FBC: Full Bayesian Calibration

Contents

Introduction 1

1. Using `FBC` 2

2. Calibration Model 14

3. Parameter Estimation 17

4. Calibrated Prediction 19

5. Implementation 20

6. Application 20

Appendix 21

References 22

Introduction

`FBC` is a package that uses the data from both field experiment and computer simulation to calibrate the simulator model [1]. A field experiment relates a set of experimental variables as inputs to a response as output. A computer simulation also relates the same experimental variables as inputs to the response as output but also include calibration inputs. These inputs either 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. 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 `FBC` package is a based on the well-known Kennedy and O'Hagan (KOH) calibration model (2001). In the KOH model, field and simulator data are stacked in a single model, where simulator response is fitted using a Gaussian Process (GP) that models the simulator model and field response is fitted using a three-component model that accounts for simulator model using simulator GP, bias-correction using another independent GP, and a Gaussian random error term representing measurement error. Using two independent GPs and a Gaussian random error introduces new hyperparameters to the calibration model that must be estimated along with calibration parameters. In its original formulation, KOH formulates a hierarchical Bayesian framework with two sequential phases. In the first phase, model hyperparameters are predicted 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.

[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.*

Neither KOH's original formulation nor later suggestions are fully Bayesian as they use MLE methods to estimate hyperparameters and fix the hyperparameters in the second phase which ignores the added uncertainty of the estimated hyperparameters, largely due to computational infeasibility (Kennedy & O'Hagan, 2001; Higdon et al., 2004; Liu et al., 2009). On the other hand, `FBC` runs a fully Bayesian model that includes both calibration parameters and model hyperparameters in its Bayesian framework. 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 using prior specifications. All common prior distribution are implemented in `FBC` to allows for high degree of flexibility in specifying the prior information or the lack thereof. `FBC` runs a variation of Markov Chain Monte Carlo (MCMC) algorithm called Metropolis-Within-Gibbs algorithm to find the posterior distribution of calibration parameters and model hyperparameters. Furthermore, `FBC` enables calibrated prediction for new input configurations using the calibration model.

The current vignette is structured into following sections: The first section, Using `FBC`, 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. 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 justified. In the third section, Parameter Estimation, the theoretical results to characterize the posterior distribution of model parameters are presented along with procedure for MCMC-based estimation of parameters. In the fourth section, Calibrated Prediction, the theoretical results to derive calibrated predictions for new input configuration using the estimated parameters are offered. 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. And finally, the Appendix provides deeper discussion and further details to some of the concepts presented in the sections.

1. Using `FBC`

`FBC` package has two main public functions: `calibrate()` and `predict()`. As function names suggest, `calibrate()` takes the field and simulation training data to calibrate the simulator model and `predict` takes a new field input configuration and the calibration model object to predict the field response and its associated uncertainty. In addition, `FBC` has four more public functions to help with prior specifications, summarizing, and visualization of calibration. Furthermore, there are three more public function that are not directly used by user to build calibration model but provide functionalities that are not offered by base R. This section explains how to use the mentioned functions and what to expect in their use and results using a simple pedagogic example. Throughout this section, the user is expected to be well-versed with calibration model in general and KOH model in particular. It is designed to be a self-contained section on the usage of `FBC` package but can be augmented with section six, Applications, to further examine the functionalities and limitation of the package. Less experienced users are encouraged to start from next section which aims to explain the theory behind calibration models and calibrated predictions in more details.

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.

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

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

```
library(FBC)
```

1.2 Data:

Building a calibration model requires data from both field experiment and computer simulation. To focus on the functionality of the package, we use a simple pedagogic example as experimental setting. In the field 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 a falling ball is modelled by a mathematical equation that relates h to t, however, one must also provide gravity (g) as calibration input. Note that in the field experiment, the earth's gravity is fixed but unknown to the experimenter [2]. 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 derived from introductory physics results.

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

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 (McKaThe `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` [4].

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

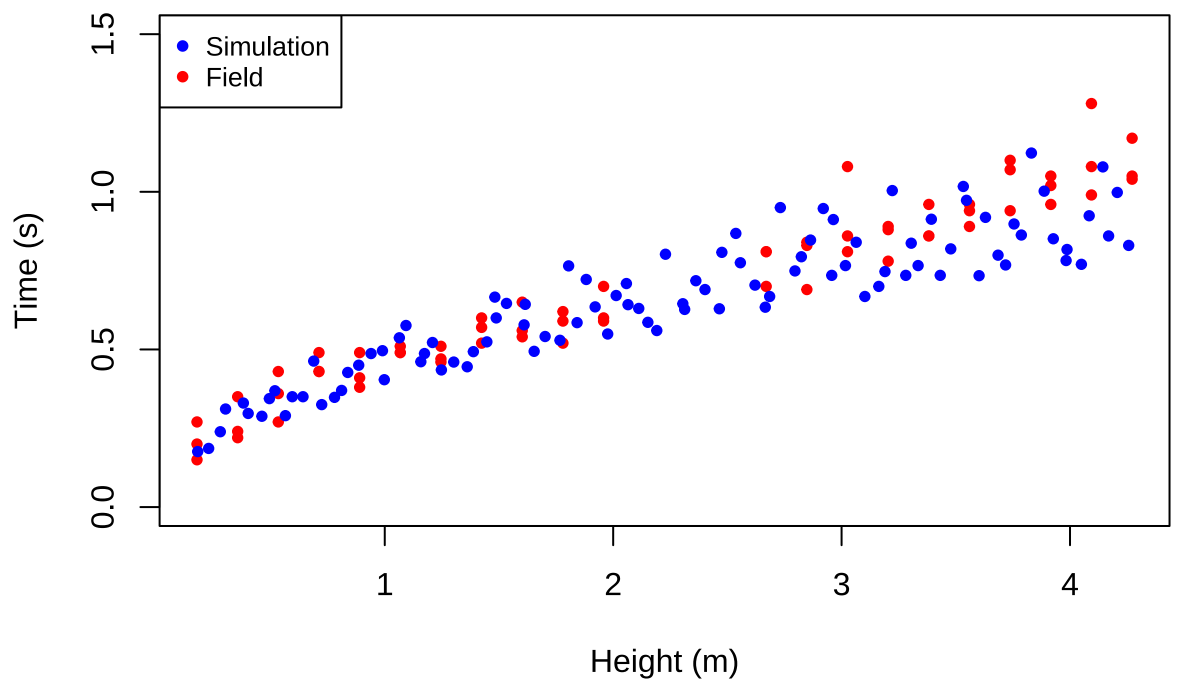


Figure 1: The time versus height plot for both field experiment and simulation.

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 to the details of a complex mathematical model. 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 parameters. Other than data arguments which must be supplied by user, all other arguments have reasonable default values for ball example [5].

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

The first and second arguments `sim` and `field` must be supplied by user. As mentioned earlier, both must be either in matrix format representing the simulation and field data respectively. In both `ballSim` and `ballField` of our ball example, the first column represent the response **t** and the second column represent experimental input **h**. Additionally, `ballSim` has a third column that represents calibration input **g**. Calibration inputs are implicit in field experiment and are absent in the data matrix `ballField`.

[5] *Running the `calibrate()` function with reasonable number of MCMC runs is too lengthy to run while knitting the current vignette. However, this command is 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 and last 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 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", "gaussian", "gamma", "beta", "lognormal", "logistic", "betashift", "exponential", "inversegamma", "jeffreys", and "fixed" [6]. 2) Initial value, which is double representing starting point of the parameter in MCMC algorithm (suffixed with "init" in argument names). 3) First distribution parameter (suffixed with "p1" in argument names) and 4) second distribution parameter (suffixed with "p2" in argument names) are doubles representing the two parameters of the chosen distribution [7], [8]. 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	Initial Value	Shape Parameters
True field calibration inputs (κ)	`kappaDist`	`kappaInit`	`kappaP1` & `kappaP2`
Simulation GP scale (θ_s)	`thetaSDist`	`thetaSInit`	`thetaSP1` & `thetaSP2`
Simulation GP smoothness (α_s)	`alphaSDist`	`alphaSInit`	`alphaSP1` & `alphaSP2`
Bias-correction GP scale (θ_b)	`thetaBDist`	`thetaBInit`	`thetaBP1` & `thetaBP2`
Bias-correction GP smoothness (α_b)	`alphaBDist`	`alphaBInit`	`alphaBP1` &`alphaBP2`
Simulation marginal variance (σ_s^2)	`sigma2SDist`	`sigma2SInit`	`sigma2SP1` & `sigma2SP2`
Bias-correction marginal variance (σ_b^2)	`sigma2BDist`	`sigma2BInit`	`sigma2BP1` & `sigma2BP2`
Measurement error variance (σ_ϵ^2)	`sigma2EDist`	`sigma2EInit`	`sigma2EP1` & `sigma2EP2`
Bias-correction mean (μ_b)	`muBDist`	`muBInit`	`muBP1` & `muBP2`

[6] *"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.*

[7] *For example, if "gaussian" 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.*

[8] *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.*

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 [9]. 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. 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 `kappaInit`, `kappaP1`, and `kappaP2` must be also scaler, or can be a vector of length five that supplies the distribution types for all calibration parameters and `kappaInit`, `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. This will put the `calibrate()` in interactive mode and will show the percentage of the MCMC draws along with sample draws [10].

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"    "priorFns"    "proposalSD"
```

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   6.16    5.14    0.69    1.94    1.84    0.68    1.90    0.09    0.30    0.09 -0.35
> 2   8.72    4.92    1.04    1.94    1.97    0.17    1.81    0.09    0.66    0.09  0.24
> 3  13.70    4.35    0.76    1.90    1.93    0.43    1.79    0.10    0.44    0.08  0.15
```

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.

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

[9] *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 ($U(0,1)$.*

[10] *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 purposes.*

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

There are ten more elements in `caqlMod` other than matrix `Phi`, which are listed here along with a brief description. The data frame `estimates`, which provides a summary table of all model parameters, includes the 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. The function list `priorFns` includes the prior functions that are created during calibration based on given prior specifications. And finally, `proposalSD` is a numeric vector that represents the final standard deviation of proposal for each parameter.

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.

```r
predsMAP
> $pred
> [1] 0.6922523 0.7301342
>
> $se
> [1] 0.077 0.077
```

```
predsBayes
> $pred
> [1] 0.695 0.733
>
> $se
> [1] 0.071 0.071
```

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_hyperParameters()` function. All arguments of this function, which collectively specify the priors for all hyperparameters, have reasonable default values. Therefore `set_hyperParameters()` can be used without arguments to set the hyperparameters. In fact, the default value of the `hyper` argument in `calibrate()` is the function `set_hyperParameters()` without any argument. Nevertheless, when there is prior belief about structure of correlation structures (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. The first parameter of the beta distribution, the initial value for these parameters, and all other parameter specification remain unchanged.

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

1.6 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.

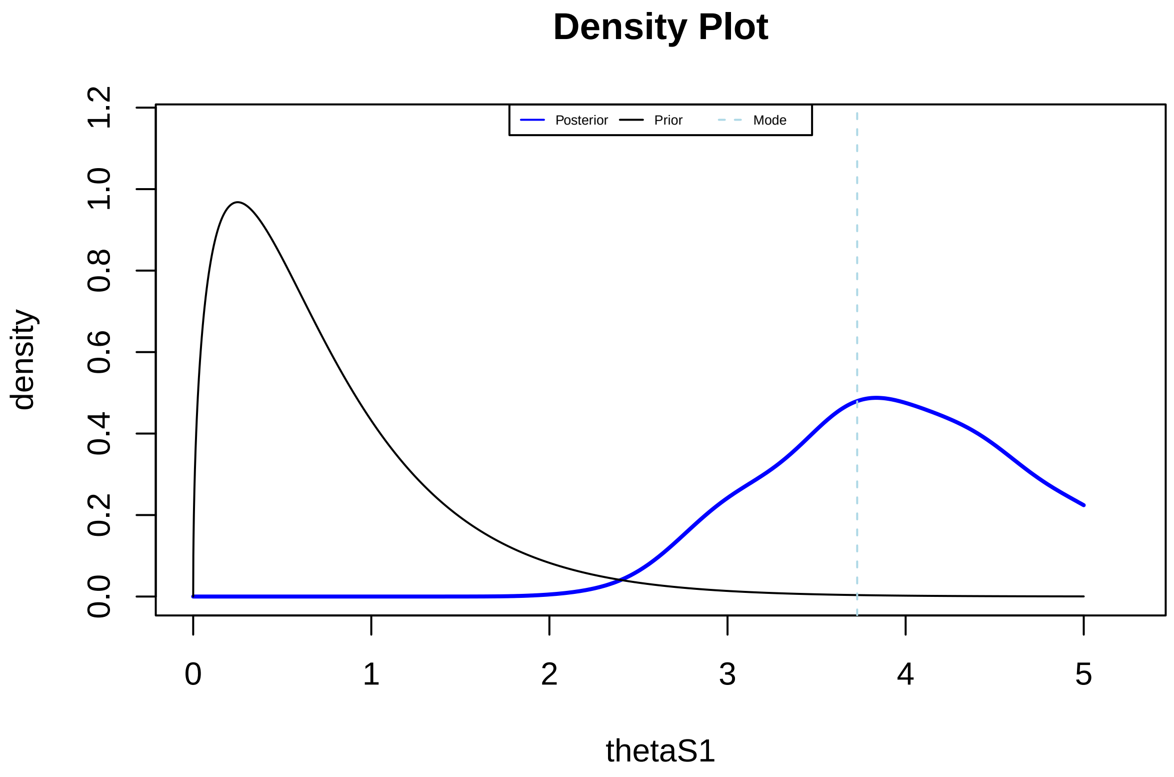
```
calModSum <- summary(calMod)
print(calMod)
>          mean median  mode  lwr50  upr50  lwr80  upr80    sd
> kappa1  9.118  8.594 7.221 7.521 10.233 6.914 12.316 1.995
> thetaS1 4.120  4.044 3.728 3.545  4.576 3.049  5.223 0.837
> thetaS2 0.741  0.710 0.610 0.599  0.839 0.510  1.017 0.210
> alphaS1 1.896  1.906 1.921 1.865  1.927 1.837  1.946 0.046
> alphaS2 1.862  1.862 1.817 1.817  1.911 1.780  1.946 0.065
> thetaB1 0.601  0.520 0.196 0.287  0.791 0.170  1.190 0.418
> alphaB1 1.793  1.824 1.772 1.719  1.905 1.572  1.961 0.147
> sigma2S 0.091  0.089 0.082 0.082  0.098 0.076  0.107 0.012
> sigma2B 0.785  0.611 0.280 0.354  0.984 0.267  1.545 0.627
> sigma2E  0.100  0.100  0.103  0.089  0.109  0.081  0.120 0.016
> muB     -0.119 -0.118 -0.140 -0.257  0.021 -0.349  0.136 0.192
```

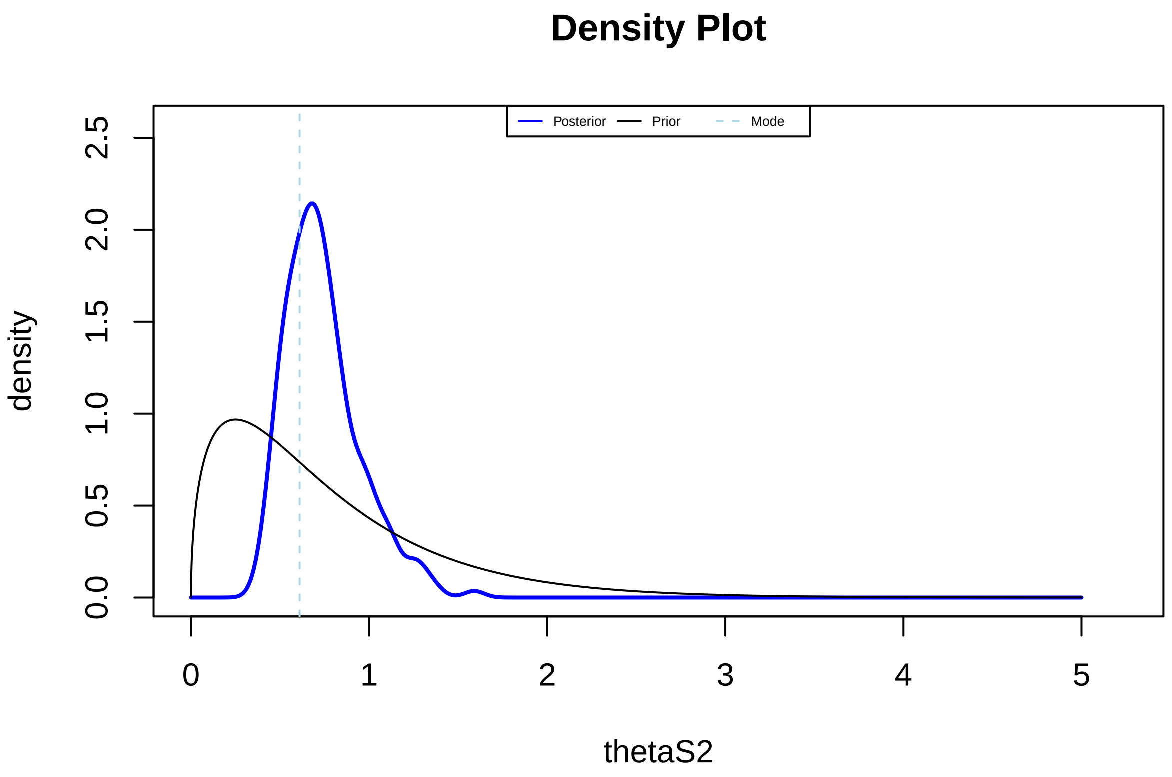
1.7 Visualization of Calibration Model

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 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.

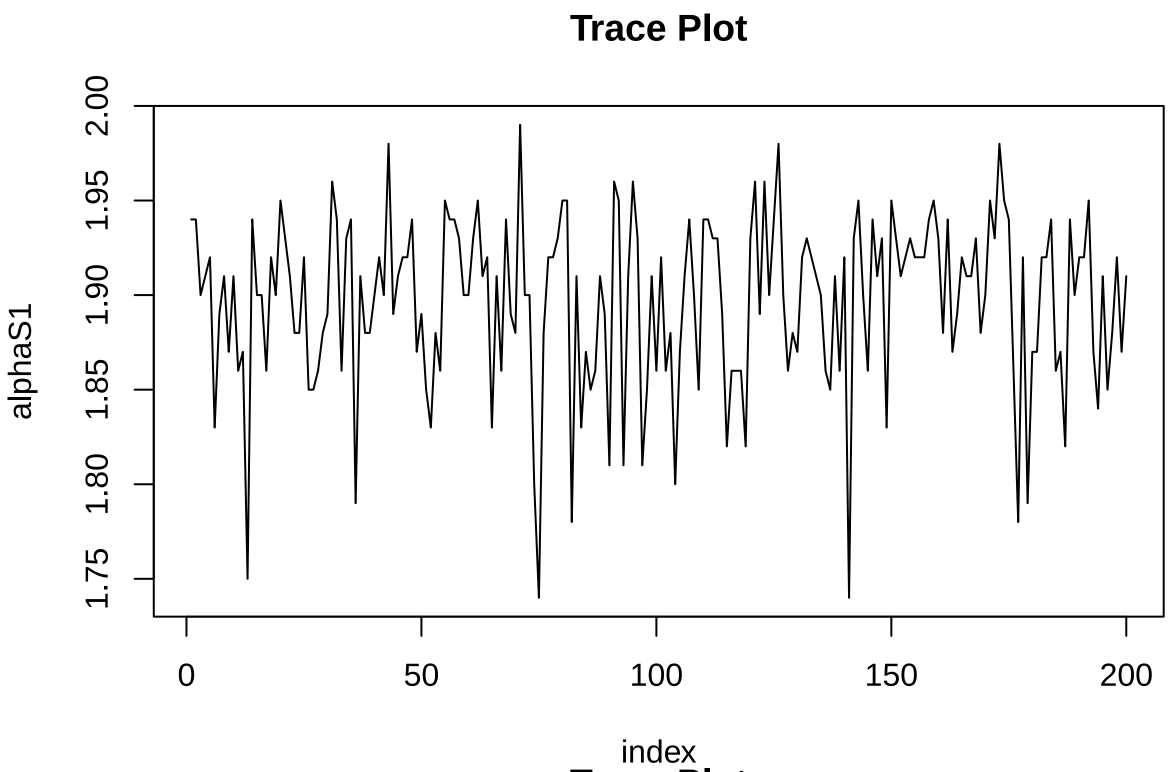
```
# Note that there are two correlation scale parameters in simulator GP and there will be two plots
plot(calMod, parameter = "thetaS")
```

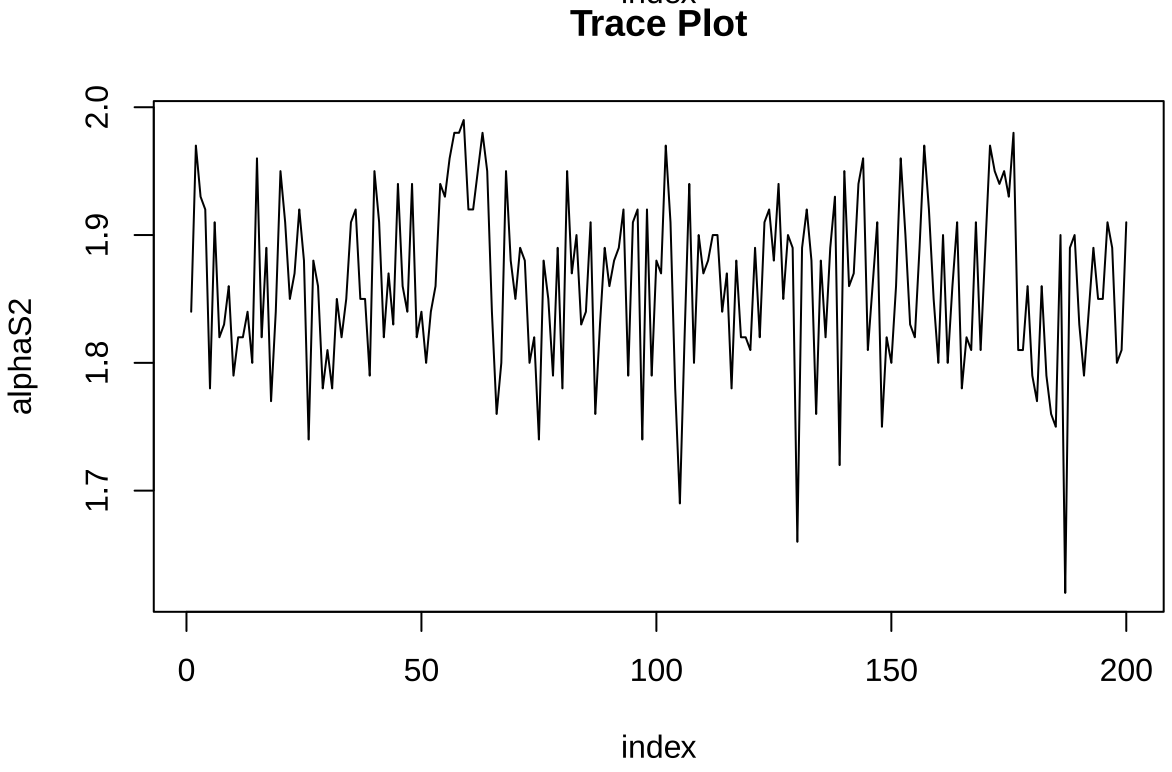




Using “trace” as `type` argument, `plot()` will plot the progression of the given parameter as MCMC draws are taken. It is similar to the time series of the parameter but indexed with number of iteration in MCMC rather than time. The trace plot can be used to determine whether there is good mixing in 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.

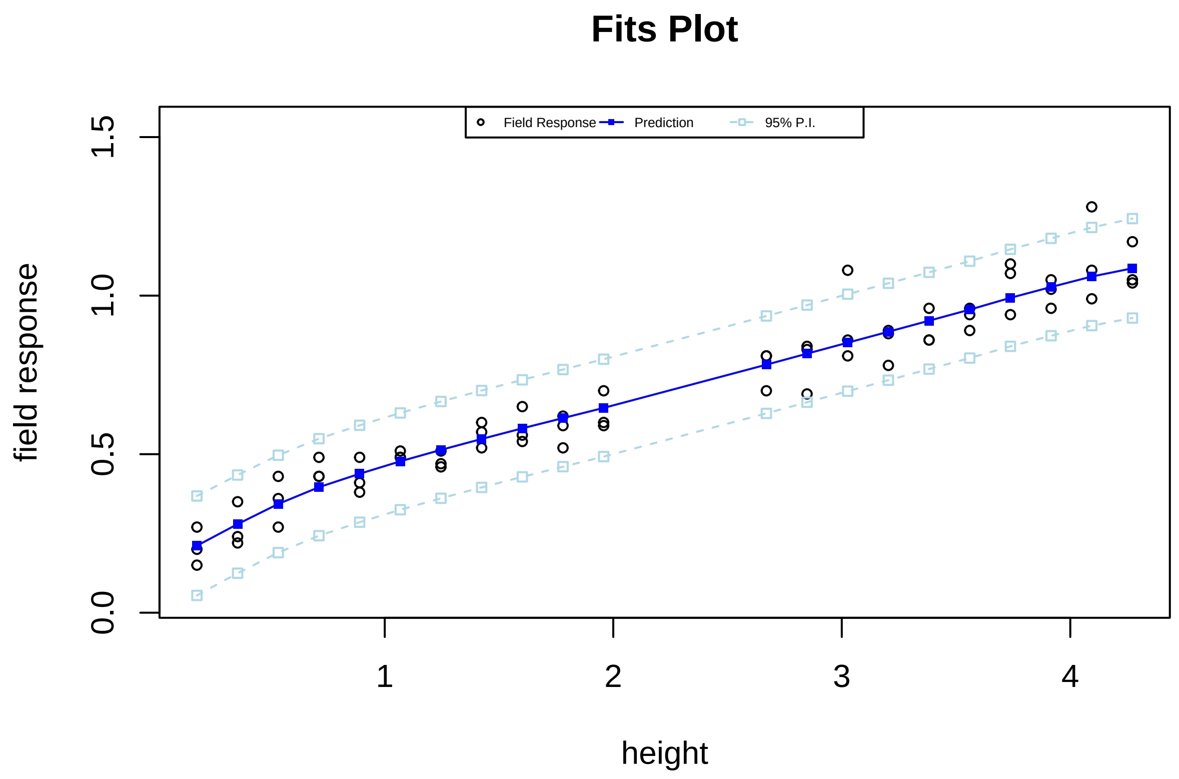
```r
# Note that there are two correlation smoothness parameters in simulator GP and there will be
# two plots
plot(calMod, parameter = "alphaS", type = "trace")
```





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")
```



There are also three more exported functions in FBC package that are not required to build calibration models or to perform calibrated prediction. These functions are used internally in the package but since they offer functionalists that are not supported in base R, they are exported for use.

1.8 Computing Correlation:

This function is used to compute the correlation between rows of two given matrices assuming a power exponential correlation family structure. This family is a generalization of correlation, where distinct scale and smoothness hyperparameters are used for different dimensions of the given data (columns of given matrices) when computing the correlation. In FBC, `correlation()` function treats different dimensions using separate scale and smoothness parameters. Note that only the first matrix, characterized by argument `X` must be supplied and the default value for the second matrix, characterized by `Y` is `NULL`. When only a single matrix is supplied, `correlation()` will compute the correlation of that matrix with itself. Other than the given matrix or matrices, which must have same number of columns, user must supply two vectors with `theta` and `alpha` arguments to characterize scale and smoothness parameters. The length of both vectors either must be same and equal to the number of columns in given matrices, or they must be scaler in which case for all dimensions same values of scale or smoothness will be used.

```r
X     <- matrix(c(1, 3, 5,
                  2, 2, 6,
                  1, 4, 1), nrow = 3, byrow = TRUE)

Y     <- matrix(c(7, 3, 0,
                  2, 2, 4), nrow = 2, byrow = TRUE)

sc    <- c(1, 2, 3) # scale parameters of correlation structure
sm    <- c(2, 1, 2) # smoothness parameters of correlation structure

# correlation of a matrix with itself
round(correlation(X, theta = sc, alpha = sm), 5)
>          [,1]    [,2] [,3]
> [1,] 1.00000 0.00248    0
> [2,] 0.00248 1.00000    0
> [3,] 0.00000 0.00000    1

# correlation between two matrices
round(correlation(X, Y, theta = sc, alpha = sm), 5)
>       [,1]    [,2]
> [1,]    0 0.00248
> [2,]    0 0.00001
> [3,]    0 0.00000
```

1.9 Building Prior Functions:

This function will create a prior function based on given distribution and distribution parameters. The function arguments are `prior`, which characterize the distribution family of the prior and two other the arguments, `p1` and `p2` that characterize the parameters of the chosen distribution. The `prior` argument can take a character string value from "uniform", "gaussian", "gamma", "beta", "lognormal", "logistic", "betashift", "exponential", "inversegamma", "jeffreys", "fixed". For example for `prior = "gamma"`, `p1` and `p2` determine shape and scale of the gamma distribution, or for Gaussian distribution, `p1` and `p2` determine mean and standard deviation of the Gaussian distribution. When "fixed" is used for `prior`, the prior function is simply `x = 1`. The output of `prior_builder()` is a function that given a value, computes the probability density of the chosen distribution. To reduce the load of internal computation during creation of the calibration model, `prior_builder()` computes the log of density and if used externally must be transformed.

```
# create a prior function for beta(2, 5). Note that the function compute log of priors and must be tran
pr_fun <- prior_builder(prior = "beta", p1 = 2, p2 = 5)
round(exp(pr_fun(c(-1, 0, 0.1, 0.5, 0.9, 1, 2))), 3)
> [1] 0.000 0.000 1.968 0.938 0.003 0.000 0.000

# create a prior function for a Uniform distribution with lower bound of -10, and upper bound of 10
pr_fun <- prior_builder(prior = "uniform", p1 = -10, p2 = 10)
round(exp(pr_fun(c(-11, -5, 0, 4, 10, 12))), 3)
> [1] 0.00 0.05 0.05 0.05 0.05 0.00

# create a prior function for Gaussian distribution with mean of 1 and standard deviation of 2
pr_fun <- prior_builder(prior = "gaussian", p1 = 1, p2 = 2)
round(exp(pr_fun(c(-9, -5, -3, -1, 1, 3, 5, 7, 11))), 3)
> [1] 0.000 0.002 0.027 0.121 0.199 0.121 0.027 0.002 0.000
```

1.10 Estimating the Mode of a Continious Variable

The function `pmode()` computes an estimate of the mode for a continuous distribution. It works similar to a histogram, in which the domain of the distribution is broken into same length bins. For each bin, the draws of the distribution that fall within that bin is counted and at the end the mean of the bin with highest count is returned as estimated mode. The function also takes the number of bins through `breaks` argument. The default value is `NULL`, which makes `pmode` to determine the number of required bins dynamically based on number of data points and the domain of the distribution.

```
# find the estimated mode of a vector
vec <- runif(100, 0, 10)
pmode(vec)
> [1] 4.016289
pmode(vec, breaks = 10)
> [1] 5.683952
```

2. Calibration Model

Calibration model is statistical model that represent both field and simulator response as function of input configuration. In this section, the theory behind building a calibration model is explained along with the notation to represent the components of the calibration model [11].

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 [12], 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 [13]
$\mathbf{X_f}$	Field input matrix $((n \times p))$	$\mathbf{h}$ [14]
y_f	Univariate field response	t (scaler)
$\mathbf{y_f}$	Vector of n univariate field response	$\mathbf{t}$ (vector of length 63)

[11] *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.*

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

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

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

Notation	Description	Ball Example
κ	Vector of true calibration inputs in field experiment	True value of gravity κ_1[15]
$\mathbf{x}_\kappa$	Augmented field input vector containing $(p+q)$ inputs	$(h,\, \kappa_1)$ (vector of length 2)
$\mathbf{X}_\kappa$	Augmented field input matrix $((n \times (p+q)))$[16]	$[\mathbf{h} \quad \boldsymbol{\kappa_1}]$
$\mathbf{x_s}$	Simulation input vector containing $(p+q)$ inputs	$(h,\, g)$ (vector of length 2)
$\mathbf{X_s}$	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 $\delta_\kappa(\mathbf{x_f})$ [17] 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.

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

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

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

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 [18]. Moreover, σ_s^2 and σ_b^2 denote marginal variance of simulator and bias-correction GPs, and $R_s(.,.)$ [19] and $R_b(.,.)$ [20] are correlation matrix of simulator and bias-correction GP.

[15] *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.*

[16] *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.*

[17] *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 $\delta_\kappa(.)$.*

[18] *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. However, the mean of the bias-correction GP is dependent on extent of simulator model inadequacy and will vary from one model to another. 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.*

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

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

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 the both flexibility and interpretability. Assuming $\mathbf{x}$ and $\mathbf{x}'$ are two rows of full input matrix [21] 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 [22].

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

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

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

[22] *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.*

$\mathbf{x}$ 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.

$$\begin{aligned} y &= \zeta(\mathbf{x}) \\ \mathrm{y} &= \zeta(X) \end{aligned}$$

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 $C(.,.)$ result in a mean vector μ and a covariance matrix C.

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

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

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

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

`FBC` employs a full Bayesian approach to jointly estimate all parameters. 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 Distribution:

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 [23]. 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 [24].

[23] *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.*

[24] *Assuming iteration i in MCMC algorithm, the updated parameter in the current row is updated.*

$$\mathcal{P}[\phi] \;\propto\; \prod_{i=1}^{q} \mathcal{P}[\kappa_i] \qquad \text{(priors for calibration parameters)}$$

$$\times \prod_{i=1}^{p+q} \mathcal{P}[\theta_{si}] \;\times\; \prod_{i=1}^{p+q} \mathcal{P}[\alpha_{si}] \;\times\; \mathcal{P}[\sigma_s^2] \qquad \text{(priors for } \eta(.) \text{ GP)}$$

$$\times \prod_{i=1}^{p} \mathcal{P}[\theta_{bi}] \;\times\; \prod_{i=1}^{p} \mathcal{P}[\alpha_{bi}] \;\times\; \mathcal{P}[\sigma_b^2] \;\times\; \mathcal{P}[\mu_b] \qquad \text{(priors for } \delta_\kappa(.) \text{ GP)}$$

$$\times \; \mathcal{P}[\sigma_\epsilon^2] \qquad \text{(prior for measurement error variance)}$$

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 Psterior 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.

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

$$= -\frac{1}{2}\log(|\Sigma|) - \frac{1}{2}(\mathbf{y}-\mu).\mathbf{\Sigma^{-1}}.(\mathbf{y}-\mu)^{\mathbf{T}} \qquad \text{(log liklihood given full response)}$$

$$+ \; \sum_{i=1}^{q} \mathcal{P}[\kappa_i] \qquad \text{(priors for calibration parameters)}$$

$$+ \; \sum_{i=1}^{p+q} \mathcal{P}[\theta_{si}] + \sum_{i=1}^{p+q} \mathcal{P}[\alpha_{si}] \;+\; \mathcal{P}[\sigma_s^2] \qquad \text{(priors for } \eta(.) \text{ GP)}$$

$$+ \; \sum_{i=1}^{p} \mathcal{P}[\theta_{bi}] + \sum_{i=1}^{p} \mathcal{P}[\alpha_{bi}] \;+\; \mathcal{P}[\sigma_b^2] + \mathcal{P}[\mu_b] \qquad \text{(priors for } \delta_\kappa(.) \text{ GP)}$$

$$+ \; \mathcal{P}[\sigma_\epsilon^2] \qquad \text{(prior for measurement error variance)}$$

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:

The numerical method used in `FBC` is a version Markov Chain Monte Carlo (MCMC) algorithm to draw samples from the joint posterior distribution of the parameter space. In current implementation of `FBC` Metropolis-Within-Gibbs MCMC algorithm is used with adaptive proposal to build `Phi` matrix row by row. The first row of matrix `Phi` (the initial vector ϕ) is given by user to start the MCMC algorithm. Then, the algorithm creates a Markov chain by updating parameters in each iteration according to a proposal scheme. Next, given the updated parameter vector and data, the posterior log likelihood can be computed according to above equation. If joint posterior density is larger than the density of a random draw from standard uniform distribution, the algorithm keeps the given parameter configuration by updating the matrix `Phi` [25], otherwise updates the `Phi` matrix by last value of that parameter [26]. After updating the parameters one by one, either accepting or rejecting the new updates, an iteration of MCMC is complete which corresponds to a row in matrix `Phi`. The detailed MCMC algorithm is presented in the Appendix A.2.

In the end, each row of matrix `Phi` 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 [27] or calibrated predictions.

4. Calibrated Prediction

As mentioned earlier, building a calibration model results in a `fbc` object, which contains matrix `Phi` as a MCMC-based sample from joint posterior distribution of all parameters of the model. This sample can be used to to predict the physical response, simulator response, and bias for new input configurations. in `FBC`, two methods have been implemented for prediction. In the first method `Bayesian`, which is the default value of `method` argument in `predict()` function, all joint samples (all rows of matrix `Phi`) are used to form a sample distribution for the new input configuration. Using the distribution, a point prediction is made along with the uncertainty associated with it. In the second method "MAP", the row of matrix `Phi` that results in maximum posterior log likelihood, is chosen to form the point prediction.

4.1 Setup:

The input data for the `predict()` function is taken by two arguments: `object` argument which takes a `fbc` object as calibration model and `newdata` argument which takes the new input configuration data in matrix format. Each row represents a new input configuration and columns represent experimental variables. It is similar to the `field` argument of the `calibrate()` function, except the first column in `field`, which is the physical response. `predict()` aims to predict the physical response for all row of `newdata` in the form of a vector of predicted responses and a vector of standard error. Both vectors will be the same length as number of rows in `newdata`. The third argument `method` characterized the method of prediction and must be chosen from one of the character string "Bayesian" and "MAP".

[25] *Assuming iteration i in MCMC algorithm, the updated parameter in the current row is updated.*

[26] *Note that MCMC algorithm starts from the second iteration and first row of `Phi` is supplied by user through initial values for the parameters. This will enable the algorithm to use the values in the previous rows.*

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

4.2 Bayesian Method

In the Bayesian method, the posterior distribution of the response, conditional on hyperparameter estimates (matrix `Phi`), is also a GP

$$E(y_f(\mathbf{x_f}^*)|\phi) = \mu_{\mathbf{b}} + \begin{bmatrix} \mathbf{y_f} \\ \mathbf{y_s} \end{bmatrix}$$

4.3 MAP Method

5. Implementation

5.1 Metropolis-Within-Gibbs Algorithm Imlementation

5.2 Adaptive Proposal

Furthermore, input matrix is scaled according to mean and standard deviation of columns of the input matrix (consisting of experimental and calibration values) such that X_s columns span $[0, 1]$.

6. Application

6.1 Analytic Example:

6.2 Ball Example:

6.3 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:

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

6.4 Kinetic Example

Appendix

A.1 Prior Specification in FBC:

Nevertheless, prior for calibration parameters is defaulted to Beta(1.1, 1.1) distribution [28]. 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[29]. 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

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:

$$
\begin{array}{c}
\begin{array}{ccccccccccccccccccc}
(\mathbf{1}) & & & & ... & & & & & ... & & & & & & & & & (\mathbf{d}) \\
\mathbf{k}_1^* & ... & \mathbf{k}_q^* & \mathbf{t}_{s1}^* & ... & \mathbf{t}_{s(p+q)}^* & \mathbf{a}_{s1}^* & ... & \mathbf{a}_{s(p+q)}^* & \mathbf{t}_{b1}^* & ... & \mathbf{t}_{bp}^* & \mathbf{a}_{b1}^* & ... & \mathbf{a}_{bp}^* & \mathbf{v}_s^* & \mathbf{v}_b^* & \mathbf{v}_e^*
\end{array} \\
\Phi = \begin{bmatrix}
k_1^{(1)} & ... & k_q^{(1)} & t_{s1}^{(1)} & ... & t_{s(p+q)}^{(1)} & a_{s1}^{(1)} & ... & a_{s(p+q)}^{(1)} & t_{b1}^{(1)} & ... & t_{bp}^{(1)} & a_{b1}^{(1)} & ... & a_{bp}^{(1)} & v_s^{(1)} & v_b^{(1)} & v_e^{(1)} \\
& ... & & & & & ... & & & & & & ... & & & & & ... \\
k_1^{(N)} & ... & k_q^{(N)} & t_{s1}^{(N)} & ... & t_{s(p+q)}^{(N)} & a_{s1}^{(N)} & ... & a_{s(p+q)}^{(N)} & t_{b1}^{(N)} & ... & t_{b(p+q)}^{(N)} & a_{b1}^{(N)} & ... & a_{b(p+q)}^{(N)} & v_s^{(N)} & v_b^{(N)} & v_e^{(N)}
\end{bmatrix}
\end{array}
$$

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

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

- 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 [30]. $\phi_j^{(i-1)}$, called Metropolis Update (MU):

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

     where σ_p^2 is adaptively adjusted [31] 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,

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

  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.

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

[31] 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).