GaSP: Train and Apply a Gaussian Stochastic Process Model (Draft)

Yilin Yang, William J. Welch

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 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 $R(x, x')$, it quantifies the relationship of the two response variables y and y' from the input vectors x and 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 3.1.3, 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 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 6.

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.

This 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

For *GaSP*, the setup is very straightforward, the training dataframe X and y do NOT need to be scaled manually as *GaSP* will perform scaling internally. The internal scaling will be detailed in section (3.1.3).

GaSP also has no constraint on the size of the inputs. As all intense mathematical calculations are dealt with in `C` code, the package will have no `R` overhead, making *GaSP* as fast as it can be. Although only `dataframes` are mentioned, `matrix` types are also allowed via `as.data.frame`.

We will use the borehole function 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
```

The `x_pred` is a dataset of the untried runs that we want to predict to get a scalar response y vector `y_pred`. For evaluations, we have provided `y_true` to plot prediction plots as well as to calculate RMSE values. For more details of the borehole function, please refer to the website in the citations by Surjanovic and Bingham (2013). Please note that *GaSP* does not handle missing values and will not give meaningful results if there are missing values in the inputs, it is the users responsibility to handle such cases.

3. Gaussian Stochastic Process Model Formulation and GaSP-Model Object

In *GaSP*, functions communicate through the `GaSPModel` object, which contains all the relevant information about a Gaussian Stochastic Process Model. There is no need for users to manually set up a `GaSPModel` object as `Fit` and `GaSPModel` will handle the object creation and will feature many parameter checks. Here, we will go into detail about the model formulation, as well as how to interpret the key components of a `GaSPModel` object.

3.1 Gaussian Stochastic Process Model Components

Following the Gaussian Stochastic Process Model introduced by Sacks et al. (1989), *GaSP* defines the Gaussian Stochastic Process model as follows:

$$y(\mathbf{x}) = \mu(\mathbf{x}) + Z(\mathbf{x}) + \epsilon \tag{1}$$

Here the scalar response $y(\mathbf{x})$ is the realization of a stochastic process $Z(\mathbf{x})$, mean function $\mu(\mathbf{x})$ and random error ϵ. We will explain each component in the following sections.

3.1.1 Mean Function Component

We can write the mean function as:

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

Here, the β_j are the linear model regression coefficients to be estimated and $f_j(x)$ are the user defined regression functions. In a `GaSPModel`, the mean function formula is specified by parameter `reg_model`. The model specification is very similar to the `formula` parameter in `lm`, however we put restrictions on the mean function formula to only allow for polynomial formulas and interaction terms. To specify a polynomial function, we must wrap the variable in `I()`, for example, to invoke a second degree polynomial function for a variable x, we add `I(x^2)` to the formula. Additionally, there will be no need to specify the response variable on the LHS of the formula as we use parameter `y` instead. For example:

```
reg_model_standard <- ~ 1 + r + rw + Tu
```

is a regression model with the first 3 columns in borehole x as predictors. Mathematically, we can express this model as:

$$\mu(\mathbf{x}) = \beta_0 + \beta_{\mathrm{r}} x_{\mathrm{r}} + \beta_{\mathrm{rw}} x_{\mathrm{rw}} + \beta_{\mathrm{Tu}} x_{\mathrm{Tu}}$$

Our definitions can also allow us to do bizarre models such as:

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

Therefore, without the stochastic process $Z(\mathbf{x})$, *GaSP* will train a polynomial linear model instead.

3.1.2 Stochastic Process Component

The random process $Z(\mathbf{x})$ is assumed to have zero mean and covariance $\sigma^2 R(x, x')$. The σ^2 is the stochastic process variance which will be estimated in the random error component section. The function $R(x, x')$ quantifies correlations between $Z(x)$ and $Z(x')$. Suppose x and x' are two different trials, then $R(x, x')$ is defined as a product of all special weighted distance functions:

$$R(x, x') = \sum_{j=1}^{k} \beta_j R_j(h_j)$$

Where $h_j = |x_j - x'_j|$ and $R_j(h_j)$ is a distance function for feature j in the stochastic process, this is also known as the correlation function. There are many correlation-function families, and the two families that *GaSP* has implemented will be explained in the next section.

In *GaSP*, the stochastic process formula is specified by parameter `sp_model`. For the default `sp_model = NULL`, *GaSP* will use a stochastic process model that contains every term in the training x data frame, additionally there will be no constant term in the stochastic process model. Similar to `reg_model`, interaction terms and polynomial functions are still allowed with the same specifications. Once again, there will be no need to specify the response variable term on the LHS of the formula as we use parameter `y` instead. For example we can specify a bizarre stochastic process model as:

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

this is extremely similar to `reg_model_bizarre`, but we do not need to specify the intercept term, a warning will be generated if the intercept term is specified, but no need for alarm.

3.1.3 Correlation Families

In this section we introduce the two correlation families *GaSP* has implemented:

- **Power-exponential**: we define the power exponential correlation function as follows:

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

  $R_j(h_j)$ here depends on a distance-scale parameter θ_j, controlling the rate of correlation decay as the distance h_j increases. Here $\theta_j \geq 0$. Different from other packages, the θ_j is in the numerator, thus $\theta_j = 0$ will imply a perfect correlation, making the variable an inactive input. $R_j(h_j)$ also has a

smoothness parameter α_j, with $\alpha_j = 2$ being extremely smooth. It is worth emphasizing that α_j is different from some definitions and has the valid range of $0 \leq \alpha_j \leq 1$. The $\alpha_j = 0$ case is known as the squared-exponential (Gaussian) correlation. The $\alpha_j = 1$ case is known as the exponential correlation. Power-exponential is a generalization of these correlations and is much more flexible. *GaSP* will estimate these terms 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.

- **Matérn**: The matern function is defined as:

$$R_j(h_j) = \frac{2^{(1-\nu_j)}}{\Gamma(\nu_j)} \left(\sqrt{2\nu_j} \frac{h_j}{\rho} \right)^{\nu_j} K_{\nu_j} \left(\sqrt{2\nu_j} \frac{h_j}{\rho} \right) \tag{3}$$

Here, the Γ is the gamma function and the K_{ν_j} is the modified Bessel function of the second kind with parameter ν_j. ρ is a sensitivity parameter, and ν_j controls smoothness. In *GaSP*, for consistency with the power-exponential for the exponential and squared-exponential special cases common to both, we define distance-scale parameter $\theta_j = \frac{\sqrt{2\nu_j}}{\rho}$ and the smoothness parameter $\delta_j = \frac{1}{2} - \nu_j$ with only 4 discrete levels of smoothness (number of derivatives). Thus consequently the Matérn function will be simplified into four cases:

$$R_j(h_j) = \begin{cases} \exp(-\theta_j h_j) \text{ for } \delta_j = 0 \text{ (the exponential correlation)} \\ \exp(-\theta_j h_j)(\theta_j h_j + 1) \text{ for } \delta_j = 1 \\ \exp(-\theta_j h_j)((\theta_j h_j)^2/3 + \theta_j h_j + 1) \text{ for } \delta_j = 2 \\ \exp(-\theta_j h_j^2) \text{ for } \delta_j \to \infty \text{ (the squared-exponential correlation)} \end{cases} \tag{4}$$

It should be noted that $\delta_j \to \infty$ is defined as $\delta_j = 3$ in *GaSP*. Similar to the power exponential case, *GaSP* allows the user to restrict legal ranges by changing the min and max values of `Theta` and `Derivatives`. Therefore it is possible to use the special cases of power exponential and squared exponential instead. Although the parameterization of θ is not the usual, but the one we've chosen implies that θ has the same interpretation as power exponential for the two special cases.

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. 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.

3.1.4 Random Error Component

The random error ϵ denotes the total variance due to random error and has two components, the stochastic process variance σ^2 and error variance named as `sp_var` and `error_var` in *GaSP*. The error variance is the

random error from measurements or white-noise. We use a boolean value `random_error` to indicate if *GaSP* optimizes random error, it should be stressed that it does not mean there will be no error. As *GaSP* also includes a `nugget` term, which is a small amount of error such as 10^{-9} that may improve numerical stability. Since the user has the ability to set the `nugget = 0`, therefore along with the boolean `random_error`, we will have four cases: We use the default `nugget = 1e-9` for cases where `nugget > 0`.

- `random_error = FALSE` and `nugget = 0`: the estimate of the proportion of the total variance due to the stochastic process will be 1.
- `random_error = FALSE` and `nugget > 0`: the proportion of the total variance due to the stochastic process will be $1 - 10^{-9}$.
- `random_error = TRUE` and `nugget = 0`: the estimate of the proportion of the total variance due to the stochastic process will be unbounded during optimization.
- `random_error = TRUE` and `nugget > 0`: The estimate of the proportion of the total variance due to the stochastic process will be upper bounded by $1 - 10^{-9}$ 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* behavior consistent, functions that require a `GaSPModel` input will generate a warning assuming `random_error = TRUE` in these cases. There is no need to be alarmed, as these functions simply will use `random_error = TRUE` instead.

3.2 GaSPModel object

Using our previously specified model formulation, if we have relevant data, we can set up a `GaSPModel` object with `GaSPModel` function. Here, we will focus less on the `GaSPModel` function specifications as it will be detailed in the following section, and more on the object itself. 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 3.1.1 and section 3.1.2.
- `cor_family` and `cor_par`: these are respectively the correlation family and the data frame containing the correlation parameters as specified in section 3.1.3.
- `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 3.1.4.
- `beta`: these are the correlation parameters β_j for the mean function component in section 3.1.1.
- `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.

4. Fit and GaSPModel

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 `GaSPModel` is for either loading up previously trained or manual inputs, as to avoid retraining the model.

4.1 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.

4.1.1 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"
)
```

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:

$$-\frac{n}{2}\ln(2\pi)-\frac{n}{2}\ln(\widehat{\sigma^2})-\frac{1}{2}\ln(\det(\mathbf{R}))-\frac{n}{2} \tag{5}$$

We operate on the log-scale for numerical stability.

4.1.2 Maximum A Posteriori Estimation

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

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` (TODO: scaling of lambda need to be checked)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:

$$\log p(\psi|\mathbf{y}) = \lambda\boldsymbol{\theta} - \frac{n-k}{2}\ln(\widehat{\sigma^2_{\boldsymbol{\theta}}}) - \frac{1}{2}\ln(\det(\mathbf{R})) - \frac{1}{2}\ln(\det((\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F}))) \tag{6}$$

4.1.3 Further Mentions for Fit

In both methods we did not address two parameters `log_obj_tol` and `log_obj_diff`. 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` and whether they need to be specified:

- `x` and `y`: the user has to specify the input matrices.
- `reg_model`: the user has to specify the mean function.
- `sp_model`: default is `NULL`, all variables in `x` will be used.
- `random_error`: the user has to specify the random_error.
- `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 is `1e-9`, user does not need to specify this.
- `tries`: the default is `10`, user does not need to specify this.
- `seed`: the default is `500`, user does not need to specify this.
- `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`, user does not need to specify this.
- `log_obj_diff`: the default is `0`, user does not need to specify this.
- `lambda_prior`: the default is `0.1`, user does not need to specify this.
- `model_comparison`: default is `"Objective"`, if correct, user does not need to specify this.

This list is just to get a general idea of our examples, and why some variables are not specified, 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.

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.

4.2 GaSPModel

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`, here the `cor_family` will still 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. Predict and CrossValidate

Following the creation of our `GaSPModel` object, we can do predictions, leave-one-out cross validation and visualizations. As visualizations need additional setup, here we use the `borehole_gasp` object generated from `GaSPModel` to carry out the other two functions `Predict` and `CrossValidate`.

5.1 Predict

After our model fit, we can use our trained `GaSPModel` to predict the response $y(x')$ for an untried 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 head for our `x_pred` data frame:

rw	r	Tu	Hu	Tl	Hl	L	Kw
0.1266540	4737.606	71057.92	1075.227	105.06737	781.0703	1316.775	10763.67
0.1345655	31551.870	113416.59	1085.278	109.88709	761.2122	1527.622	10303.86
0.1427761	18935.703	103624.42	1073.503	71.53184	783.3706	1414.381	12044.93
0.0903671	23941.571	90453.57	1085.738	76.65859	739.9306	1481.283	11515.26
0.0890826	12831.885	106440.29	1015.181	91.33051	819.8423	1549.836	10158.17
0.1053926	20380.009	110384.72	1030.292	71.37192	796.3017	1636.683	11236.76

`x_pred` will be similar to our input `x`, and it must have the same column names and the same column ordering as `x`. 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:

Pred	SE
119.62558	0.2442234
123.56481	0.7313592
156.45423	1.1220239
68.50243	0.3676312
32.47294	0.5536124
55.94849	1.0749414

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.

5.2 CrossValidate

We can use `CrossValidate` function as follows:

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

Leave-one-out cross-validation will use prediction methods identical to `Predict`. The result `borehole_cv` is a data frame with the prediction and the standard error as columns. However, 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.

5.3 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`:

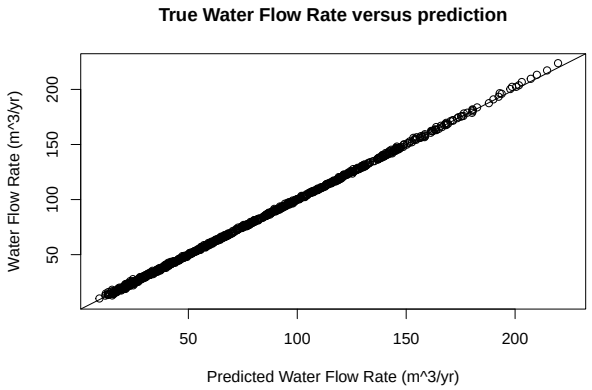
- `PlotPredictions`: plot true versus predicted output (response) made by `Predict` or `CrossValidate`.
- `PlotResiduals`: plot residuals versus each input variable.
- `PlotStdResiduals`: Plot standardized residuals versus predictions made by `Predict` or `CrossValidate`. Here, the standardized residuals 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`.

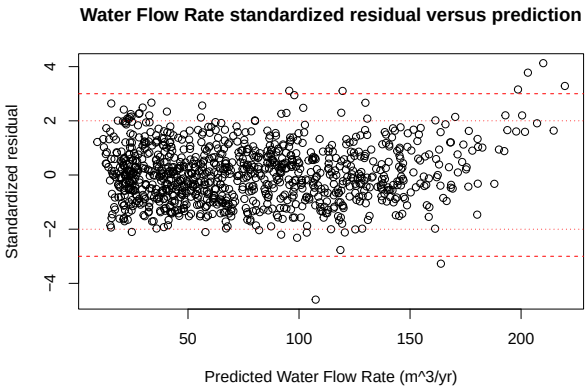
As the inputs for Plotting methods have the same parameters but different specifications for`Predict` or `CrossValidate`, we will explain this difference in the following two sections.

5.3.1 Plots for Predict

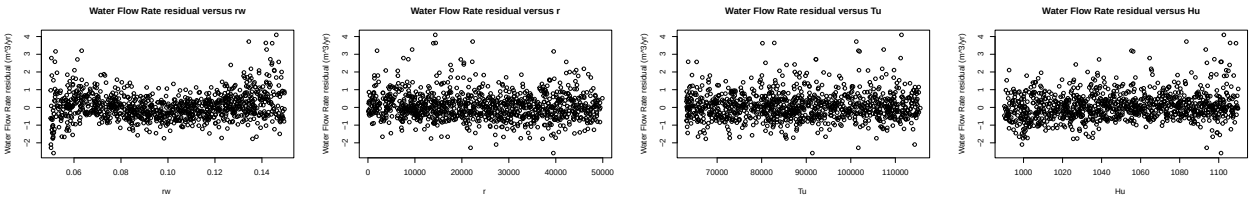
Here we will showcase plots for `Predict`:

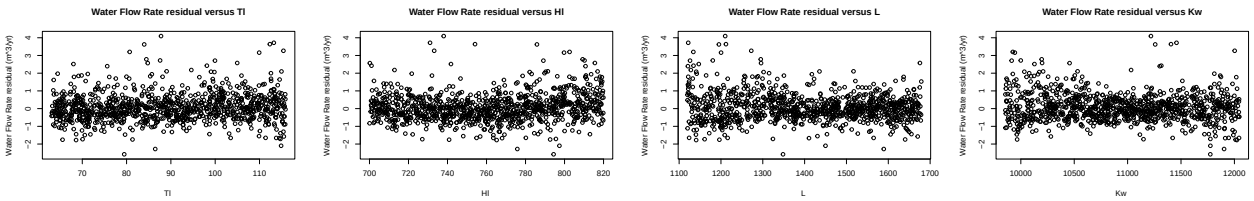
```
PlotPredictions(borehole_pred$y_pred, y_true,
  y_name = "Water Flow Rate", y_units = "m^3/yr", title = "Predict")
PlotStdResiduals(borehole_pred$y_pred, y_true,
  y_name = "Water Flow Rate", y_units = "m^3/yr", title = "Predict")
```



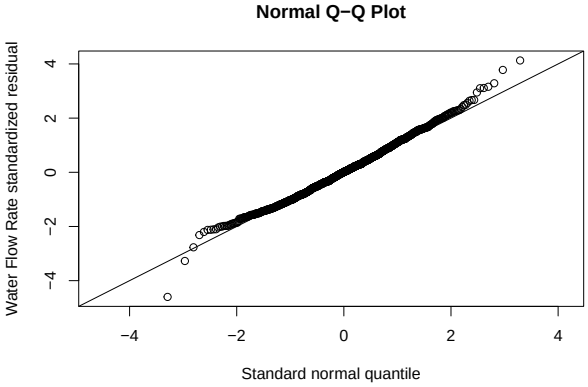



```
PlotResiduals(x_pred, borehole_pred$y_pred,
              y_true, y_name = "Water Flow Rate", y_units = "m^3/yr")
```






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



`y_true` is 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.

5.3.2 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`.

5.3.3 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)
```

6. 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`.

6.1 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:

Variable	Min	Max
rw	0.05	0.15
r	100.00	50000.00
Tu	63070.00	115600.00
Hu	990.00	1110.00
Tl	63.10	116.00
Hl	700.00	820.00
L	1120.00	1680.00
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.

6.2 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` will 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 = 20` and `interaction_percent = 1`:

```
borehole_vis <- Visualize(borehole_gasp, borehole_x_desc, main_percent = 20, interaction_percent = 1)
```

Thus the main effect and joint effect data frame will look like the following:

Variable.x_i	x_i	y	SE
rw	0.05	18.42223	0.5062615
rw	0.06	25.76570	0.2033376
rw	0.07	35.06013	0.1725205
rw	0.08	46.01326	0.1757075
rw	0.09	58.39714	0.1850701
rw	0.10	72.07113	0.1676012
rw	0.11	86.98572	0.1640696
rw	0.12	103.16630	0.2255259
rw	0.13	120.67941	0.2363277
rw	0.14	139.58675	0.2043894
rw	0.15	159.89451	0.8503547

Variable.x_i	Variable.x_j	x_i	x_j	y	SE
rw	Hu	0.05	990	14.87616	1.0059070
rw	Hu	0.05	1002	15.41472	0.7806689
rw	Hu	0.05	1014	16.01475	0.6337129
rw	Hu	0.05	1026	16.67569	0.5618602

Variable.x_i	Variable.x_j	x_i	x_j	y	SE
rw	Hu	0.05	1038	17.39691	0.5439814
rw	Hu	0.05	1050	18.17770	0.5504929

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

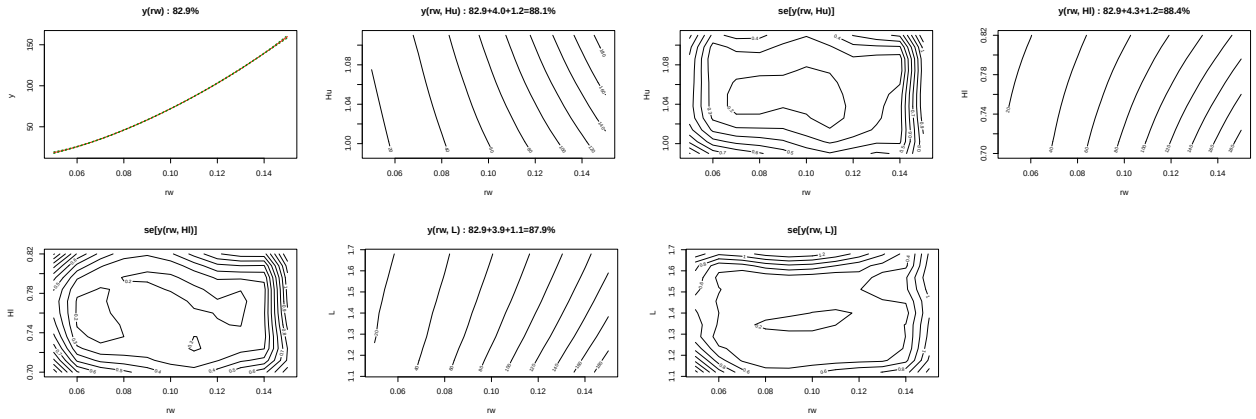
6.3 Visualise Plots

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



(TODO: add some remarks?)

7. 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.

8. 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

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.