GaSP: Train and Apply a Gaussian Stochastic Process Model

Yilin Yang and William J. Welch

2021-12-07

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-validated predictions are also available along with many model diagnostic plots such as residual plots, standardized residual plots, and normal 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 above 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 estimated by 'GaSP', 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 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 "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 set up 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 $delta_j$ 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.

In practice, `GaSP` will store the values of the correlation parameters in a datraframe named `cor_par`. `cor_par` contains one row for each term in the stochastic process model and two columns. The first is named `Theta`, and the second is either `Alpha` for the power-exponential case or `Derivatives` for the Matérn case.

- **Internal Scaling**: Recall that we previously mentioned in the setup section `GaSP` does not require manual scaling. Here is a brief explanation on how `GaSP` performs internal scaling. As the distance h_j will be on different scales for each variable, in order to operate on a $[0, 1]$ scale and use uniform restrictions for correlation parameters, `GaSP` will scale the matrix `x` to the range of $[0, 1]$. We denoted the range for variable j as r_j, and the distance functions h_j will be $h'_j = h_j/r_j$ where h'_j is the distance on the $[0, 1]$ scale. The smoothing parameters `Alpha` and `Derivatives` are not affected and do not need to be scaled, therefore we only need to scale `Theta`. We denote the θ_j as the theta for variable j on the original scale, and θ'_j as the theta for variable j on the $[0, 1]$ scale. The relationship of θ'_j and θ_j will be $\frac{\theta'_j}{r_j^{(2-\alpha_j)}} = \theta_j$ for the power-exponential case, and in the Matérn case the relationship will be $\frac{\theta'_j}{r_j} = \theta_j$ for $0, 1, 2$ derivatives, and $\frac{\theta'_j}{r_j^2} = \theta_j$ for $\delta_j \to \infty$. Therefore, there is no need to supply scaled parameters and `GaSP` will return scaled results.

3.4 Random error component

The random error term ϵ is independent "white noise" with variance σ_ϵ^2. 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 $\sigma^2 = \sigma_Z^2 + \sigma_\epsilon^2$ be the total variance, and let $\gamma = \sigma_Z^2/\sigma^2$ be the proportion of the total variance due to the stochastic process. Internally `GaSP` optimizes σ^2 and γ but reports back $\sigma_Z^2 = \gamma\sigma^2$ and $\sigma_\epsilon^2 = (1-\gamma)\sigma^2$ 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 $\sigma_\epsilon^2 = (1-\gamma)\sigma^2$. 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, the pure deterministic model with only $\sigma^2 = \sigma_Z^2$ 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 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; the 'noisy data' model where σ_Z^2 and σ_ϵ^2 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`. To combat this contradiction and to keep `GaSP` behaviour consistent, functions such as `Predict` that require a `GaSPModel` 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` class object: `Fit` and `GaSPModel`. `Fit` is used when we want to use the provided MLE or MAP methods to train a `GaSPModel` and will be detailed in section 5, and `GaSPModel` is for either loading up previously obtained results from `Fit` as to avoid retraining the model or using results from other packages to use the functions `Predict`, `CV`, and `Visualize`.

A `GaSPModel` object will have these following parameters:

- `x` and `y`: these are the input (explanatory variable) training data and output (response) training data.
- `reg_model` and `sp_model`: these are respectively the regression model and an optional stochastic process model, as specified in section 3.2 and section 3.3.
- `cor_family` and `cor_par`: these are 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`: these are respectively the boolean to indicate a random error term, the stochastic process variance and the random error variance. This is specified in section 3.4.
- `beta`: these are the correlation parameters β_j for the mean function component in section 3.2.
- `objective`, `cond_num` and `CVRMSE`: these will be only used as feedback from `Fit` and they are respectively the maximum fit objective, the condition number and the model's cross-validated root mean squared error, see section 5.

Although `Fit` is more commonly used, for directly creating a `GaSPModel` object, we do the following:

```
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
)
```

For parameter inputs,`GaSPModel` will be very similar to `Fit` as `GaSPModel` function parameter is a subset of `Fit`, which will be detailed in section 5. Here the `cor_family` will have a default set to `"PowerExponential"`, and `sp_model` will also have a default of using all variables in `x`. However, it should be noted that `sp_var` needs to be specified, `error_var` will also require specification when `random_error = FALSE`. It should be stressed from our `cor_par` setup that `cor_par` will need to have row names of variables that are in the stochastic process model, as we take the default stochastic process model here, we can just use the column names of `x` for `cor_par`. This implies that `x` should also have correct column names or the results will be difficult to interpret. `GaSP` will check for all these cases and many more with helpful warning/error messages so there is no need for manual checking. It should also be emphasized that although `GaSP` has implemented a plethora of parameter checks, it has no way of knowing if the parameters will generate a good, stable result prior to running the `C` interface. So it is up to the user when using `GaSPModel` to ensure the parameter inputs will generate the desired model.

Additionally, `GaSP` provides the option to use these values as starting points for our `Fit` method:

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

The user can set `tries = 1` to only train and evaluate the inputs.

5. Fit

Following the methods introduced by Sacks et al. (1989), in `GaSP` we specify our objective that `Fit` attempts to maximize by setting parameter `fit_objective` to `"Likelihood"` for maximum likelihood estimation or `"Posterior"` Bayesian maximum a posteriori estimation. We will outline the two methods in the next section.

5.1 Maximum likelihood estimation

To use a MLE method, we can specify our `Fit` as follows:

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

The RHS of the input `x = x` refers to the borehole `x` in section 2, and similarly for `y`. Here we choose the power-exponential correlation family with random error and the default nugget value. This function

returns a `GaSPModel` class object, and we can use this `GaSPModel` for further calculations. The parameter `model_comparison` are the criterion used to select from multiple solutions when there are multiple tries: the objective function `"Objective"` or leave-one-out cross validation `"CV"`. For the objective function of MLE, this is:

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

We operate on the log-scale for numerical stability.

5.2 Maximum a posteriori (MAP) estimation

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

The RHS of the input `x = x` refers to the borehole `x` in section 2, and similarly for `y`. Here we choose the Matérn correlation family as well as no random error and no nuggets, we will also use the MAP method. Although we didn't specify it in this model, `lambda_prior` is also a parameter users could set during the MAP method, it represents the rate parameter of an exponential prior for each `Theta` parameter. Along with the constant prior for linear model regression coefficients $\pi(\boldsymbol{\beta}) \propto 1$ and a Jeffery's prior for the stochastic process variance $\pi(\sigma^2) \propto 1/\sigma^2$, the products of these priors form the prior of the MAP method in `GaSP`.

Therefore for the objective function of MAP, we use:

$$\log p(\psi|\mathbf{y}) = \lambda\boldsymbol{\theta} - \frac{n-k}{2}\ln(\widehat{\sigma^2_{\boldsymbol{\theta}}}) - \frac{1}{2}\ln(\det(\mathbf{R})) - \frac{1}{2}\ln(\det((\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F})))$$

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 are less commonly used. Here are their explanations:

- `log_obj_tol`: An absolute tolerance for terminating the maximization of the log of the objective, it is the stopping criteria for `Fit`.
- `log_obj_diff`: The default value is 0 and will have no effect on the output. However, if set to values greater than 0, during iterations of the optimization model, an informal hypothesis test is carried out to simplify the model by trying to set `Thtea` values to 0.

In summary, here are all the parameters for `Fit` that 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 the random error.

and here are all the parameters for `Fit` that do not need to be specified if the default is correct:

- `sp_model`: default is `NULL`, all variables in `x` will be used.
- `cor_family`: default is `"PowerExponential"`, if correct, user does not need to specify this.
- `cor_par`: the default is an empty dataframe, it is NOT a legal user input, it is to communicate with the `C` interface that `cor_par` was not specified.

- `sp_var` and `error_var`: the defaults are -1, however -1 is NOT a legal user input, it is an out of bounds, arbitrary value to communicate with the `C` interface that these parameters were not specified.
- `nugget`: the default for the random error nugget is `1e-9`.
- `tries`: the default for the number of optimizations is `10`.
- `seed`: the default for random seed number is `500`.
- `fit_objective`: default is `"Likelihood"`, if correct, user does not need to specify this.
- `theta_standardized_min` and `theta_standardized_max`: These are the ranges for the theta values, and will not need to be specified.
- `alpha_min` and `alpha_max`: These are the ranges for the alpha values, and will not need to be specified.
- `derivatives_min` and `derivatives_max`: These are the ranges for the derivative values, and will not need to be specified.
- `log_obj_tol`: the default is `1e-5`.
- `log_obj_diff`: the default is `0`.
- `lambda_prior`: the default is `0.1`.
- `model_comparison`: default is `"Objective"`, if correct, user does not need to specify this.

as well as allow user inputs for the initial try. Additionally, `GaSP` allows the user to restrict legal ranges by changing the min and max values of `Theta` and `Alpha`. Therefore it is possible, for example, to only use squared-exponential for all terms by setting range parameter `alpha_max` to 0.

This list is just to get a general idea of why some variables are not specified in our examples, in practice, `GaSP` will perform thorough parameter checks, so users do not have to check these manually. The warning/error messages of these checks will look like this:

```
borehole_fit <- Fit(
  reg_model = ~ 1 + a, sp_model = ~ 1 + r, x = x, y = y, 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'.
```

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

As stated in section 4, `Fit` will generate a `GaSPModel` object with some parameters used as feedback from `Fit`:

- `objective`: The maximum value found for the objective function, this will be the log likelihood for `fit_objective = "Likelihood"` or the log posterior for `fit_objective = "Posterior"`.
- `cond_num`: The condition number.}
- `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 `$` symbol such as `borehole_fit$cor_par` to access the estimated correlation parameters, the user can also call `borehole_fit` to print the whole object in the console.

Lastly, to showcase the flexibility of our `Fit` method, and to fit a model with the least amount of parameters permitted, we create a bizarre, minimalist model as following:

```
borehole_fit <- Fit(
  reg_model = ~ 1 + r + I(r^2) + I(r^3) + I(Hu^2), x = x, y = y, random_error = FALSE
)
```

It should also be mentioned that sometimes the R console will print an error matrix with the header: "The following warning/error messages were generated:". This error matrix is feedback generated by the `C` functions. There is no need for alarm as long as there are no `R` errors shown.

6 Predict

After our model setup, we can use our trained `GaSPModel` to predict the response $y(\mathbf{x}')$ for an untried $\mathbf{x}'$. Given our `GaSPModel` object, we can obtain the estimated predictive mean (or best linear unbiased predictor) and the predictive variance that incorporates the uncertainty from estimating the coefficients $\boldsymbol{\beta}$. For the details of the derivation, refer to Sacks et al. (1989). We first showcase the first three rows of our `x_pred` data frame and the true response values `y_true`:

```
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
```

This `x_pred` is from the borehole setup, please refer to section 2 for the borehole details. Now, we can use `Predict` function as follows:

```
borehole_pred <- Predict(
  GaSP_model = borehole_gasp,
  x_pred = x_pred,
  generate_coefficients = TRUE
)
```

Here, the Predict function will be the same for both MLE and MAP methods. The only difference of these two methods are the changes in degrees of freedom, and is already factored in the stochastic process variance `sp_var`. The head for the resulting `y_pred` data frame will look as follows:

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

For the MLE method, the prediction will follow a normal distribution, and for the MAP method the prediction will follow a t-distribution instead.

`generate_coefficients` is the option for generating vector `pred_coeffs`. The `pred_coeffs` 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 for the output at the new point is the dot product of $\mathbf{c}$ and $\mathbf{r}$. The prediction `borehole_pred` is a data frame with the prediction and the standard error as columns.

7 CrossValidate

After our model setup, we can use the `CrossValidate` function as follows:

```
borehole_cv <- CrossValidate(borehole_gasp)
```

Leave-one-out cross-validation will use prediction methods identical to `Predict`. The resulting object `borehole_cv` is a data frame with the prediction and the standard error as columns. It should be emphasized that this LOOCV is a fast LOOCV, meaning we do NOT retrain the model at each iteration. This method is in some cases better than standard CV methods and more efficient.

8 Plots and diagnostics for Predict and CrossValidate

One of the main strengths of `GaSP` is its wide variety of plots, here we introduce the visualization methods for `Predict` and `CrossValidate`:

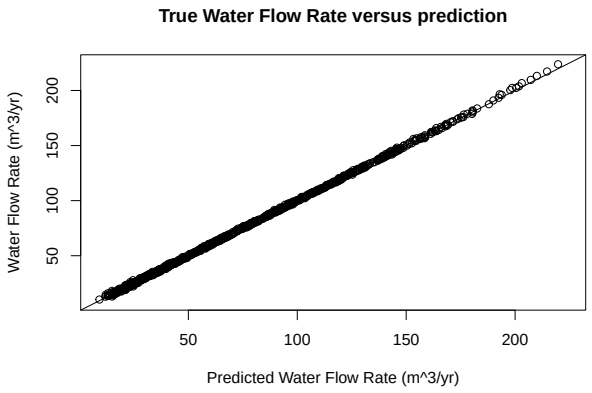
- `PlotPredictions`: plot true versus predicted output (response) made by `Predict` or `CrossValidate`.
- `PlotResiduals`: plot residuals versus each input variable.
- `PlotStdResiduals`: Plot standardized residuals versus predictions made by `Predict` or `CrossValidate`. Here, the standardized residuals will be the residuals divided by an estimate of the standard deviation of the residuals.
- `PlotQQ`: normal Q-Q plot of the standardized residuals of predictions from `Predict` or `CrossValidate`.

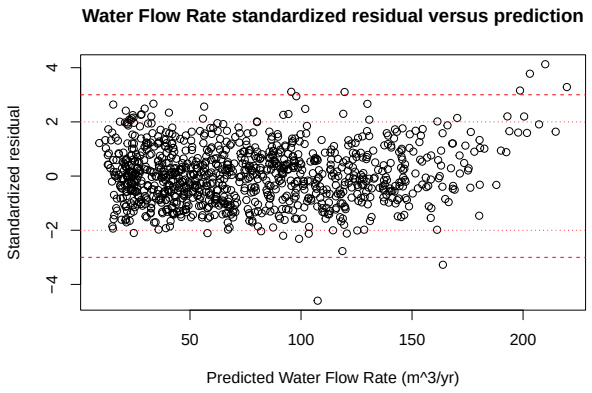
As the inputs for Plotting methods have the same parameters but different specifications for`Predict` or `CrossValidate`, we will explain this difference in the following two sections.

8.1 Plots for Predict

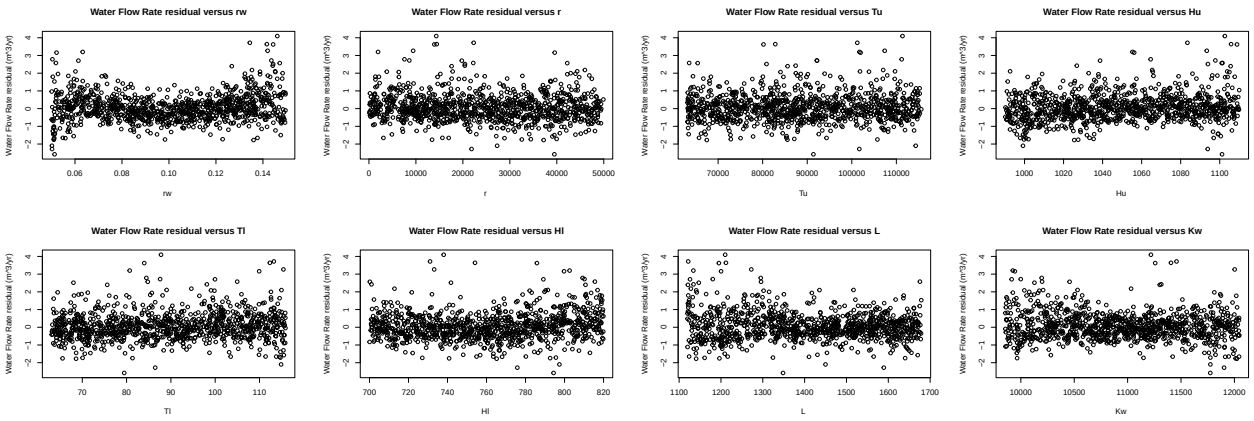
Here we will showcase plots for `Predict`:

```
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")
```

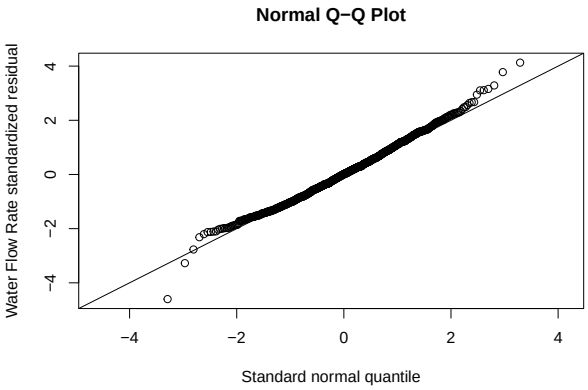





```
PlotResiduals(x_pred, borehole_pred$y_pred,
              y_true, y_name = "Water Flow Rate", y_units = "m^3/yr")
```




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



`y_true` are the true values of y for `x_pred`. Notice that we need to supply the `y_pred` field from `borehole_pred` directly, this is the case regardless of `generate_coefficients`. The parameters `y_name`and `y_units` are only used for label construction and do not need to be specified. Titles are mandatory for title construction so it must be specified.

8.2 Plots for CrossValidation

The plots for `CrossValidation` will be similar to that of `Prediction` Plots, therefore we will not show 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 the `borehole_cv` can be directly used as it has the same setup as `borehole_pred$y_pred`. And logically, we have `y` instead of `y_true`.

8.3 RMSE

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

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

and for the normalized RMSE (NRMSE):

$$\text{mean}(RMSE) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \mu)^2}{n}}$$

$$NRMSE = \frac{RMSE}{\text{mean}(RMSE)}$$

Here, μ is mean for the true values of the response y. We calculate the RMSE values for our predictions as following:

```
RMSE(borehole_pred$y_pred$Pred, y_true)
```

9. Visualize

Following Schonlau and Welch (2006), `GaSP` performs an analysis of variance (ANOVA) decomposition of the total function variability as well as use plotting coordinates to generate plots for its estimated main and two-factor joint effects. In order to do so, `GaSP` needs the ranges for all input variables, hence we first have to describe the input variables via `DescribeX`.

9.1 DecribeX

```
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)
```

Here the user describes the variable names, as well as the variables' minimum and maximum values. The output will be a dataset 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
```

Additionally we can include three optional vectors as following:

- `support`: An optional string vector for additional description of the input variables. Valid strings for the elements are:
  - `"Continuous"`: indicates the variable is continuous between it's range. This is the default assumption for all variables.
  - `"Fixed"`: indicates the variable has a range of 0, therefore it's `x_min` must equal `x_max`.
  - `"Grid"`: indicates a discrete grid on a variable, and requires the next argument.
- `num_levels`: An optional vector of integers for the number of levels of each input, must be present if the `support` argument includes `"Grid"`. An input's number of levels is 0 if it is `"Continuous"`, 1 if it is `"Fixed"`, or > 1 if it is `"Grid"` to define an equally spaced grid inclusive of the input's `x_min` and `x_max`.
- `distribution`: An optional string vector to define the weight distributions of the input variables. Valid strings are `"Uniform"` or `"Normal"`, they will be ignored for "Fixed" inputs. The default values are `"Uniform"` for all variables.

For the default behavior, `GaSP` will perform integration on all variables w.r.t uniform weight. The user can specify `distribution` to use a normal weight instead.

9.2 Visualize

Here, we perform the default `Visualize` on borehole:

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

`Visualize` will have three main fields: `anova_percent`, `main_effect`, and `joint_effect`. These are respectively the ANOVA percentages, the coordinates for the main and joint effects.

The data frame for `main_effect` and `joint_effect` can be quite large, thus we have added two parameters `main_percent` and `interaction_percent` to only output main and joint effects that have ANOVA percentages greater than the threshold. Here we set `main_percent = 1` and `interaction_percent = 1`:

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

Thus the main effect and joint effect data frame 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
```

These are the coordinates for our visualization plots, which we will detail in the next section.

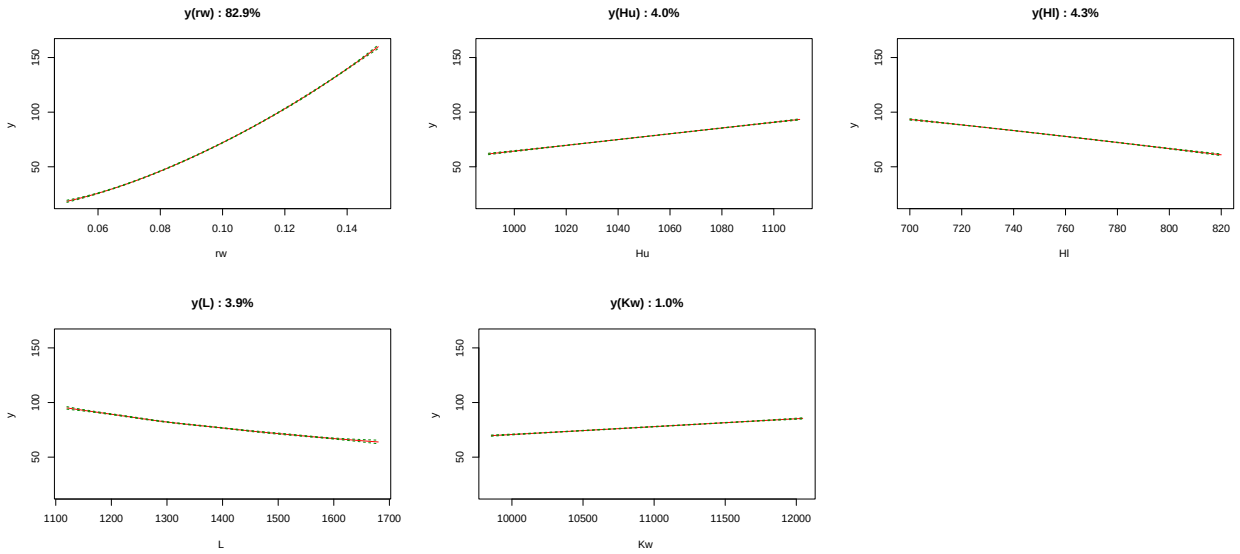
9.3 Plots for Visualize

`GaSP` also provides plot functions for main effects and joint effects for `Visualise`:

- `PlotMainEffects`: Using the coordinates given by `Visualize`, we can generate main effect plots, each plot shows an estimated main effect (red solid line) and point wise approximate 95% confidence limits (green dashed line).
- `PlotJointEffects`: Similar to `PlotMainEffects`, using the coordinates given by `Visualize`, we can plot two-way joint effects on a contour plot for their estimated joint effects, as well plot the contour plot for their estimated standard errors.

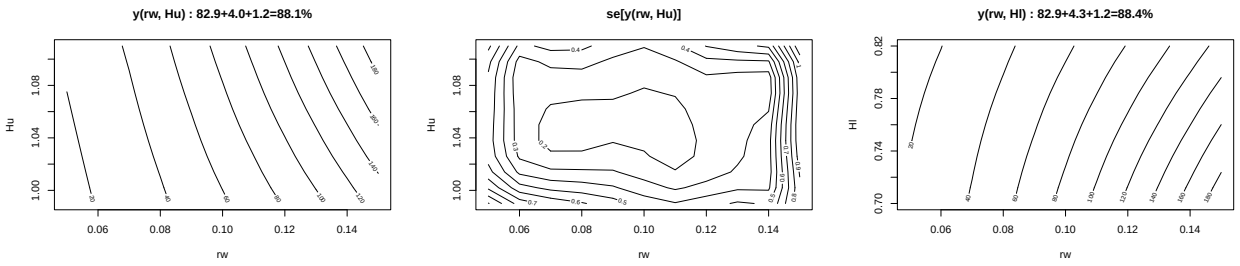
Using these two methods on our `Visualization` results:

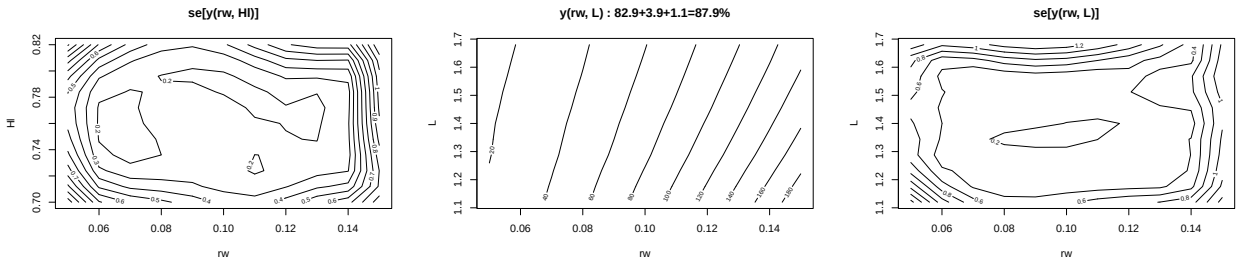
```
PlotMainEffects(borehole_vis$main_effect, borehole_vis$anova_percent)
```



Here we can see, for instance, 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, for the variables that have ANOVA percentages over the threshold we set, 1%. The red solid line is the estimated main effect and the green dashed line is the point wise approximate 95% confidence limits. Borehole is an easy modeling task for Gaussian Stochastic Processes and thus the three lines are tight.

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





Here are the two-way joint effects on a contour plot for their estimated joint effects, as well as the contour plot for their estimated standard errors. There are 3 pairs of plots for the three inputs that have an ANOVA percentage over the threshold we set, 1%. The contours in the first plot depict 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. Contours of the standard error of this joint effect is shown in the next plot. Similarly the remaining joint effects.

10. PlotAll

Last but not least, if all methods were ran, `GaSP` provides a method `PlotAll` to generate the plots all at once:

- `PlotAll`: a function to execute `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 the cross validation plots.

Thus our workflow for a borehole function 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. A complete revision of old functions will allow users with even more flexibility.

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.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and Analysis of Computer Experiments.

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.