borehole_fit <- Fit(x = x, y = y,
  reg_model = ~ 1 + a, sp_model = ~ 1 + r, random_error = FALSE
)
Warning: intercept term in 'sp_model' will not be used.
Error:
1: components of 'reg_model' terms must be column names in 'x'.
```

The error occurs bcecause `a` is not a variable in the (borehole) inputs `x`.

Warnings will be printed as soon as one is detected, and Errors will be compiled into a list before quitting gracefully.

As stated in Section 4, `Fit` will generate a `GaSPModel` object with some summaries from the optimization:

- `objective`: the maximum value found for the objective function, i.e., the log likelihood for `fit_objective = "Likelihood"` or the log posterior for `fit_objective = "Posterior"`.
- `cond_num`: the worst condition number arising in the matrix calculations for the returned object.
- `CVRMSE`: The leave-one-out cross-validation root mean squared error.

To see the values of these parameters, or any parameter of a `GaSPModel` object, use the `$` symbol, e.g., `borehole_fit$cor_par` to access the estimated correlation parameters. The user can also type `borehole_fit` to print the whole object on the console.

Sometimes the R console will print an error matrix with the header: "The following warning/error messages were generated:". This matrix is feedback generated by the underlying `C` code. There is no need for alarm as long as there are no errors (just warnings).

6 Predict

We can use a trained `GaSPModel` to predict the response $y(\mathbf{x}')$ for untried input vectors $\mathbf{x}'$ in a test set. Specifically, we can obtain the estimated predictive means (or best linear unbiased predictors) and the predictive variances, with the latter incorporating the uncertainty inherent in a GaSP model because it is a stochastic process, as well as that from estimating the coefficients β. Uncertainty from estimating the correlation parameters is not included, however. For the details of the derivation refer to Sacks et al. (1989).

We illustrate `Predict` using the first three rows of the borehole `x_pred` data frame and the true response values `y_true` (see Section 2):

```
head(borehole$x_pred, n = 3)
         rw         r        Tu       Hu        Tl       Hl        L       Kw
1 0.1266540  4737.606  71057.92 1075.227 105.06737 781.0703 1316.775 10763.67
2 0.1345655 31551.870 113416.59 1085.278 109.88709 761.2122 1527.622 10303.86
3 0.1427761 18935.703 103624.42 1073.503  71.53184 783.3706 1414.381 12044.93
head(borehole$y_true, n = 3)
         y
1 120.3835
2 123.4973
3 155.9978
```

We apply the `Predict` function as follows:

```
borehole_pred <- Predict(
  GaSP_model = borehole_gasp,
```

```
  x_pred = x_pred,
  generate_coefficients = TRUE
)
```

Here, `borehole_gasp` is from the `GaSPModel` function in Section 4, but use of the Predict function is the same for `GaSPModel` objects generated by either the MLE and MAP training methods. The first three rows of the resulting `y_pred` data frame are:

```
head(borehole_pred$y_pred, n = 3)
      Pred        SE
1 119.6256 0.2442234
2 123.5648 0.7313592
3 156.4542 1.1220239
```

It can be seen that the estimated predictive means (`Pred`) match the true response values in `y_true` fairly well. For the MLE method, a prediction follows a normal distribution with the given predictive mean and estimated standard deviation (`SE`), whereas for the MAP method the prediction will follow a t-distribution instead. Both these results are conditional on the correlation parameters, i.e., their fitted values are taken as true.

`generate_coefficients` is an option for generating a vector of prediction coefficients `pred_coeffs`. They can be used as follows. Let $\mathbf{c}$ denote the coefficients and let $\mathbf{r}$ denote a vector with element i containing the correlation between the output at a given new point and the output at training point i. Then the prediction of the output at the new point is the dot product of $\mathbf{c}$ and $\mathbf{r}$. The vector $\mathbf{c}$ is obtained as `borehole_pred$pred_coeffs` here.

7 CrossValidate

We can apply the `CrossValidate` function to a `GaSPModel` object as follows:

```
borehole_cv <- CrossValidate(borehole_gasp)
head(borehole_cv, n = 3)
      Pred        SE
1 54.61808 1.0232074
2 55.01855 0.7859013
3 71.42804 0.7572146
```

It computes a "fast" version of leave-one-out cross-validation where the correlation parameters are not re-optimized every time a training observation is removed. The output is similar to that of `Predict`: cross-validated predictions and standard errors are in the columns `Pred` and `SE`. The first three predictions track well the training data shown in Section 2.

8 Plots and diagnostics for Predict and CrossValidate

One of the main strengths of `GaSP` is its wide variety of plots. Here we introduce diagnostics based on `Predict` and `CrossValidate`:

- `PlotPredictions`: plot the true output (response) versus predictions generated by `Predict` or `CrossValidate`.
- `PlotResiduals`: plot residuals versus each input variable.
- `PlotStdResiduals`: plot the standardized residuals versus predictions made by `Predict` or `CrossValidate`. Here, a standardized residual is the residual divided by an estimate of its standard deviation.

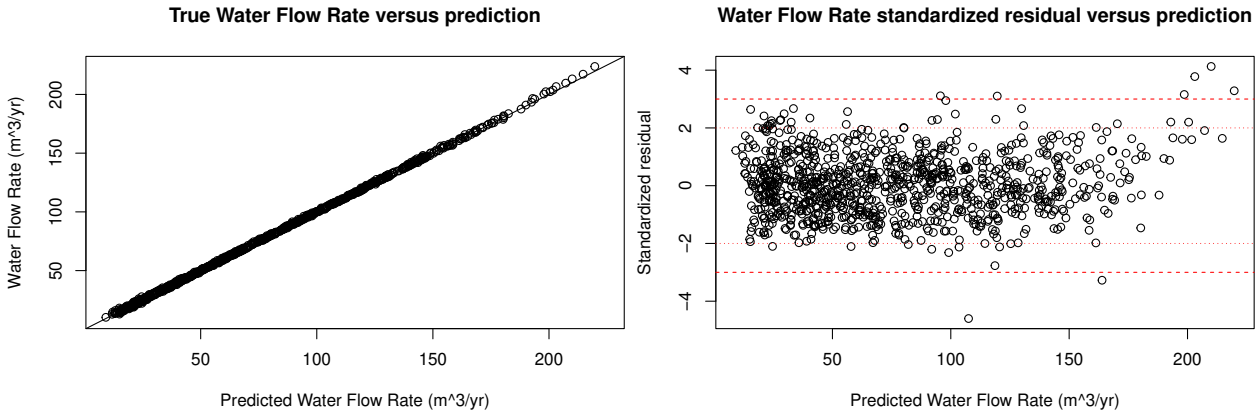
- `PlotQQ`: make a normal quantile-quantile (Q-Q) plot of the standardized residuals of predictions from `Predict` or `CrossValidate`.

The same plotting functions are used for predictions from `Predict` or `CrossValidate`, with slight differences in the setup as described in the following two sections.

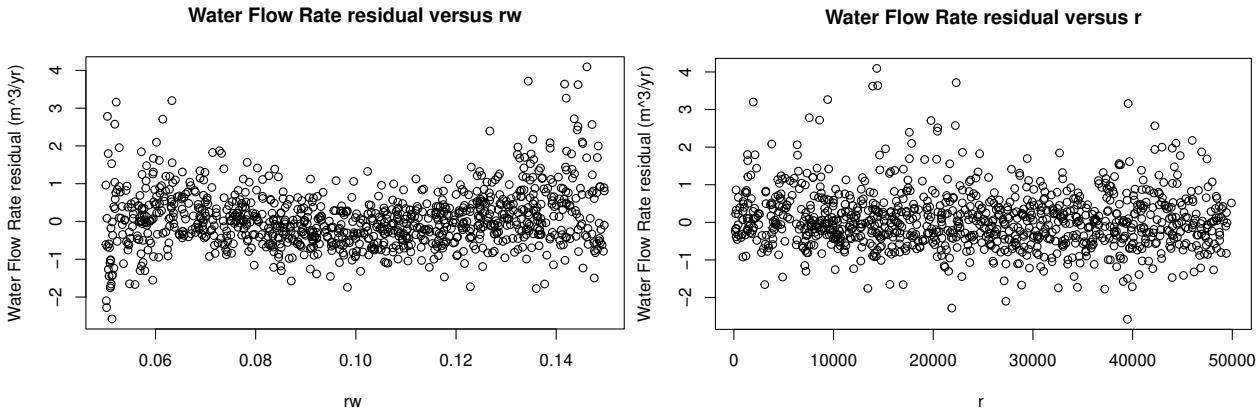
8.1 Plots for Predict

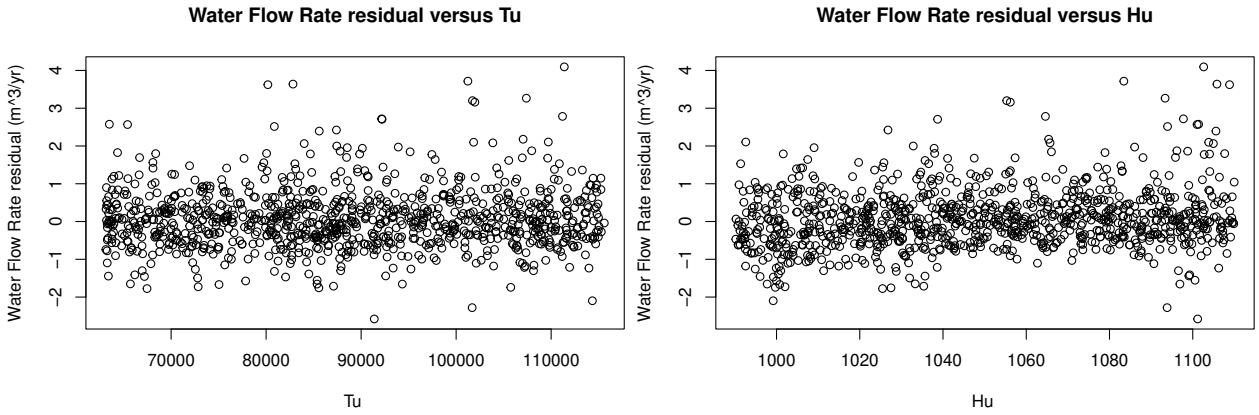
Here we showcase plots from `Predict` results:

```
PlotPredictions(borehole_pred$y_pred, y_true,
  y_name = "Water Flow Rate", y_units = "m^3/yr", title = "Predict")
PlotStdResiduals(borehole_pred$y_pred, y_true,
  y_name = "Water Flow Rate", y_units = "m^3/yr", title = "Predict")
```

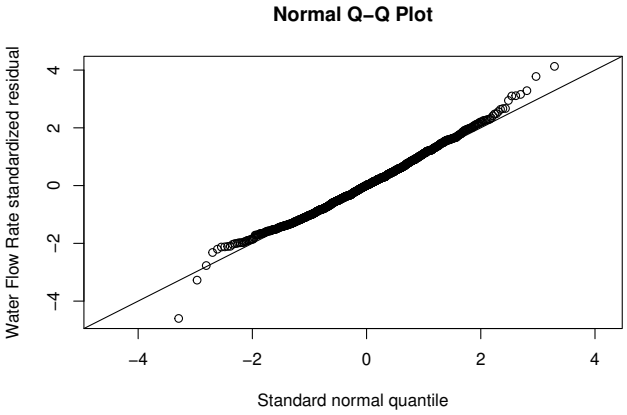



```
PlotResiduals(x_pred[, 1:4], borehole_pred$y_pred,
              y_true, y_name = "Water Flow Rate", y_units = "m^3/yr")
```






```
PlotQQ(borehole_pred$y_pred, y_true, y_name = "Water Flow Rate")
```



Note that we need to specify the `y_pred` component from the `borehole_pred` object returned by `Predict`. A value for the `title` parameter is mandatory (`"Predict"`, `"CrossValidate"`, or `""`) for the first two functions to distinguish the possible purposes. The call to `PlotResiduals` passes only the first four columns of `x_pred` here just to reduce plotting space; usually, all input columns would be of interest. The parameters `y_name` and `y_units` are only used for label construction in all these functions and do not need to be specified.

8.2 Plots for CrossValidate

The plots for predictions from `CrossValidate` are similar to those for `Predict`, and we therefore just give the syntax without showing the plots:

```
PlotPredictions(borehole_cv, y,
  y_name = "Water Flow Rate", y_units = "m^3/yr", title = "CrossValidate")
PlotStdResiduals(borehole_cv, y,
  y_name = "Water Flow Rate", y_units = "m^3/yr", title = "CrossValidate")
PlotResiduals(x, borehole_cv, y, y_name = "Water Flow Rate", y_units = "m^3/yr")
PlotQQ(borehole_cv, y, y_name = "Water Flow Rate")
```

Here `borehole_cv` can be directly used, as it is a data frame like `borehole_pred$y_pred`. And naturally for cross validation, we have `y` instead of `y_true`.

8.3 Root mean squared error (RMSE)

We also provide a function `RMSE` that calculates the root mean squared error (RMSE) of prediction or the normalized RMSE. The RMSE formula we use is

$$\sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}},$$

whereas the normalized version is

$$\sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \Big/ \sqrt{\frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n}},$$

i.e., relative to the sample standard deviation.
Here, n could be the size of the training data (the $\hat{y}_i$ are from `CrossValidate`) or the size of a test set (the $\hat{y}_i$ are from `Predict`).

For instance,

```
RMSE(borehole_pred$y_pred$Pred, y_true, normalized = FALSE)
[1] 0.7824142
RMSE(borehole_pred$y_pred$Pred, y_true, normalized = TRUE)
[1] 0.01738657
```

shows that the RMSE is small relative to the variability in the test data.

9. Visualize

Following Schonlau and Welch (2006), `GaSP` performs an analysis of variance (ANOVA) decomposition of the total function variability into contributions from main effects and 2-factor interactions, to assess how sensitive the response is to individual inputs and low-order effects. Furthermore, it can generate plotting coordinates to visualize the estimated main and 2-factor joint effects. As detailed by Schonlau and Welch (2006), the low-order effect of one or two inputs is obtained by integrating out all the other inputs. Hence, `GaSP` needs ranges of integration for all input variables. Optionally, integration over a continuous range for a given input can be replaced by summation over a grid. The first task, therefore, is to describe the input variables via `DescribeX`.

9.1 DecribeX

The user must give the input names, as well as their minima and maxima. For the borehole application we can set up the usual engineering ranges:

```
borehole_x_names <- colnames(x)
borehole_min <- c(0.05, 100.00, 63070.00, 990.00, 63.10, 700.00, 1120.00, 9855.00)
borehole_max <- c(0.15, 50000.00, 115600.00, 1110.00, 116.00, 820.00, 1680.00, 12045.00)
borehole_x_desc <- DescribeX(borehole_x_names, borehole_min, borehole_max)
```

The result is a data frame that combines these vectors:

```
borehole_x_desc
  Variable      Min       Max
1       rw     0.05      0.15
2        r   100.00  50000.00
3       Tu 63070.00 115600.00
4       Hu   990.00   1110.00
```

```
5       Tl    63.10    116.00
6       Hl   700.00    820.00
7        L  1120.00   1680.00
8       Kw  9855.00  12045.00
```

Optionally, we can include three further vectors:

- `support`: A vector of strings for additional description of the input-variable domains. Valid strings for the elements are:
  - `"Continuous"` to indicate the input is continuous over its range. This is the default assumption for all variables.
  - `"Fixed"` to indicates the input has a range of 0, i.e., its `x_min` must equal its `x_max`.
  - `"Grid"` to indicate a discrete grid for the variable, which requires the next argument.
- `num_levels`: A vector of integers for the number of levels of each input. It must be present if the `support` argument includes at least one instance of `"Grid"`. Set an input's number of levels to 0 if it is `"Continuous"`, 1 if it is `"Fixed"`, or to the appropriate number > 1 if it is `"Grid"` to define an equally spaced grid inclusive of the input's `x_min` and `x_max`.
- `distribution`: A vector of strings to define the weight distributions of the input variables. Valid strings are `"Uniform"` or `"Normal"`, which will be ignored for "Fixed" inputs. The default is `"Uniform"` for all variables.

For the default behavior, `GaSP` will perform continuos integration on all variables with respect to uniform weight. When `distribution` is `normal`, automatically the mean is the middle of the range, and the standard deviation is 1/6 of the range.

9.2 Visualize

Here, we perform `Visualize` on borehole with the default input descriptions:

```
borehole_vis <- Visualize(borehole_gasp, borehole_x_desc)
```

`Visualize` returns a list with three components: `anova_percent`, `main_effect`, and `joint_effect`. These are respectively the ANOVA percentages, the plotting coordinates for the main effects, and the coordinates for the joint effects.

The ANOVA percentages are estimates of the relative importances of low-order effects. For instance,

```
head(borehole_vis$anova_percent, n = 3)
           y
rw 82.87164
r   0.00000
Tu  0.00000
tail(borehole_vis$anova_percent, n = 3)
                 y
Hl:L  0.058709173
Hl:Kw 0.003898361
L:Kw  0.022251103
```

shows that input `rw` by itself accounts for 82.9% of the variability of the estimated prediction function over the entire 8-dimensional input space, whereas `r` as a main effect accounts for 0%. Interaction effects between two factors at a time typically have small contributions, and we see that the estimated interaction between `L` and `Kw` (denoted `L:Kw`) accounts for only 0.02% of the total variation.

The data frames `main_effect` and `joint_effect` can be quite large. Thus we can add two parameters `main_percent` and `interaction_percent` to output results only for main and joint effects that have ANOVA

percentages greater than these thresholds. Here we set `main_percent = 1` and `interaction_percent = 1`:

```
borehole_vis <- Visualize(borehole_gasp, borehole_x_desc,
                          main_percent = 1, interaction_percent = 1)
```

The main effect and joint effect data frames will look like the following:

```
head(borehole_vis$main_effect, n = 3)
  Variable.x_i  x_i        y        SE
1           rw 0.05 18.42223 0.5062615
2           rw 0.06 25.76570 0.2033376
3           rw 0.07 35.06013 0.1725205
head(borehole_vis$joint_effect, n = 3)
  Variable.x_i Variable.x_j  x_i  x_j        y        SE
1           rw           Hu 0.05  990 14.87616 1.0059070
2           rw           Hu 0.05 1002 15.41472 0.7806689
3           rw           Hu 0.05 1014 16.01475 0.6337129
```

where the first three rows are the first three plotting coordinates for the main effect of `rw` or the joint effect of `rw` and `Hu`, respectively. Typically the user will be uninterested in these large dataframes of plotting coordinates. Rather, they are plotted by special functions in the next section.

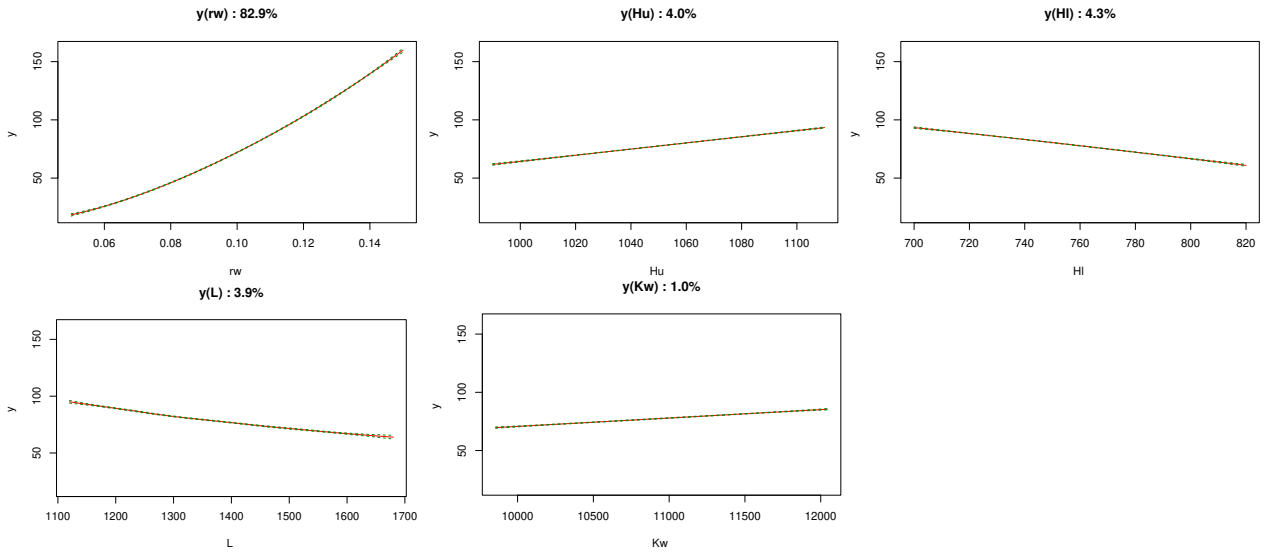
9.3 Plots for Visualize

`GaSP` also provides plotting functions for main effects and joint effects from `Visualize`:

- `PlotMainEffects`: Using the coordinates given by `Visualize`, we can generate main effect plots, with each plot showing an estimated main effect (red solid line) and point-wise approximate 95% confidence limits (green dashed lines).
- `PlotJointEffects`: Similar to `PlotMainEffects`, using the coordinates given by `Visualize`, we can make a contour plot of the two-way joint effects for two inputs at a time, as well as a contour plot of the estimated standard error.

Applying the first function to our `Visualize` results

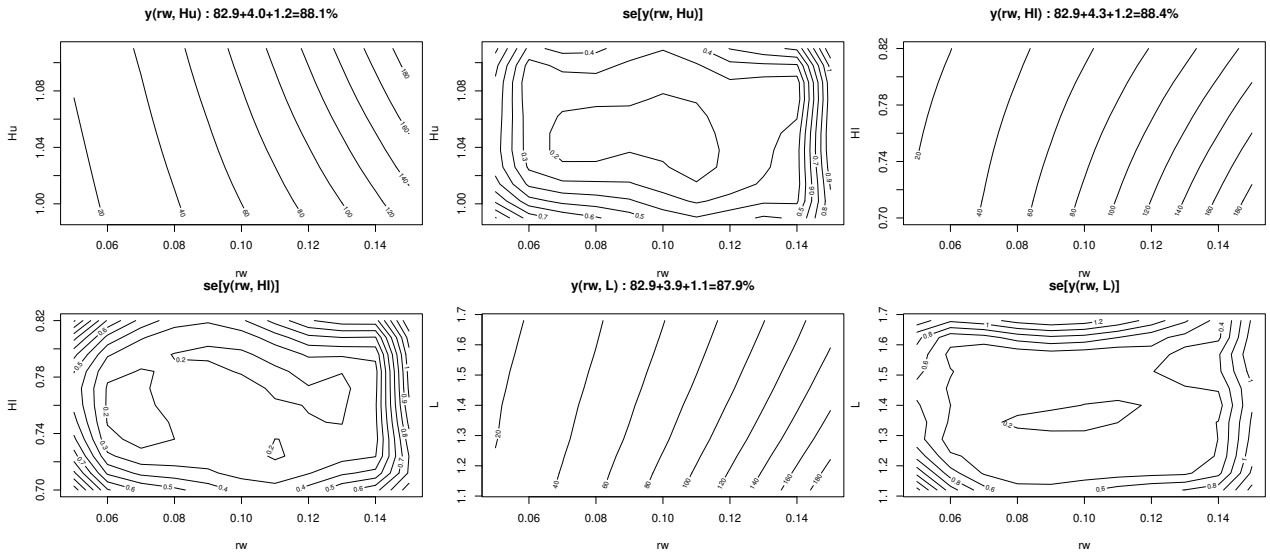
```
PlotMainEffects(borehole_vis$main_effect, borehole_vis$anova_percent)
```



we can see, for instance, in the first plot the main effect of `rw`, which is the prediction of y with the other 7 variables integrated out. There are only five main effects plotted; `r` does not appear, for example, because its ANOVA percentage did not meet the threshold of 1%. The red solid line in any of these plots is the estimated main effect, and the green dashed lines are the point wise approximate 95% confidence limits. Borehole is an easy modeling task for `GaSP`, and thus the three lines are tight.

Plotting the joint effects with

```
PlotJointEffects(borehole_vis$joint_effect, borehole_vis$anova_percent)
```



shows that only three of them satisfy the 1% interaction threshold. (If the interaction percentage is small it is sufficent to look at the respective main effects.) The first contour plot depicts the estimated joint effect of `rw` and `Hu`, i.e., the prediction as a function of these two inputs with the other six inputs integrated out. The title reports that the main effects of `rw` and `Hu` account for 82.9% and 4.0% of the total variation, respectively, with their interaction effect accounting for another 1.2%. The next plot shows the standard error of the estimated joint effect of `rw` and `Hu`. Two more pairs of plots follow, corresponding to the two further interactions exceeding the threshold.

10. PlotAll

Last but not least, if all methods are run, `GaSP` provides `PlotAll` to get a quick picture of the utility of the model. It executes `PlotPredictions`, `PlotResiduals`, `PlotStdResiduals` (all applied to CV only), `PlotMainEffects`, and `PlotJointEffects`.

```
PlotAll(borehole_gasp, borehole_cv, borehole_vis)
```

The output will be the same as directly calling each method individually, as it includes all the parameters from all the plotting functions. Note that `PlotAll` will only generate cross-validation plots. Predictions on a test set would have to be performed and assessed in a follow-up.

Thus our workflow for the borehole illustration concludes.

11. Future

`GaSP` is currently under development of version 2.0.0 and will feature full Bayesian methods that have shown to give better estimates and uncertainty quantification. There will be backwards compatibility with the version described here, however.

References

Chen, H., Loeppky, J. L., Sacks, J., and Welch, W. J. (2016). Analysis methods for computer experiments: How to assess and what counts? *Statistical Science*, 31(1):40–60.

Handcock, M. S. and Stein, M. L. (1993). A Bayesian analysis of kriging. *Technometrics*, 35(4):403–410.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and Analysis of Computer Experiments. *Statistical Science*, 4(4):409 – 423.

Schonlau, M. and Welch, W. J. (2006). *Screening the Input Variables to a Computer Model Via Analysis of Variance and Visualization*, pages 308–327. Springer New York, New York, NY.

Surjanovic, S. and Bingham, D. (2013). Virtual library of simulation experiments: Test functions and datasets. Retrieved from http://www.sfu.ca/~ssurjano.