FBC: Full Bayesian Calibration

Contents

Introduction

`FBC`, which stands for Full Bayesian Calibration, is a R software package that provides two main functionality: calibration of a simulator model using limited data from field experiment and computer simulation, and calibrated prediction of field response for a new input configuration using the calibration model. The goal of building calibration model is to make inference about calibration parameters, which in turn enables calibrated prediction for new input configurations. The first functionality is provided by `calibrate()` function that takes in the data from both field experiment and computer simulation. It returns a matrix representing a sample from the joint posterior distribution of all model parameters: calibration parameters and the hyperparameters introduced to build the calibration model. The second functionality is provided by `predict()` function that takes in the calibration model object (the output matrix produced by `calibrate()`), and the new field input configurations to perform a calibrated prediction of field response.

The `FBC` package is a based on the well-known Kennedy and O'Hagan (KOH) calibration model (2001). In the KOH model, both field response and simulator output are modelled using a stochastic Gaussian Process (GP) that emulates the expensive computer simulation. Moreover, for the field response two additional corrections are made: the inherent bias in the simulator output is corrected using another independent GP, and a Gaussian random error represents the measurement error. For simulator response, these additional terms are considered to be zero. Using two independent GPs and a Gaussian random error introduces new hyperparameters to the calibration model that must be estimated along with calibration parameters.

As the name suggests, `FBC` employs a full Bayesian approach to estimate all parameters, but neither KOH's original formulation nor later suggestions are fully Bayesian, largely due to computational infeasibility (Kennedy & O'Hagan, 2001; Higdon et al., 2004; Liu et al., 2009). Each column of the output matrix represents marginal distribution of a parameter in the model and can be used to make inference about it. In particular, `FBC` uses centrality measures such as mean, median, or mode to provide point estimates of the parameters, and computes empirical quantiles to quantify the uncertainty in the estimations. The full Bayesian approach in calibrated prediction also leads to a distribution of calibrated predictions for each input configuration, rather than a single point estimate. Hence, the Bayesian approach again enables both prediction and un certainty quantification. Implementation of the package optimizes the calibration process and memory management to increase computational efficiency. Moreover, the fully implemented Bayesian framework, enables the user to apply the expert knowledge about any of the model parameters through prior specifications.

The current vignette is structured into following sections: The first section, , explains the package functionality through a simple pedagogic example. Also in this section, the notation for inputs and outputs of both computer and physical experiments are introduced. Throughout this section, the user is expected to be well-versed with calibration models in general and the KOH model in particular. Less experienced users are encouraged to read section two, three, and four that aim to introduce the calibration model, parameter estimation, and calibrated predictions and in more details. In the second section, *Calibration Model*, generalizes the example introduced in the first section to build model components and shows how a calibration model based on KOH model is built internally. Using the general notation, while referencing the example, modelling choices are also justified in this section. In the third section, *Parameter Estimation*, the theoretical results to characterize the posterior distribution of model parameters are presented along with the numerical procedure to implement the full Bayesian estimations. In the fourth section, *Calibrated Prediction*, the theoretical results to derive calibrated predictions for new input configuration using the estimated parameters are presented. In the fifth section, *Implementation*, some of the implementation features and choices of `FBC` package are explained along with its limitations. In the sixth and last section, *Applications*, three more examples are presented to demonstrate the full functionality and limitations of the `FBC` package. Experienced users may use this section after the first section, *Using `FBC`*, to further examine the functionalities and limitation of the package. And finally, the *Appendix* provides further details to some of the concepts presented in the sections.

1. *Using FBC*

Apart from the two main public functions, `calibrate()` and `predict()`, `FBC` package has four more public functions, `set_hyperPriors()`, `plot()`, `print()`, and `summary()`, to help with prior specifications, visualizations, and summarizing of calibration models. This section explains how to use these functions and what to expect as results using a simple pedagogic example.

1.1 Setup

The simplest and safest method to obtain `FBC` package is through CRAN using following command.

```r
install.packages(FBC)
```

Alternatively, the development version of the `FBC` package can be installed directly from Github using `devtools` package. Note that, current vignette is based on published version of the package on CRAN and the development version may contain further functionalities.

```r
devtools::install_github("parpishro/FBC")
```

After installing the package, it must be loaded into the R session.

```r
library(FBC)
```

1.2 Data

Building a calibration model requires data from both field experiment and computer simulation [1]. To focus on the functionality of the package, we use a simple pedagogic example as experimental setting. In the field

[1] *Field experiments, which are sometimes called physical experiments in other texts, will be represented by f subscript (for field) and computer simulations, which are sometimes called computer experiments in other texts, will be represented by s subscript (for simulation) throughout this vignette.*

setting, a wiffle ball is dropped from different heights and the time it takes to hit the ground is measured. The experimental input is height (h). and the response is time (t). In the simulation setting, the physical process of the falling ball is modelled by a mathematical equation that relates h to t, however, one must also provide gravity (g) as input. Note that in the field experiment, the earth's gravity is fixed but unknown to the experimenter [2], but in the simulation it is an unknown parameter and must be estimated.

$$t = \sqrt{\frac{2h}{g}}$$

In general, these parameters are called calibration parameters and represent unknown but fixed physical properties that are governed by the physical system in field experiments (and therefore need not be specified in field experiments) or represent various aspects of the simulation model such as tuning hyperparameter of the model (that are not present in the field setting). In both cases, the calibration parameters must be estimated for a simulator model to mimic the physical system adequately. The goal of the calibration is to estimate calibration parameters by fitting a statistical model to the observed data using both field and simulator data. Estimated parameters of the calibration model can be used for inference or calibrated prediction.

The field experiment has been performed by Derek Bingham and Jason Loeppky and it is provided by Robert Gramacy in his book "surrogates" [3]. The mathematical model used in simulation of the ball drop experiment can be easily implemented. The simulation code takes h and g are taken as experimental and calibration inputs and returns t as simulator response based on above mathematical equation. A common method to choose the different combination of the inputs from acceptable ranges is Latin Hypercube Sampling (LHS) method (McKay et al., 1979) [4].

The `FBC` package requires the training data to be in matrix format, where for both matrices, the first column always represents response vector and following columns represent experimental inputs (for both data matrices) and calibration inputs (only for simulation data matrix). The data matrices for ball example are produced in advance and loaded into the package environment under the name of `ballField` and `ballSim` [5]. Below, the structures and dimensions of both data matrices can be inspected.

```
head(ballField, 3)
>         t     h
> [1,] 0.27 0.178
> [2,] 0.22 0.356
> [3,] 0.27 0.534
dim(ballField)
> [1] 63  2
head(ballSim, 3)
>          t     h      g
> [1,] 0.404 0.998 12.220
> [2,] 0.487 0.940  7.909
> [3,] 0.450 0.886  8.736
dim(ballSim)
> [1] 100   3
```

Figure 1 shows the distribution of time versus height for both physical and simulation experiments. Note that for higher height values, the simulation responses (blue) underestimate their corresponding field response (red), displaying a systemic bias for simulation model.

[2] *Although there are relatively precise estimates of earth's gravity, namely* $9.81m/s^2$*, it is still an estimate and subject to some level of uncertainty.*

[3] *The raw data can be downloaded from here*

[4] *To produce the design matrix,* `lhs` *R package (Rob Carnell, 2022) is used. For more information on LHS method please click here and for information on* `lhs` *package please click here.*

[5] *The preprocess code that produces* `ballField` *and* `ballSim` *from raw data can be found here.*

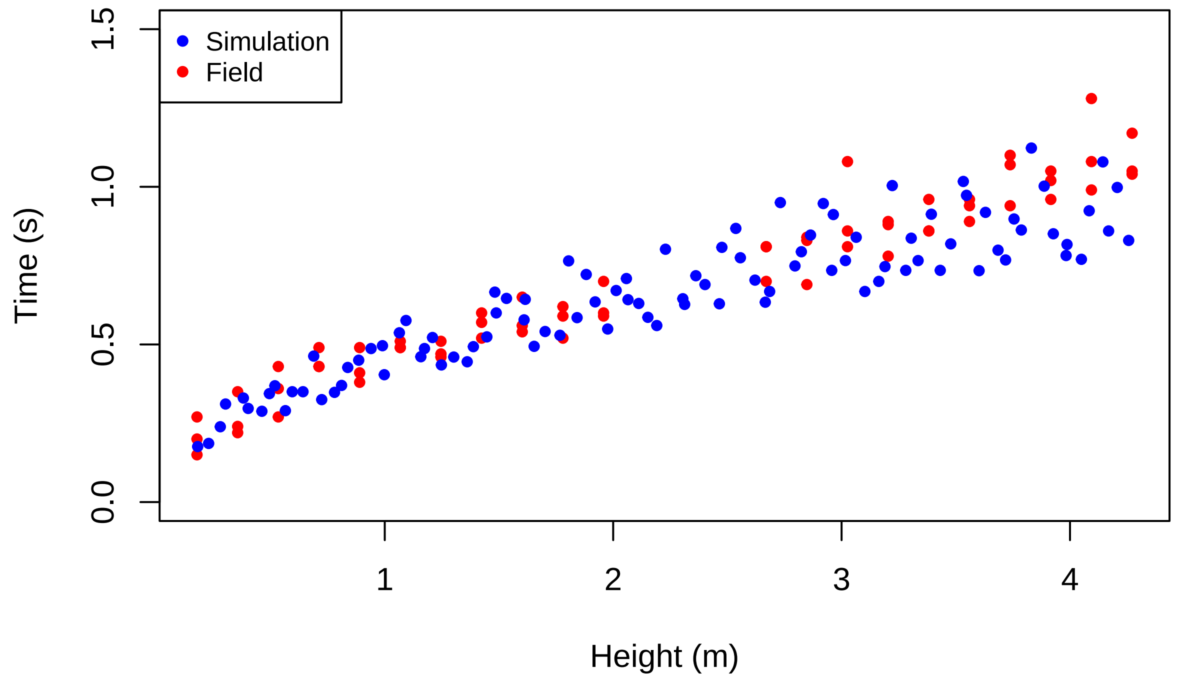


Figure 1: The time versus height plot for both field experiment and simulation.

The ball example is a simple pedagogical example as it has only one experimental input and one calibration input. Using this toy example, we demonstrate the functionality of the package without getting into the details of complex mathematical models in real-world problems. More complex and real-world examples are covered in the last section.

1.3 Building a Calibration Model

The `calibrate()` function takes three sets of arguments from user: data, MCMC, and prior specification arguments. Other than data arguments which must be supplied by user, all other arguments have reasonable default values to use with the ball example [6].

```
calMod <- calibrate(sim = ballSim, field = ballField,                  # Data
                    nMCMC = 11000, nBurn = 1000, thinning = 50,        # MCMC
                    kappaDist = "beta", kappaP1 = 1.1, kappaP2 = 1.1, # Priors
                    hypers = set_hyperPriors(),
                    showProgress = FALSE)
```

The first and second data arguments are `sim` and `field` that represent simulation and field data respectively. Both datasets are provided by users and must be in matrix format with the column structure specified in `ballSim` and `ballField`: in both matrices, the first column represent the response **t** and the second column represent experimental input **h**. Additionally, `ballSim` has a third column that represents the calibration input **g**. Calibration inputs are implicit in field experiment and are absent in the data matrix `ballField`.

[6] *Running the `calibrate()` function with reasonable number of MCMC runs is too lengthy to run while knitting the current vignette. However, this command has been run beforehand and the result is saved in package data to be loaded and used in the vignette. This `fbc` object, which is called `calMod`, can be found here.*

$$\begin{array}{lcccc} & & \text{Response} & \text{Experimental} & \text{Calibration} \\ \texttt{ballSim} & = & [\ \mathbf{t} & \mathbf{h} & \mathbf{g}\ ] \\ & & & & \\ \texttt{ballField} & = & [\ \mathbf{t} & \mathbf{h}\ ] & & \end{array}$$

The second set of arguments consists of MCMC parameters: `nMCMC` represents the number of total MCMC iterations , `nBurn` represent the number of burn-in iterations to be removed from the beginning of the chain, and `thinning` indicate the sampling rate to remove the autocorrelation from the sampled draws. For example, when `thinning = 50`, for every 50 draws from the result only one will be kept in order to remove the autocorrelation between draws.

The third set of arguments consists of prior specification for each parameter of the model. The main goal of calibration is to estimate the calibration parameters (g in ball example). However, employing KOH calibration model introduces eight additional classes of hyperparameters to the model. The parameters in all eight classes are not known in advance and must be estimated. As `FBC` employs a full Bayesian framework, the priors for all model parameters must be specified in advance. Throughout the package implementation and current guide a consistent notation is used to denote these parameters: κ, $\theta_\mathbf{s}$, $\alpha_\mathbf{s}$, $\theta_\mathbf{b}$, $\alpha_\mathbf{b}$, σ_s^2, σ_b^2, σ_e^2, and μ_b denote calibration parameters, correlation scale parameters of simulator GP, correlation smoothness parameters of simulator GP, correlation scale parameters of bias-correction GP, correlation smoothness parameters of bias-correction GP, marginal variance of simulator GP, marginal variance of bias-correction GP, measurement error variance, and mean of bias-correction GP respectively.

For each class of parameters, there are four associated arguments. 1) distribution type, which is a character string determining the prior distribution family (suffixed with "dist" in argument names). Currently, `FBC` supports almost all common distributions and can be chosen from "uniform", "normal", "normalTr", "log-normal", "gamma", "inversegamma", "beta", "betashift", "logbeta", "logistic","exponential", "fixed" [7]. 2) First distribution parameter (suffixed with "p1" in argument names) and 3) second distribution parameter (suffixed with "p2" in argument names) are doubles representing the two parameters of the chosen distribution [8], [9]. Table 1 summarizes all classes of parameters along with their corresponding argument names for prior specifications.

Table 1: *Argument names to specify priors for each parameter class.*

Parameter Class	Distribution	First Parameter	Second Parameters
True field calibration inputs (κ)	`kappaDist`	`kappaP1`	`kappaP2`
Simulation GP scale (θ_s)	`thetaSDist`	`thetaSP1`	`thetaSP2`
Simulation GP smoothness (α_s)	`alphaSDist`	`alphaSP1`	`alphaSP2`
Bias-correction GP scale (θ_b)	`thetaBDist`	`thetaBP1`	`thetaBP2`
Bias-correction GP smoothness (α_b)	`alphaBDist`	`alphaBP1`	`alphaBP2`
Simulation marginal variance (σ_s^2)	`sigma2SDist`	`sigma2SP1`	`sigma2SP2`
Bias-correction marginal variance (σ_b^2)	`sigma2BDist`	`sigma2BP1`	`sigma2BP2`
Measurement error variance (σ_ϵ^2)	`sigma2EDist`	`sigma2EP1`	`sigma2EP2`
Bias-correction mean (μ_b)	`muBDist`	`muBP1`	`muBP2`

[7] *"betashift" refers to a beta distribution that is shifted one unit to right to cover [1 2] interval and it is used to specify the priors for correlation smoothness parameters which must be constrained to [1 2] interval. Moreover, choosing "fixed" as distribution will exclude that class of parameters from the MCMC sampling. In this case, the given initial value will be used as fixed parameter value and p1 and p2 arguments will be used.*

[8] *For example, if "normal" distribution is used, p1 and p2 represent mean and variance of the distribution and if "uniform" distribution is used, p1 and p2 represent lower and upper bound of the distribution.*

[9] *Not all distribution types require two arguments. In particular, "exponential" distribution only requires rate parameter (p1) and "jeffreys" requires none. In these cases the unused arguments are ignored.*

Parameter Class	Distribution	First Parameter	Second Parameters

From the classes of parameters mentioned, the prior for calibration parameters should be specified by user based on field knowledge as these parameters are problem-specific [10]. In contrast, all other parameters have reasonable default values based on KOH calibration model literature (Kennedy & O'Hagan, 2001; Higdon et al. 2004; Chen et al. 2017). For this reason and to avoid unwanted complexity, the priors for all other classes of parameters are specified with `hypers` argument and using a helper function `set_hyperPriors()`. In particular, the default value of `hypers` argument is `set_hyperPriors()` without any argument. This will ensure that all default values are used for prior specification of hyperparameters. Nevertheless, expert knowledge about one or more of these parameters can be supplied using arguments of the `set_hyperPriors()` function, which are defined in Table 1, to change the default prior specifications.

Note that some classes of parameters may contain more than one parameters. In this case, the argument values can be vector instead of default scaler. In this case all four fields of that parameter class must be also in vector format with same length as number of parameters in the class. For example, if there are five calibration parameters, `kappaDist` can either be a scaler, in which case the distribution for all calibration parameters will be set to that scaler value, and `kappaP1` and `kappaP2` must be also scaler, or it can be a vector of length five that supplies the distribution types for all calibration parameters, and `kappaP1` and `kappaP2` must also be vector of length 5.

Moreover, there is a logical argument `showProgress` that indicate whether function must show the progress in calibration on console. Setting this argument to `TRUE`, will show the percentage of the MCMC draws along with sample draws [11].

The output of the `calibrate()` function is an object of class `fbc` that contains the samples from posterior joint distribution of parameters, along with other model information. Below, the components of the a `fbc` object is displayed.

```
names(calMod)
> [1] "Phi"        "estimates"  "logPost"    "priors"     "acceptance" "vars"       "data"
> [8] "scale"      "indices"
```

The first and main component of the `calMod` is matrix `Phi` whose columns represent the sample of posterior densities for each unknown parameter of the model in the same order as parameter classes in table 1. Since κ, θ_s, α_s, θ_b, α_b parameter classes may contain more than one parameter, they are suffixed by a number that represents the index of the parameter in the class. For example, if there are 3 calibration inputs, the first 3 columns of the matrix `Phi` represent posterior density of calibration parameters and the column headers will be `kappa1`, `kappa2`, and `kappa3`. In the ball example, there is only one calibration parameter g, which is denoted by κ_1 and represented by `kappa1` in matrix `Phi`. Each row of the matrix `Phi` represents a MCMC draw.

```
head(calMod$Phi, 3)
>    kappa1 thetaS1 thetaS2 alphaS1 alphaS2 thetaB1 alphaB1 sigma2S sigma2B sigma2E     muB
> 1 0.5783  0.6744  0.1233  1.9794  1.9607  0.6394  1.6308  3.2139  0.1689  0.0985 -0.2902
> 2 0.1822  0.6744  0.1160  1.9826  1.9743  0.6493  1.5982  4.1193  0.1662  0.0984  0.9832
> 3 0.3921  1.0425  0.1449  1.9851  1.9850  0.9704  1.5978  2.8183  0.1150  0.0984  0.2554
```

Table 2 provides an overview of the model parameters and the notation to represent them in ball example. Later sections will explain in detail why are these hyperparameters introduced, what do they represent, and how they are estimated.

[10] *Although a vague prior is specified as default for calibration parameters values, the user is encouraged to specify the prior based on the field knowledge. The default vague prior is specified using a uniform distribution with lower bound of 0 and upper bound of 1, which characterize the lower and upper bound of parameter domain after standardization ($U(0,1)$).*

[11] *Running `calibrate()` with high number of parameters or MCMC runs will be a lengthy process. This argument shows the progress of algorithm on percentage basis, along with sample of results, which can be useful for debugging and monitoring purposes.*

Table 2: Notation used in matrix `Phi` to represent parameters in ball example.

Column	Notation	Description
`kappa1`	κ_1	Unknown value of true calibration input g
`thetaS1`	θ_{s1}	Scale parameter of h input for simulator correlation
`thetaS2`	θ_{s2}	Scale parameter of g input for simulator correlation
`alphaS1`	α_{s1}	Smoothness parameter of h input for simulator correlation
`alphaS2`	α_{s2}	Smoothness parameter of g input for simulator correlation
`thetaB1`	θ_{b1}	Scale parameter of h input for bias-correction correlation
`alphaB1`	α_{b1}	Smoothness parameter of h input for bias-correction correlation
`sigma2S`	σ_s^2	Marginal variance of simulator covariance
`sigma2B`	σ_b^2	Marginal variance of bias-correction covariance
`sigma2E`	σ_ϵ^2	Variance of random measurement error in field
`muB`	μ_b	bias-correction mean

Apart from the matrix `Phi`, there are eight more elements in `calMod` object: The data frame `estimates`, which stores a summary table of all model parameters, namely their mean, mode, median, standard deviation, and 50% and 80% upper and lower quantiles of the marginal distribution of all parameters. The vector `logPost` contains the posterior log likelihood given a parameter draw from `Phi` matrix. The nested list `priors` contains prior specifications for all parameters. The vector `acceptance` represent the acceptance rate of each parameter after running the MCMC algorithm with adaptive proposal. It is important to note that, the current implementation of MCMC algorithm employs Metropolis-Within-Gibbs variation, which is a one-dimensional proposal scheme and the optimal acceptance rate must be close to 0.44. The vector of strings `vars` contains the parameter notation used in code and as `Phi` column headers. The rest of the components in `Phi` are not of great importance for user, however, they are needed for calibrated prediction. The list of matrices and vectors `data` includes the training data in the forms that match KOH model components. The numeric vector `scale` contains scaling factors that are used to scale the training data during calibration. The named list `indices` contain the indices of parameters in each row of `Phi` matrix.

1.4 Calibrated Prediction

Similar to any other predict function, `predict()` requires a model object argument called `object`, which in `FBC` package must be a `fbc` object, along with an argument representing a new input configuration called `newdata`. Moreover, current implementation of the `predict()`, support two different methods of prediction: Maximum A Posteriori ("MAP") and MCMC-based fully Bayesian ("Bayesian") methods. The method can be selected using `method` argument that can take a character string value of either "MAP" or "Bayesian".

```r
predsMAP   <- predict(object = calMod, newdata = matrix(c(2.2, 2.4), ncol = 1), method = "MAP")
predsBayes <- predict(object = calMod, newdata = matrix(c(2.2, 2.4), ncol = 1), method = "Bayesian")
```

The return value of `predict()` is a list consisting of two fields: `pred`, which is a vector of the predicted response for every new input configuration (rows of `newdata`), and `se`, which is a vector of the predicted response's standard errors.

```
predsMAP
> $pred
> [1] 0.6968523 0.7319061
>
> $se
> [1] 0.0801 0.0801
predsBayes
```

```
> $pred
> [1] 0.697 0.732
>
> $se
> [1] 0.0746 0.0746
```

In "Bayesian" method, which is the default value of `method` argument, MCMC draws of calibration model parameters are used to form a distribution of each predicted value. In particular, each rows of matrix `Phi` from the output of the calibration model, is used to predict the response for every row of `newdata`. Therefore, a distribution of response is created for each new input configuration. Then, the predictive mean and variance of the resulting distribution are used to compute the point predictions as well as standard errors for predictions.

In "MAP" method, the row of `Phi` matrix that results in maximum log posterior, is extracted and taken as model parameters to compute both point estimates and standard errors. This method is much faster than the "Bayesian" method as it only computes the prediction for each row of observation once.

1.5 Specifying Parameter Priors

As mentioned hyperparameters of the calibration model can be set using `set_hyperPriors()` function. All arguments of this function, which collectively specify the priors for all hyperparameters, have reasonable default values. Therefore `set_hyperPriors()` can be used without arguments to set the hyperparameters. In fact, the default value of the `hyper` argument in `calibrate()` is the function `set_hyperPriors()` without any argument. Nevertheless, when there is prior belief about structure of the correlation (either simulator GP or bias-correction GP), these beliefs can be applied to the model in the form of prior specification using `set_hyperPriors()`. For example in the following snippet, only correlation scale parameters of the simulation GP are set to "beta" distributions and the second parameter of beta distribution is set to 6. Therefore, all other parameter specification remain unchanged.

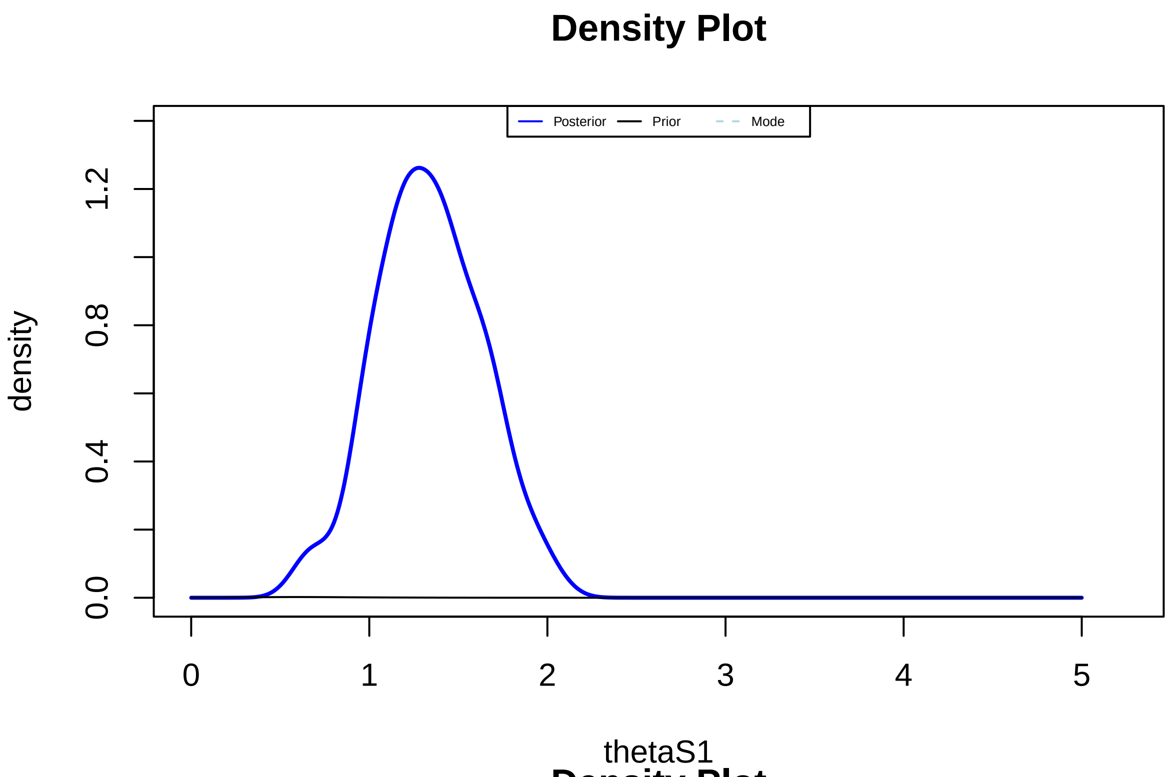
```
priors <- set_hyperPriors(thetaSDist = "beta", thetaBP2 = 6)
```

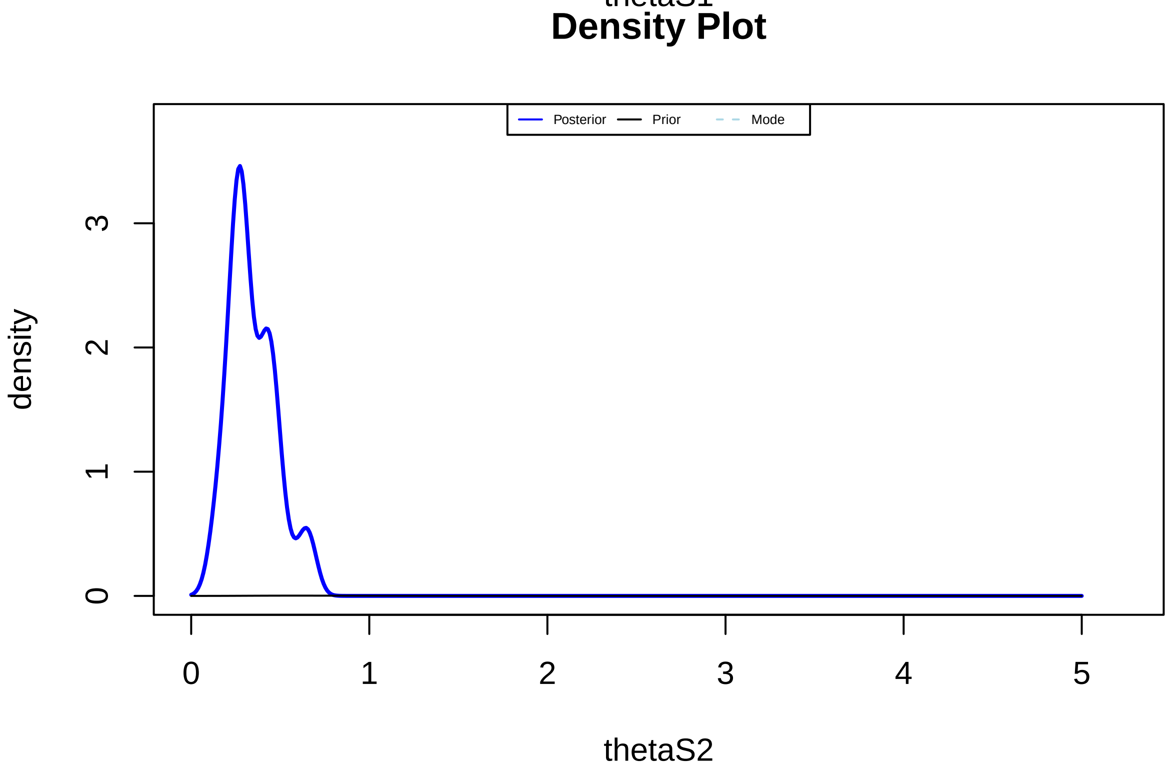
1.6 Visualization of Calibration Model

The implementation of this generic function `plot()` in `FBC` package enables visualization of a calibration model results. Given a calibration model in the form of a `fbc` object using argument `x`, `plot()` can visualize the model in three different mode, which can be chosen by supplying the `type` argument.

In particular, `type` must be a character string form "density", "trace", and "fits". Using "density", which is the default value of the `type` argument, `plot()` will plot the marginal posterior density distribution for the given parameter using `parameter` argument. It does so by estimating a density function given the MCMC-based posterior draws of the parameter. It also plots the prior distribution of the given parameter in the same plot for ease of comparison and visualizes the empirical mode of the posterior density. The `parameter` argument must be a character string consistent with the notation used throughout the package and current vignette , namely it must be chosen from "kappa", "thetaS", "alphaS", "thetaB", "alphaB", "sigma2S", "sigma2B","sigma2E", or "muB". The default value for `parameter` argument is "kappa" or calibration parameters, which usually are the parameters of the interest. Note that `parameter` argument characterizes the class of parameters and when there is more than one parameter in that class,`plot()` will plot the density distribution for all parameters in that class in separate plots.

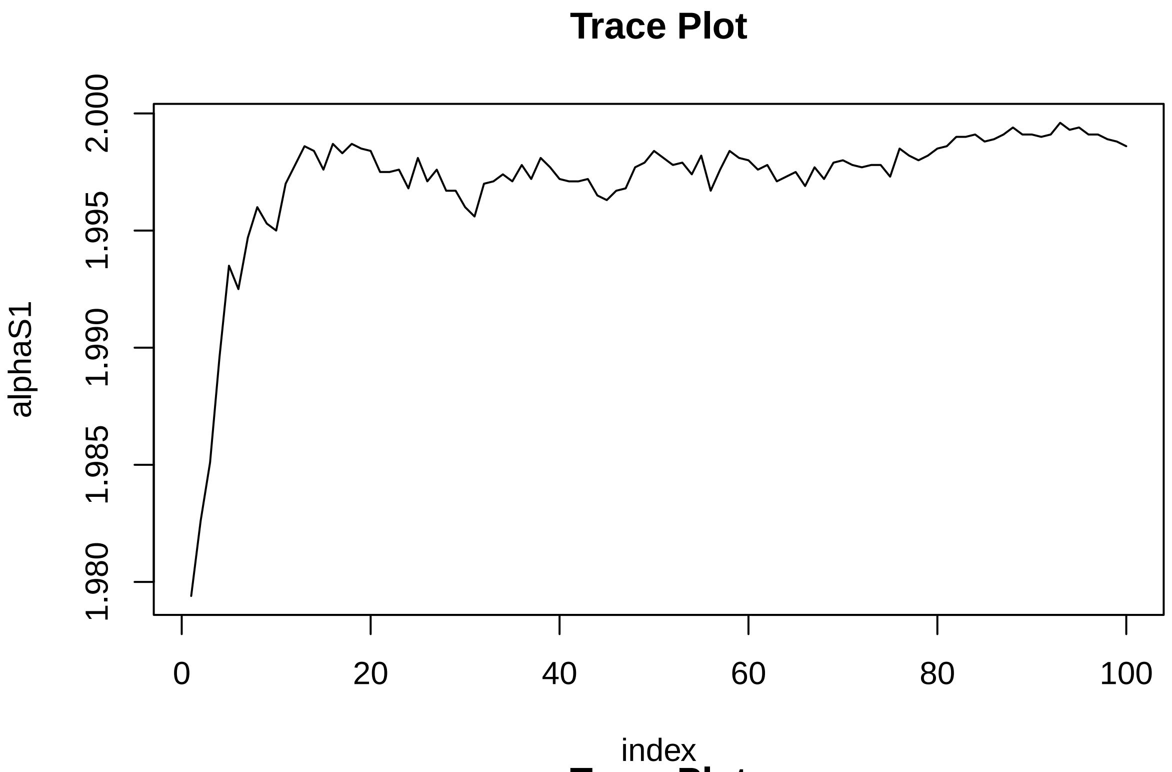
```
# Note that there are two correlation scale parameters in simulator GP and there will be two plots
plot(calMod, parameter = "thetaS")
```

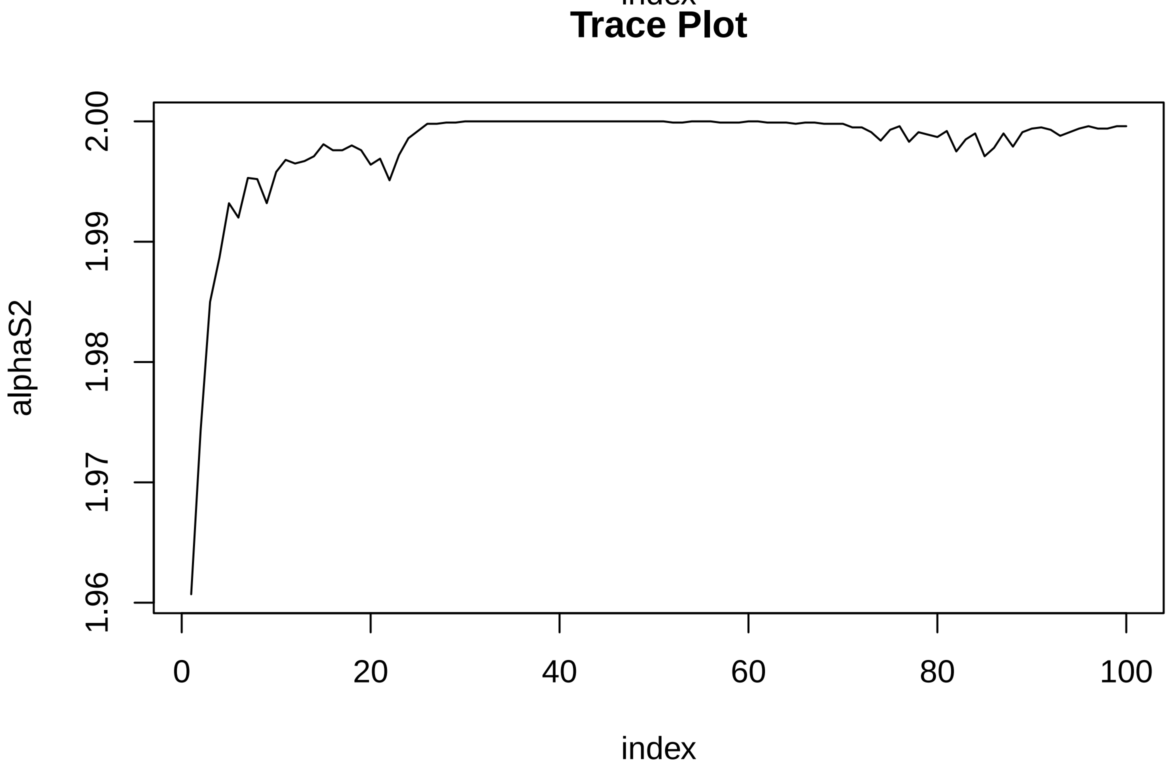




Using "trace" as `type` argument, `plot()` will plot the progression of the given parameter as MCMC draws are taken. It is similar to the time series of the parameter but indexed with number of iteration in MCMC rather than time. The trace plot can be used to determine whether there is good mixing in MCMC draws. Using "trace" as `type` argument also requires supplying the `parameter` from aforementioned list of possible parameter classes. And similar to density plots, trace plots will be plotted for all of the parameters in the given class in separate plots.

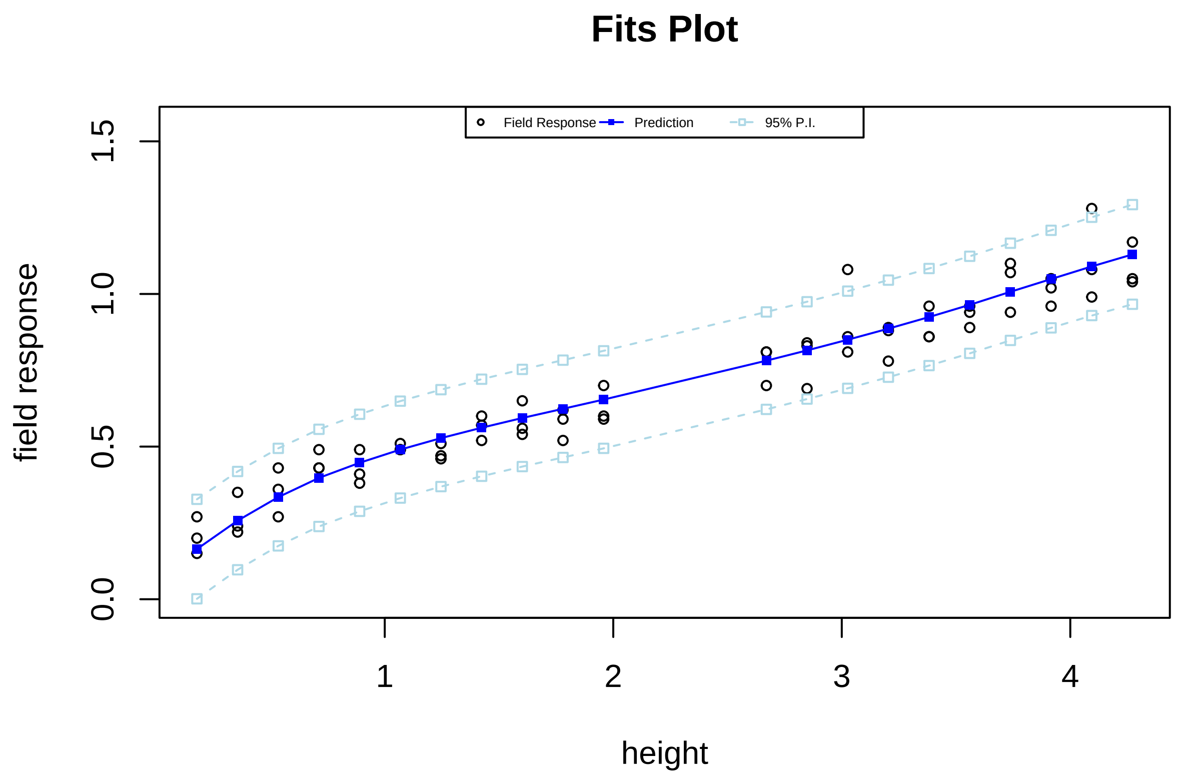
```r
# Note that there are two correlation smoothness parameters in simulator GP and there will be
# two plots
plot(calMod, parameter = "alphaS", type = "trace")
```





And finally using "fits" as `type` argument, `plot()` will plot the fitted values of the response versus all experimental variables in separate plots. The `parameter` argument is not required for this type and will be ignored. Fits plots can be used to visually determine the goodness of fits versus actual response during interpolations. Internally, `plot()` will use `predict()` function (using "MAP" method) to compute the fitted values for the training input configurations and will plot them in along with actual values. Furthermore, `xlab` argument can be supplied with a character string to characterize the experimental variables' names. If not supplied, the x-axis will labelled by "x1", "x2", until last experimental variable.

```r
# Plots the fitted values versus all experimental inputs along with actual values in separate plots
plot(calMod, type = "fits", xlab = "height")
```



1.7 Summarizing the Calibration Model

Both `summary()` and `print()` generic functions are implemented to work with the output of `calibrate()` function. In particular, given a `fbc` object, `summary()` returns the `estimate` component of `calibrate()` output, which is a data frame containing statistical summary of calibration parameters. Similarly, `print()` will display the same summary data frame in the console.

```r
calModSum <- summary(calMod)
print(calMod)
>       mean  median    mode   lwr50  upr50   lwr80   upr80     sd
> 1   0.3154  0.2803  0.2885  0.1710 0.4453  0.0930  0.6058 0.1990
> 2   1.3368  1.3245  1.1670  1.1384 1.5366  0.9934  1.7134 0.2925
> 3   0.3443  0.3019  0.2735  0.2560 0.4280  0.1930  0.4892 0.1304
> 4   1.9972  1.9978  1.9980  1.9971 1.9985  1.9959  1.9991 0.0030
> 5   1.9980  1.9995  1.9996  1.9980 2.0000  1.9957  2.0000 0.0051
> 6   0.8740  0.8477  0.8390  0.6629 1.0787  0.5160  1.2629 0.2801
> 7   1.8164  1.8625  1.9446  1.7202 1.9419  1.5982  1.9624 0.1464
> 8   7.2093  6.8783  4.8156  4.8918 9.1035  3.9589 11.8601 2.9346
> 9   0.1727  0.1704  0.1796  0.1262 0.2010  0.1048  0.2448 0.0540
> 10  0.1130  0.1127  0.1239  0.0997 0.1246  0.0965  0.1302 0.0133
> 11 -0.1727 -0.3197 -0.5427 -0.6788 0.3548 -0.8869  0.7469 0.5987
```

2. Calibration Model

Calibration is a statistical method that aims to tune in calibration parameters so that simulator model mimics field process more closely or at least account for the systemic bias of the simulator. Calibration model is a statistical object that characterizes the posterior distribution of calibration parameters given training data. In this section, the theory behind building a calibration model is explained along with the notation to represent the components of the calibration model [12].

2.1 Data

In general, a field experiment has n observations, each of which require p experimental inputs. On the other hand, a simulation also has m code runs, each of which require $p+q$ experimental and calibration inputs. The calibration inputs, which represent either tuning parameters or physical properties that are not controllable by field experimenter, are implicit in the field experiment and their values are unknown but assumed to be fixed throughout observations. To represent both experiments in a unified structure, the unknown calibration inputs of field experiment, which are represented by κ [13], must be augmented to experimental inputs, so that both physical and computer experiments have $(p + q)$ inputs and a univariate response. Stacking the input vectors, we can represent both field experiment and simulation input data in matrix notation. Similarly the response in both field experiment and simulation can be represented in vector notation. Subscripts f denotes field data, subscript s denotes simulation data, and subscript κ denotes augmented field data. Table 3 describes the response and inputs of both field experiment and computer simulation both using both vector and matrix notation.

Table 3: Notation used to represent field and simulation data component of calibration model along with their corresponding values and identifiers for the ball example.

Notation	Description	Ball Example
n	Number of field observations	63
m	Number of simulation runs	100
p	Number of experimental inputs	1
q	Number of calibration inputs	1
$\mathbf{x_f}$	Field input vector containing p experimental inputs	h [14]
$\mathbf{X_f}$	Field input matrix $((n \times p))$	$\mathbf{h}$ [15]
y_f	Univariate field response	t (scaler)
$\mathbf{y_f}$	Vector of n univariate field response	$\mathbf{t}$ (vector of length 63)
κ	Vector of true calibration inputs in field experiment	True value of gravity κ_1[16]
$\mathbf{x}_\kappa$	Augmented field input vector containing $(p+q)$ inputs	(h, κ_1) (vector of length 2)
$\mathbf{X}_\kappa$	Augmented field input matrix $((n \times (p+q)))$[17]	$[\mathbf{h} \quad \boldsymbol{\kappa_1}]$
$\mathbf{x_s}$	Simulation input vector containing $(p+q)$ inputs	(h, g) (vector of length 2)

[12] *Note that the notation to use the calibration model is already described in section 1. In this section the notation used to represent the data and calibration model components are explained, which are used internally in the package.*

[13] *Elements of vector κ are parameters of the calibration model and will be estimated by `calibrate()`.*

[14] *In ball example there is only one experimental input and therefore $\mathbf{x_f}$ is a vector of length one or scaler.*

[15] *In matrix notation of the field data, $\mathbf{X_f}$ is a (63×1) matrix or a vector of length 63.*

[16] *In the ball example, there is only one calibration parameter (gravity), which is denoted by κ_1 and represented by `kappa1` in `Phis` column headers.*

[17] *Note that calibration parameters κ are often assumed to be unchanged throughout field experiment. Therefore, same (κ) vector is augmented to all of the field input configurations.*

Notation	Description	Ball Example
$\mathbf{X_s}$	Simulation input matrix $((m \times (p+q)))$	$[\mathbf{h}\ \mathbf{g}]$ $((100 \times 2)$
y_s	Univariate simulation response	t (scaler)
$\mathbf{y_s}$	Vector of m univariate simulation response	$\mathbf{t}$ (vector of length 100)

2.2 KOH Model

KOH models the functional relationship between simulation input and output as a realization of a random function $\eta(\mathbf{x_s})$. Similarly, KOH models the functional relationship between field input and output as a realization of random function $\eta(\mathbf{x}_\kappa)$ but acknowledges a systemic model discrepancy and measurement errors. Furthermore, the discrepancy term is modelled using another random function $\delta_\kappa(\mathbf{x_f})$ [18] and the error term ϵ is considered to be an independent draw from a normal distribution with zero mean and unknown variance σ_ϵ^2. This formulation, emphasizes that model has systemic bias, even under best calibration parameters. However, KOH postulates that reasonable correction can be made through $\delta_\kappa(.)$

$$y_f = \eta(\mathbf{x}_\kappa) + \delta_\kappa(\mathbf{x_f}) + \epsilon$$

$$y_s = \eta(\mathbf{x_s})$$

$$\text{where} \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

Therefore other than κ and σ_ϵ^2 parameters, the random functions $\eta(.)$ and $\delta_\kappa(.)$ are also unknown and must be specified. KOH models $\eta(.)$ and $\delta_\kappa(.)$ by two independent GPs.

$$\begin{aligned} \eta(.) &\sim GP\ (0,\ \sigma_s^2.R_s(.,.)) \\ \delta_\kappa(.) &\sim GP\ (\mu_b,\ \sigma_b^2.R_b(.,.)) \end{aligned}$$

Where μ_b represent the mean of the bias-correction GP and considered to be constant and the mean of simulator GP considered to be zero [19] . Moreover, σ_s^2 and σ_b^2 denote marginal variance of simulator and bias-correction GPs, and $R_s(.,.)$ [20] and $R_b(.,.)$ [21] are correlation matrix of simulator and bias-correction GP.

2.3 Correlation Structure

There are many choices to characterize the correlation structure including Gaussian correlation family, Matern, and power exponential family. `FBC` employs a power exponential correlation family to represent the correlation structure of both GPs as it provides flexibility and interpretability. Assuming $\mathbf{x}$ and $\mathbf{x}'$ are

[18] *Note that although $\delta_\kappa(.)$ is considered to be function of $\mathbf{x_f}$, it is still dependent on κ parameters. This is emphasized by using the κ subscript in $\delta_\kappa(.)$.*

[19] *Note that the mean of the simulator GP is considered to be zero because `calibrate()` function first standardizes simulation response $\mathbf{y_s}$ to have the mean of zero and standard deviation of one. On the other hand, the mean of the bias-correction GP is dependent on the extent of simulator model inadequacy and will vary from one model to another. However, some studies indicate that fitting a linear regression model does not necessarily improve the performance as the GP models are already flexible (Chen et al., 2017). Therefore a constant mean is assumed for the mean of bias-correction GP as a calibration model parameter that may or may not be zero.*

[20] *The input of $R_s(.,.)$ can be either rows of $\mathbf{X}_\kappa$ or $\mathbf{X_s}$.*

[21] *The input of $R_b(.,.)$ must be the rows of $\mathbf{X_f}$.*

two rows of full input matrix [22] and $\mathbf{x_f}$ and $\mathbf{x'_f}$ are two rows of field experimental input matrix $\mathbf{X_f}$, the correlation matrices $R_s(\mathbf{x}, \mathbf{x'})$ and $R_b(\mathbf{x_f}, \mathbf{x'_f})$ are defined as following:

$$R_s(\mathbf{x},\ \mathbf{x'})\ = \prod_{i=1}^{p+q} e^{-\theta_i |x_i - x'_i|^{\alpha_i}}$$

$$R_b(\mathbf{x_f},\ \mathbf{x_f}') = \prod_{j=1}^{p} e^{-\theta_j |x_j - x'_j|^{\alpha_j}}$$

Where x_i, x'_i, x_j, and x'_j denote the input elements of vectors $\mathbf{x}$, $\mathbf{x'}$, $\mathbf{x_f}$, and $\mathbf{x'_f}$ respectively. Using separable power exponential correlation family introduces additional four classes of new hyperparameters to calibration model: vector $\theta_\mathbf{s}$, which denotes the scale parameters of $R_s(.,.)$ (θ_is for $i \in (1, ..., p+q)$), vector $\alpha_\mathbf{s}$, which denotes the smoothness parameters of $R_s(.,.)$ (α_is for $i \in (1, ..., p+q)$), vector $\theta_\mathbf{b}$, which denotes the scale parameters of $R_b(.,.)$ (θ_js for $j \in (1, ..., p)$), and vector $\alpha_\mathbf{b}$, which denotes the smoothness parameters of $R_b(.,.)$ (α_js for $j \in (1, ..., p)$). Together, they flexibly determine the shape of correlation structure. To recap, in addition to κ, and σ_ϵ^2, seven classes of parameters are introduced to the calibration model by using two independent GPs to characterize the unknown random functions $\eta(.)$ and $\delta_\kappa(.)$: $\theta_\mathbf{s}$, $\alpha_\mathbf{s}$, $\theta_\mathbf{b}$, $\alpha_\mathbf{b}$, σ_s^2, σ_b^2, σ_ϵ^2, and μ_b. The joint vector of all parameters in the final calibration model is denoted by vector ϕ [23].

$$\phi = (\kappa_1,\ ...,\ \kappa_q,\ \theta_{s1},\ ...,\ \theta_{s(p+q)},\ \alpha_{s1},\ ...,\ \alpha_{s(p+q)},\ \theta_{b1},\ ...,\ \theta_{bp},\ \alpha_{b1},\ ...,\ \alpha_{bp},\ \sigma_s^2,\ \sigma_b^2,\ \sigma_\epsilon^2,\ \mu_b)$$

Table 1 in section 1 summarizes all calibration model parameters along with the notation used in current vignette and code. It also indicates the corresponding parameters in each class for the ball example.

2.4 Full Model

After augmentation of true calibration inputs (vector κ to field data, both simulation and field experiment have the same input structure. KOH stacks both datasets from field experiment and simulation to build a joint dataset.

$$\mathbf{y}\ = \begin{bmatrix}\mathbf{y_f}\\ \mathbf{y_s}\end{bmatrix} = (y_1,\ ...\ ,\ y_{n+m})^T \qquad \text{(joint vector of responses)}$$

$$\mathbf{X} = \begin{bmatrix}\mathbf{X}_\kappa\\ \mathbf{X_s}\end{bmatrix} = \begin{bmatrix}\mathbf{x_1}\ \mathbf{x_2}\ ...\ \mathbf{x_{p+q}}\end{bmatrix} \qquad \text{(joint input matrix)}$$

$$\mathbf{x_i} = (x_1, x_2, ..., x_{n+m}) \qquad \forall i \in \{1, 2, ..., p+q\}$$

And the full model that represents the functional relationship between joint input and response is realization of a random function, denoted by $\zeta(.)$. Where y is the response in full model (an element of vector $\mathbf{y}$) and $\mathbf{x}$ is an input configuration in full model (a row in matrix $\mathbf{X}$). Using matrix notation, the full model can be represented by following equation, which forms the basis for estimating the calibration model parameters.

$$y\ = \zeta(\mathbf{x})$$
$$\mathrm{y}\ = \zeta(X)$$

[22] *The full input matrix is considered to be either $\mathbf{X_s}$ or $\mathbf{X}_\kappa$, which is the augmented field input matrix.*

[23] *Note that in the ball example, the column headers of matrix `Phi` exactly match to model parameters. In fact, each rows of matrix `Phi` is a vector ϕ that represents a draw from joint parameter space.*

Because both $\eta(.)$ and $\delta_\kappa(.)$ are modelled by two GPs and the error term is Gaussian, $\zeta(.)$ can also be modelled by a GP (Santner et al., 2018) and its components can be computed using the calibration model components. Moreover, given the data component, computing the mean function $\mu(.)$ and covariance function $\Sigma(.,.)$ result in a mean vector μ and a covariance matrix Σ.

$$\zeta(.) \sim GP\ (\mu(.),\ \Sigma(.,.))$$

$$\text{where,} \quad \mu = \begin{bmatrix} \mu_b \\ 0 \end{bmatrix}$$

$$\Sigma = \sigma_s^2 . \begin{bmatrix} R_s(X_\kappa, X_\kappa) & R_s(X_\kappa, X_s) \\ R_s(X_s, X_\kappa) & R_s(X_s, X_s) \end{bmatrix} + \sigma_s^2 . \begin{bmatrix} R_b(X_f, X_f) & 0 \\ 0 & 0 \end{bmatrix} + \sigma_\epsilon^2 . \begin{bmatrix} I_n & 0 \\ 0 & 0 \end{bmatrix}$$

Note that in the following equation correlation functions are represented using matrix notation. For example, $R_s(X_s, X_s)$ represents the correlation matrix that is created using $R_s(.,.)$ between rows of X_s and itself.

3. Parameter Estimation

In its original formulation, KOH introduces a hierarchical semi-Bayesian framework with two sequential phases. In the first phase, model hyperparameters are estimated using Maximum Likelihood Estimation (MLE) method. In the second phase, Bayesian analysis is employed to derive the posterior distribution of calibration parameters while fixing the hyperparameters found in the first phase. This approach not only ignores the added uncertainty of the estimated hyperparameters, it also does not enable inference about hyperparameters. On the other hand, `FBC` runs a fully Bayesian model that includes both calibration parameters and model hyperparameters in its Bayesian framework. Therefore, to find the joint posterior density distribution of all parameters, priors for all parameters must be specified and the joint likelihood must be estimated from full data.

3.1 Joint Prior Density

There are nine classes of parameters in the calibration model, from which at least calibration parameters (vector κ), measurement error variance (scaler σ_ϵ^2), and bias-correction GP mean (scaler μ_b) are application-dependent. It is recommended for user to specify the prior arguments for these parameters based on prior knowledge or consensus [24]. Appendix A.1 describes the full prior specification in `FBC` that is based on literature review on computer experiments and calibration models. Collectively, the priors for all model parameters ϕ is denoted by $\mathcal{P}[\phi]$, which is the product of all parameter priors [25].

[24] *Even for these three classes of parameters, the priors has been specified in* `FBC` *using default values, but the user is encouraged to specify them based on their application.*

[25] *The parameters in the calibration model are considered to be independent from one another. Therefore, their joint prior is simply the product of all parameter priors.*

$$
\begin{aligned}
\mathcal{P}[\phi] \;\propto\; & \prod_{i=1}^{q} \mathcal{P}[\kappa_i] && \text{(priors for calibration parameters)} \\
\times\; & \prod_{i=1}^{p+q} \mathcal{P}[\theta_{si}] \;\times\; \prod_{i=1}^{p+q} \mathcal{P}[\alpha_{si}] \;\times\; \mathcal{P}[\sigma_s^2] && \text{(priors for } \eta(.) \text{ GP)} \\
\times\; & \prod_{i=1}^{p} \mathcal{P}[\theta_{bi}] \;\times\; \prod_{i=1}^{p} \mathcal{P}[\alpha_{bi}] \;\times\; \mathcal{P}[\sigma_b^2] \;\times\; \mathcal{P}[\mu_b] && \text{(priors for } \delta_\kappa(.) \text{ GP)} \\
\times\; & \mathcal{P}[\sigma_\epsilon^2] && \text{(prior for measurement error variance)}
\end{aligned}
$$

3.2 Conditional Likelihood

Since the full $\mathbf{y}$ is modelled as a GP, by definition, the conditional likelihood of the full response given a parameter vector ϕ is characterized by following equation.

$$\mathcal{L}(\mathbf{y} \mid \phi) = |\mathbf{\Sigma}|^{-\frac{1}{2}} \;.\; \mathbf{e}^{-\frac{1}{2}\;(\mathbf{y}-\mu).\;\mathbf{\Sigma^{-1}}.\;(\mathbf{y}-\mu)^{\mathbf{T}}}$$

Where $|\Sigma|$ and Σ^{-1} are the determinant and inverse of the full covariance matrix.

3.3 Conditional Posterior Distribution of Parameters

Given the joint prior distribution of the parameters and conditional likelihood of the response, the posterior distribution of the parameters given the full response can be characterized using the following relationship.

$$\mathcal{P}[\phi \mid \mathbf{y}] \;\propto\; \mathcal{L}(\mathbf{y}|\phi) \;.\; \mathcal{P}[\phi]$$

Taking the log from both sides will decrease computational load and increase speed.

$$
\begin{aligned}
\log(\mathcal{P}[\phi \mid \mathbf{y}]) \;\propto\; & \log(\mathcal{L}(\mathbf{y} \mid \phi)) \;+\; \log(\mathcal{P}[\phi]) \\
= & -\frac{1}{2}\log(|\Sigma|) - \frac{1}{2}(\mathbf{y}-\mu).\mathbf{\Sigma^{-1}}.(\mathbf{y}-\mu)^{\mathbf{T}} && \text{(log liklihood given full response)} \\
+\; & \sum_{i=1}^{q} \mathcal{P}[\kappa_i] && \text{(priors for calibration parameters)} \\
+\; & \sum_{i=1}^{p+q} \mathcal{P}[\theta_{si}] + \sum_{i=1}^{p+q} \mathcal{P}[\alpha_{si}] \;+\; \mathcal{P}[\sigma_s^2] && \text{(priors for } \eta(.) \text{ GP)} \\
+\; & \sum_{i=1}^{p} \mathcal{P}[\theta_{bi}] + \sum_{i=1}^{p} \mathcal{P}[\alpha_{bi}] \;+\; \mathcal{P}[\sigma_b^2] + \mathcal{P}[\mu_b] && \text{(priors for } \delta_\kappa(.) \text{ GP)} \\
+\; & \mathcal{P}[\sigma_\epsilon^2] && \text{(prior for measurement error variance)}
\end{aligned}
$$

In order to find the unconditional posterior distribution of the parameters, above equation must be integrated. However, this equation is intractable and thus a numerical method must be employed to sample from the posterior distribution.

3.4 MCMC Simulation

A common numerical method to draw samples from the joint posterior distribution of the parameter space is Markov Chain Monte Carlo (MCMC) algorithm . To select the suitable variation of MCMC algorithm, it is worth noting that building a calibration model can often introduce a large number of parameters [26], which can in turn make updating the whole parameter vector at once harder. To circumvent this issue, a Metropolis-Within-Gibbs MCMC algorithm is used with adaptive proposal to build a sample of joint distribution of parameters. In every run, a vector ϕ is generated by algorithm that together build matrix Φ row by row [^27]. The first row of Phi is assumed to the mean of the given prior distribution. Then, the algorithm creates a Markov chain by updating parameters in each iteration according to a proposal scheme. In Metropolis-Within-Gibbs algorithm, parameters are updated one by one using a one-dimensional proposal function. Updating all parameters once constitutes an iteration of MCMC and adds one row to Φ.

[^27] *In the current implementation of `FBC` package, matrix Φ, which is the output of MCMC algorithm, is denoted by `Phi`*

Once a new value for the parameter is proposed, the conditional posterior log likelihood can be computed according to the equation in section 3.3, given the training data and the most recent updates of parameters. If joint posterior density is larger than the density of a random draw from standard uniform distribution, the algorithm accepts the proposed parameter by adding the parameter value to the current row in matrix Φ, otherwise the proposed value is rejected and parameter value is updated using the last update [27]. After updating the parameters one by one, either accepting or rejecting the new proposals, an iteration of MCMC is complete which corresponds to a row in matrix Φ.

To propose a parameter a random draw is taken from a Gaussian distribution with the mean of most recent update of that parameter and an adaptive standard deviation (SD). In adaptive Mtreopolis-Within-Gibbs, the SD of the proposal must be adaptively adjusted, so that the rate of proposal acceptance is approximately 0.44 (Bedard & Rosenthal, 2008). Often SD is considered to be in the form of e^{l_i}, where l_i is adaptively adjusted. One approach to adapt l_i is to break up the MCMC runs into batches. After each batch completes, the acceptance rate is computed in the last batch. If the rate is more than 0.44, l_i is increased by δ_i, otherwise it is decreased by δ_i. Furthermore, in order to guarantee convergence of the sample from posterior joint distribution to the actual distribution, δ_i must approach zero as number of iterations approaches to infinity [28] (Roberts & Rosenthal, 2007). The detailed MCMC algorithm is presented in the Appendix A.2.

In the end, each row of Φ represents a draw from joint distribution of parameters and each column represents a parameter in the model. The matrix represents joint samples of the parameters and any given column represents a sample from marginal posterior distribution of the corresponding parameter, which can be used further for inference [29] or calibrated predictions.

4. Calibrated Prediction

Estimating the joint distribution of calibration model parameters enables calibrated prediction. Matrix Φ can be used to to predict the physical response, simulator response, and bias for new input configurations. Here, only the calibrated prediction of physical response, which in practice is often the objective, is discussed using two different approaches that unlike MLE method provides a measure of uncertainty. In one approach

[26] *In the ball example that was introduced in chapter one, even though the simulation had only one experimental input and one calibration input, the calibration model had 11 parameters in total.*

[27] *Assuming iteration i in MCMC algorithm, the parameter value in the i-th row of Φ is updated.*

[28] *In `FBC` implementation of adaptive proposal, δ_i is $\frac{1}{\sqrt{i}}$*

[29] *Summary statistics of the sample of marginal distributions are provided by `FBC` to facilitate inference about a given parameter.*

and following a Bayesian framework, all joint parameter samples (all rows of matrix Phi) are used to form a sample distribution of predictions for a given input configuration. In another approach, which is significantly faster than first approach, the parameter vector in Φ that results in maximum posterior log likelihood is extracted from matrix Φ to form Maximum A Posteriori (MAP) prediction for a given input configuration.

4.1 Setup

To form a field response prediction, y_f^*, given a new field input configuration, $\mathbf{x_f^*}$, the data components must be standardized and augmented in the same way as training data. However, instead of updating the parameter vector, calibration parameters in rows of matrix Φ is used to augment the data and form prediction components. Furthermore, although calibrated prediction is performed for a given new input vector, a collection of new input configurations can be denoted by $\mathbf{X_f^*}$ that, when predicted one by one, leads to a vector of predicted field responses $\mathbf{y_f}^*$. The notation for new field input configuration and the parameters are described in Table 4.

Table 4: Notation used to represent new field and augmented field data.

Notation	Description
n^*	Number of new field input configurations
$\mathbf{x_f^*}$	New field input vector containing p experimental inputs
$\mathbf{X_f^*}$	New field input matrix $((n^* \times p))$
y_f^*	Predicted univariate field response
$\mathbf{y_f^*}$	Vector of n^* predicted field responses

To compute calibrated prediction, the notation used in Table 3 is also used to represent training data components.

4.2 Bayesian Approach

In Bayesian approach, the i-th row of Φ, denoted by $\phi^{(i)}$, is used to form the i-th prediction for $\mathbf{x_f^*}$. Specifically, $\kappa^{(i)}$, which is a subset of $\phi^{(i)}$, is augmented to $\mathbf{x_f^*}$ to form $\mathbf{x}_\kappa^{(\mathbf{i})}$. As a GP, the field response random function, $\zeta(.)$, can be characterized by identifying its mean and covariance functions. In the Bayesian method, each MCMC-based parameter vector will be used to compute predictive mean and variance values and together they provide a conditional sample of the distribution for any given input configuration. Then, numerical methods can be used to compute the unconditional mean and variance of the predictions using the distribution of predictive means and variances. The components to perform calibrated prediction along with their notation is introduced in Table 5.

Table 5: Components of calibrated prediction in Bayesian approach along with their notation.

Notation	Description
$\phi^{(i)}$	A vector of all model parameters (i-th row of Φ)
$\kappa^{(i)}$	A vector of calibration input configuration (subset of $\phi^{(i)}$)
$\sigma_s^{2\ (i)}$	marginal variance of simulator GP (subset of $\phi^{(i)}$)
$\sigma_b^{2\ (i)}$	marginal variance of bias-correction GP (subset of $\phi^{(i)}$)
$\sigma_\epsilon^{2\ (i)}$	measurement variance (subset of $\phi^{(i)}$)
$\mathbf{x}_\kappa{}^{*\ (\mathbf{i})}$	An augmented field input vector containing $(p+q)$ inputs

Notation	Description
$\mathbf{X}_{\kappa}{}^{*\ (\mathbf{i})}$	Augmented field input matrix, built using training data and $\kappa^{(i)}$
$\Sigma^{-1\ (i)}$	Covariance matrix of $\zeta(.)$ given training data and $\phi^{(i)}$

To compute the distribution of predictive means for $\zeta(\mathbf{x_f}^*)$, the following equation is used for $\phi^{(i)}$, where i represents the i-th row in Φ. In this equation, correlation functions are represented using vector/matrix notation. In particular, $R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X_s})$ represents the correlation vector that is computed using $R_s(.,.)$ between vector $x_{\kappa}^{*^{(i)}}$ and rows of X_s. Similarly, $R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X}_{\kappa}^{(\mathbf{i})})$ represents the correlation vector that is computed using $R_s(.,.)$ between vector $x_{\kappa}^{*^{(i)}}$ and rows of $\mathbf{X}_{\kappa}^{(\mathbf{i})}$, and $R_b(\mathbf{x_f}^*, \mathbf{X_f})$ represents the correlation vector that is computed using $R_b(.,.)$ between vector $\mathbf{x_f}^*$ and rows of $\mathbf{X_f}$. The inverse of covariance matrix of $\zeta(.)$ must also be computed for each $\phi^{(i)}$ using training data.

$$E(\zeta(\mathbf{x_f}^*)|\mathbf{y}, \phi^{(\mathbf{i})}) = \mu + \begin{bmatrix} R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X_s}) \\ R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X}_{\kappa}^{(\mathbf{i})}) + \mathbf{R_b}(\mathbf{x_f}^*, \mathbf{X_f}) \end{bmatrix}^{\mathbf{T}} .\mathbf{\Sigma^{-1}}^{(\mathbf{i})}.(\mathbf{y} - \mu)$$

Assuming M is the total number of Φ rows, the overall predictor can be computed using the conditional predictive means.

$$E(\zeta(\mathbf{x_f}^*)) = \frac{\mathbf{1}}{\mathbf{M}} \sum_{\mathbf{i=1}}^{\mathbf{M}} \mathbf{E}(\zeta(\mathbf{x_f}^*)|\mathbf{y}, \phi^{(\mathbf{i})})$$

To compute the distribution of predictive variance for $\zeta(.)$, the following equation is used. It contains similar components as predictive mean but it is not dependent on μ and $\mathbf{y}$. Additionally, the predictive variance will be dependent on marginal variances $\sigma_s^{2\ (i)}$ and $\sigma_b^{2\ (i)}$ as well as measurement variance $\sigma_\epsilon^{2\ (i)}$.

$$\begin{aligned}\text{Var}(\zeta(\mathbf{x_f}^*)|\mathbf{y}, \phi^{(\mathbf{i})}) &= \sigma_{\mathbf{s}}^{\mathbf{2}\ (\mathbf{i})} + \sigma_{\mathbf{b}}^{\mathbf{2}\ (\mathbf{i})} + \sigma_{\epsilon}^{\mathbf{2}\ (\mathbf{i})} \\ &- \begin{bmatrix} R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X_s}) \\ R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X}_{\kappa}^{(\mathbf{i})}) + \mathbf{R_b}(\mathbf{x_f}^*, \mathbf{X_f}) \end{bmatrix}^{T} .\Sigma^{-1^{(i)}}. \begin{bmatrix} R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X_s}) \\ R_s(\mathbf{x}_{\kappa}^{*^{(\mathbf{i})}}, \mathbf{X}_{\kappa}^{(\mathbf{i})}) + \mathbf{R_b}(\mathbf{x_f}^*, \mathbf{X_f}) \end{bmatrix}\end{aligned}$$

The overall unconditional variance can be computed using the law of total variance.

$$\text{Var}(\zeta(\mathbf{x_f}^*)) = \frac{\mathbf{1}}{\mathbf{M}} \sum_{\mathbf{i=1}}^{\mathbf{M}} \text{Var}(\zeta(\mathbf{x_f}^*)|\mathbf{y}, \phi^{(\mathbf{i})}) + \frac{\mathbf{1}}{\mathbf{M-1}} \sum_{\mathbf{i=1}}^{\mathbf{M}} (\mathbf{E}(\zeta(\mathbf{x_f}^*)|\mathbf{y}, \phi^{(\mathbf{i})}) - \mathbf{E}(\zeta(\mathbf{x_f}^*)))^{\mathbf{2}}$$

Computing unconditional mean and variance for each new input configuration, rows of $\mathbf{X_f^*}$, results in an estimate for $\mathbf{y_f^*}$ along with vector of variances that characterizes the uncertainty in prediction.

4.3 MAP Approach

In MAP approach, the row of Φ that results in maximum posterior log likelihood, denoted by $\phi^{(\text{MAP})}$, is used to form the prediction for $\mathbf{x_f^*}$. Specifically, $\kappa^{(\text{MAP})}$, which is a subset of $\phi^{(\text{MAP})}$, is augmented to $\mathbf{x_f^*}$ to form $\mathbf{x}_{\kappa}{}^{*\ (\text{MAP})}$. Unlike Bayesian approach, only one parameter vector, $\phi^{(\text{MAP})}$, will be used to compute mean and variance for any given input configuration. This is method is much faster and provides good estimates for the more involved Bayesian approach. The components to perform calibrated prediction using MAP approach along with their notation is introduced in Table 6.

Table 6: Components of calibrated prediction in MAP approach along with their notation.

Notation	Description
$\phi^{(\text{MAP})}$	The vector of all model parameters that results in MAP
$\kappa^{(\text{MAP})}$	The vector of calibration input configuration that results in MAP
$\sigma_s^{2\ (\text{MAP})}$	marginal variance of simulator GP (subset of $\phi^{(i)}$)
$\sigma_b^{2\ (\text{MAP})}$	marginal variance of bias-correction GP (subset of $\phi^{(i)}$)
$\sigma_\epsilon^{2\ (\text{MAP})}$	measurement variance (subset of $\phi^{(i)}$)
$\mathbf{x}_\kappa{}^{*\ (\text{MAP})}$	Augmented field input vector (augmented with κ^{MAP})
$\mathbf{X}_\kappa{}^{*\ (\text{MAP})}$	Augmented field input matrix, built using training data and $\kappa^{(i)}$
$\Sigma^{-1\ (\text{MAP})}$	Covariance matrix of $\zeta(.)$ given training data and $\phi^{(MAP)}$

Similar to the Bayesian approach to compute the mean for $\zeta(\mathbf{x_f}^*)$, the following equation is used for $\phi^{(\text{MAP})}$. The formulation of correlation vectors same as Bayesian approach but computed using parameters in $\phi^{(\text{MAP})}$. The inverse of covariance matrix of $\zeta(.)$ must also be computed for each $\phi^{(\text{MAP})}$ using training data.

$$E(\zeta(\mathbf{x_f}^*)) \approx \mu + \begin{bmatrix} R_s(\mathbf{x}_\kappa^{*(\mathbf{MAP})}, \mathbf{X_s}) \\ R_s(\mathbf{x}_\kappa^{*(\mathbf{MAP})}, \mathbf{X}_\kappa^{(\mathbf{MAP})}) + \mathbf{R_b}(\mathbf{x_f}^*, \mathbf{X_f}) \end{bmatrix}^{\mathbf{T}} .\mathbf{\Sigma^{-1}}^{(\mathbf{MAP})}.(\mathbf{y} - \mu)$$

Also similar to the Bayesian approach, to compute unconditional variance for $\zeta(.)$, the following equation is used.

$$\begin{aligned} \text{Var}(\zeta(\mathbf{x_f}^*)) &\approx \sigma_{\mathbf{s}}^{\mathbf{2}\ (\mathbf{MAP})} + \sigma_{\mathbf{b}}^{\mathbf{2}\ (\mathbf{MAP})} + \sigma_\epsilon^{\mathbf{2}\ (\mathbf{MAP})} \\ &- \begin{bmatrix} R_s(\mathbf{x}_\kappa^{*(\mathbf{MAP})}, \mathbf{X_s}) \\ R_s(\mathbf{x}_\kappa^{*(\mathbf{MAP})}, \mathbf{X}_\kappa^{(\mathbf{MAP})}) + \mathbf{R_b}(\mathbf{x_f}^*, \mathbf{X_f}) \end{bmatrix}^T .\Sigma^{-1^{(MAP)}}. \begin{bmatrix} R_s(\mathbf{x}_\kappa^{*(\mathbf{MAP})}, \mathbf{X_s}) \\ R_s(\mathbf{x}_\kappa^{*(\mathbf{MAP})}, \mathbf{X}_\kappa^{(\mathbf{MAP})}) + \mathbf{R_b}(\mathbf{x_f}^*, \mathbf{X_f}) \end{bmatrix} \end{aligned}$$

5. Application

5.1 Analytic Example with 2 Calibration Parameter

```
# priors    <- set_hyperPriors(thetaSDist  = "logbeta",      thetaSP1  = 5,  thetaSP2 = 5,
#                              alphaSDist  = "fixed",        alphaSP1  = 2,
#                              thetaBDist  = "logbeta",      thetaBP1  = 5,  thetaBP2 = 5,
#                              alphaBDist  = "fixed",        alphaBP1  = 2,
#                              sigma2SDist = "inversegamma", sigma2SP1 = 5,  sigma2SP2 = 0.2,
#                              sigma2BDist = "inversegamma", sigma2BP1 = 12, sigma2BP2 = 0.5,
#                              sigma2EDist = "inversegamma", sigma2EP1 = 5,  sigma2EP2 = 100,
#                              muBDist     = "fixed",        muBP1     = 0)
# analytic1 <- calibrate(analytic11S, analytic11F, nMCMC = 15000, nBurn = 5000,
#                        kappaDist = "normal", kappaP1 = 0.5, kappaP2 = 0.25, hypers = priors)
```

5.2 Spot Weld Example

The physical model has three inputs: gauge (G), load (L), and current (C):

- Gauge (G):
- Load (L):

- Current (C):

The simulation model has one additional input, τ that affects the amount of heat produced in the metal sheets. τ cannot be controlled in the physical experiment and its value is unknown. However it has to be specified for the simulation model as calibration input t:

- Heat generation factor τ: Factor affecting amount of heat produced

To map the spot weld data (both field and simulation data) and parameters to `FBC` input configuration, we use the dagger † superscript to distinguish process parameters and variables with `FBC` variables and parameters:

$$\begin{aligned} \mathrm{x}_1 &\longrightarrow G^\dagger \\ \mathrm{x}_2 &\longrightarrow L^\dagger \\ \mathrm{x}_3 &\longrightarrow C^\dagger \\ \kappa_1 &\longrightarrow \tau^\dagger \end{aligned}$$

5.3 Kinetic Example

5.4 Ball Example

Appendix

A.1 Prior Specification in `FBC`

Nevertheless, prior for calibration parameters is defaulted to Beta(1.1, 1.1) distribution [30]. It is close to standard uniform distribution (U(0, 1)) but densities approach to zero sharply as samples approach boundaries. This default choice has been made to ensure a somewhat non-informative prior while de-emphasizing on boundary values[31]. For all other classes of parameters reasonable priors have been specified using default values. Priors for correlation scale parameters (vectors θ_s and θ_b) have been set to Gamma(1.1, 0.1) distribution. Similarly, the priors for correlation smoothness parameters (vectors α_s and α_b) have been set to Beta(5, 2) distribution that is shifted one unit to right to span [1, 2] as is the acceptable range for moothness parameters. This choice emphasizes higher (closer to 2 than 1) smoothness parameters. Finally, the priors for marginal simulator and bias-correction and measurement error variances (scalers σ_s^2, σ_b^2, and σ_ϵ^2) are set to be Inverse Gamma(1.5, 1.5). This emphasizes very low variances and de-emphasizes higher values.

For calibration parameters, often non-informative uniform prior is used over wide but finite domain, unless expert opinion is available in the form of prior distributions. An alternative choice can be a regularizing prior that prevent from over-concentration of posterior density on boundaries of domain.

A.2 MCMC Algorithm

Implementation of Metropolis within Gibbs algorithm

[30] *Note that the range of Beta distribution is the span of [0, 1], which is the range of calibration inputs for simulator after scaling .*

[31] *Boundary values of [0, 1] corresponds to $-\infty$ and ∞ in the original scale of calibration parameters. .*

Let Φ be the matrix of parameter values (columns) indexed by MCMC iterations. Each column represents (after MCMC completes) the posterior density of a parameters. Since all parameters are included in Φ but have overlapping indices, the parameter densities (columns) are renamed to $(\phi_1, ..., \phi_d)$, where $d = 4p+3q+3$ is total number of parameters to have a unique index:

$$
\begin{array}{ccccccccc}
(\mathbf{1}) & & \dots & & & \dots & & & (\mathbf{d}) \\
\mathbf{k}_1^* \ \dots \ \mathbf{k}_q^* & \mathbf{t}_{s1}^* \dots \mathbf{t}_{s(p+q)}^* & \mathbf{a}_{s1}^* \ \dots \ \mathbf{a}_{s(p+q)}^* & \mathbf{t}_{b1}^* \ \dots \ \mathbf{t}_{bp}^* & \mathbf{a}_{b1}^* \ \dots \ \mathbf{a}_{bp}^* & \mathbf{v}_s^* & \mathbf{v}_b^* & \mathbf{v}_e^* &
\end{array}
$$

$$
\Phi = \begin{bmatrix}
k_1^{(1)} \dots k_q^{(1)} & t_{s1}^{(1)} \dots t_{s(p+q)}^{(1)} & a_{s1}^{(1)} \dots a_{s(p+q)}^{(1)} & t_{b1}^{(1)} \dots t_{bp}^{(1)} & a_{b1}^{(1)} \dots a_{bp}^{(1)} & v_s^{(1)} & v_b^{(1)} & v_e^{(1)} \\
\dots & & \dots & & \dots & & & \dots \\
k_1^{(N)} \dots k_q^{(N)} & t_{s1}^{(N)} \dots t_{s(p+q)}^{(N)} & a_{s1}^{(N)} \dots a_{s(p+q)}^{(N)} & t_{b1}^{(N)} \dots t_{b(p+q)}^{(N)} & a_{b1}^{(N)} \dots a_{b(p+q)}^{(N)} & v_s^{(N)} & v_b^{(N)} & v_e^{(N)}
\end{bmatrix}
$$

- Initialize the first row with user-given initial values:

$$
\begin{aligned}
\phi^{(1)} &= (\phi_1^{(1)},\ ...,\ \phi_d^{(1)}) \\
&= (k_1,\ ...,\ k_q,\ \theta_0^{(1)},\ ...,\ \theta_0^{(p+q)},\ \alpha_0^{(1)},\ ...,\ \alpha_0^{(p+q)},\ \theta_0^{(1)},\ ...,\ \theta_0^{(p)},\ \alpha_0^{(p)},\ ...,\ \alpha_0^{(p)},\ v_{s0},\ v_{b0},\ v_{e0})
\end{aligned}
$$

- At each iteration $i \in (2, ..., N)$, and to update j-th parameter ($j \in (1, ..., d)$ and first $(j-1)$ parameters are already updated):

  1. Propose a new value for $\phi_j^{(i)}$ based on its last update [32]. $\phi_j^{(i-1)}$, called Metropolis Update (MU):

  $$\phi_j^* = \mathcal{N}(\phi_j^{(i-1)}, \sigma_p^2)$$

  where σ_p^2 is adaptively adjusted [33] to ensure (faster) convergence.

  2. Form the parameter vector ϕ^* based on MU:

  $$
  \begin{aligned}
  \phi^{(last)} &= (\phi_1^{(i)}, ..., \phi_{j-2}^{(i)},\ \phi_{j-1}^{(i)},\ \phi_j^{(i-1)},\ ...,\ \phi_d^{(i-1)}) \\
  \phi^* &= (\phi_1^{(i)}, ..., \phi_{j-1}^{(i)},\ \phi_j^*,\ \phi_{j+1}^{(i-1)},\ ...,\ \phi_d^{(i-1)})
  \end{aligned}
  $$

  3. Draw a random sample u from $U(0,1)$ and take its log: $\ln(u)$

  4. Compute the difference between the log of joint posterior density given current and last parameter vectors:

  $$h(\phi^*,\ \phi^{(last)}) = \ln(L(\mathrm{y}|\phi^*)) + \ln(\mathcal{P}[\phi^*]) - \ln(L(\mathrm{y}|\phi^{(last)})) + \ln(\mathcal{P}[\phi^{(last)}])$$

  5. If $h(\phi^*,\ \phi^{(last)}) > \ln(u)$, set:

  $$\phi_j^{(i)} = \phi_j^*$$

  otherwise,

  $$\phi_j^{(i)} = \phi_j^{(i-1)}$$

[32] If it is first parameter, the last update is the last row ($\phi^{(i-1)}$

[33] Every 50 iterations, acceptance rate (AR) is computed. If $AR < 0.44$, proposal variance σ_p^2 is decreased, and vice versa.It has been shown that for one-dimensional proposals used in Metropolis within Gibbs algorithm, the optimal acceptance rate is 0.44 (TODO: citation).

6. The update vector now is:

$$\phi^{(last)} = (\phi_1^{(i)}, ..., \phi_{j-1}^{(i)},\ \phi_j^{(i)},\ \phi_{j+1}^{(i-1)},\ ...,\ \phi_d^{(i-1)})$$

- If all parameters are updated, go to next iteration of i
- When iterations of i is completed, return the matrix of Φ that contains joint posterior density distribution of all parameters. Marginal distribution of each parameter can be used for point prediction and uncertainty quantification (credible interval).

References

McKay, M.D.; Beckman, R.J.; Conover, W.J. (May 1979). "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code". Technometrics. American Statistical Association. 21 (2): 239–245. doi:10.2307/1268522. ISSN 0040-1706. JSTOR 1268522. OSTI 5236110.