Package: funGp (via r-universe)

August 25, 2024

Type Package

Title Gaussian Process Models for Scalar and Functional Inputs

Version 1.0.0

Maintainer Jose Betancourt <fungp.rpack@gmail.com>

Description Construction and smart selection of Gaussian process models for analysis of computer experiments with emphasis on treatment of functional inputs that are regularly sampled. This package offers: (i) flexible modeling of functional-input regression problems through the fairly general Gaussian process model; (ii) built-in dimension reduction for functional inputs; (iii) heuristic optimization of the structural parameters of the model (e.g., active inputs, kernel function, type of distance). An in-depth tutorial in the use of funGp is provided in Betancourt et al. (2024) <doi:10.18637/jss.v109.i05> and Metamodeling background is provided in Betancourt et al. (2020) <doi:10.1016/j.ress.2020.106870>. The algorithm for structural parameter optimization is described in <https://hal.science/hal-02532713>.

Note research product of the RISCOPE project (ANR, project No.16CE04-0011) <https://perso.math.univ-toulouse.fr/riscope/>.

License GPL-3

URL https://djbetancourt-gh.github.io/funGp/, https://github.com/djbetancourt-gh/funGp

BugReports https://github.com/djbetancourt-gh/funGp/issues

Depends R (>= 3.5.0)

Imports methods, foreach, knitr, scales, microbenchmark, doFuture, doRNG, future, progressr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Collate '0_funGp_Doc.R' '0_show_Doc.R' '0_summary_Doc.R' '8_outilsCode.R' '3_ant_admin.R' '2_fgpKern_Class.R' '2_fgpProj_Class.R' '1_fgpm_Class.R' '1_Xfgpm_Class.R' '3_ant_search.R' '3_training_F.R' '3_training_S.R' '3_training_SF.R' '4_prediction_F.R' '4_prediction_S.R' '4_prediction_SF.R' '5_simulation_F.R' '5_simulation_S.R' '5_simulation_SF.R' '6_updating.R' '7_blackBoxFunctions.R' '7_checkingFunctions.R' '7_correlFunctions.R' '7_dimRedFunctions.R' '7_distanceFunctions.R' '7_plottingFunctions.R' '7_plottingFunctionsStandard.R' '8_outilsStats.R' '8_precalculated_Xfgpm_objects.R'

Repository https://djbetancourt-gh.r-universe.dev

RemoteUrl https://github.com/djbetancourt-gh/fungp

RemoteRef HEAD

RemoteSha 1cf8c4974cceb272d45f69a87a6abc8488718f9f

Contents

funGp-package *Gaussian Process Models for Scalar and Functional Inputs*

Description

Construction and smart selection of Gaussian process models for analysis of computer experiments with emphasis on treatment of functional inputs that are regularly sampled. Smart selection is based on Ant Colony Optimization ACO algorithm.

Base functionalities

- **Main methods**
  fgpm: creation of funGp regression models
  predict,fgpm-method: output estimation at new input points based on a funGp model
  simulate,fgpm-method: random sampling from a funGp Gaussian process model
  update,fgpm-method: modification of data and hyperparameters of a funGp model
- **Plotters**
  plot,fgpm-method: validation plot for a `fgpm` model
  plot.predict.fgpm: plot of predictions based on a `fgpm` model
  plot.simulate.fgpm: plot of simulations based on a `fgpm` model

Model selection

- **Main method**
  fgpm_factory: structural parameter optimization
- **Functions for pre-optimization**
  decay: regularized initial pheromones
  decay2probs: normalized initial pheromones
- **Plotters post-optimization**
  plot,Xfgpm-method: plot of the evolution of the algorithm with `which = "evolution"` or of the absolute and relative quality of the optimized model with `which = "diag"`
- **Correction post-optimization of input data structures**
  which_on: indices of active inputs in a model structure delivered by fgpm_factory
  get_active_in: extraction of active input data based on a model structure delivered by fgpm_factory

Useful material

- **Manual:** funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs (doi:10.18637/jss.v109.i05)
- **Paper:** - Gaussian process metamodeling of functional-input code for coastal flood hazard assessment (doi:10.1016/j.ress.2020.106870)
- **Tech. report:** Ant Colony Based Model Selection for Functional-Input Gaussian Process Regression (`https://hal.science/hal-02532713`)

Authors

José Betancourt, François Bachoc and Thierry Klein

Contributors

Déborah Idier and Jérémy Rohmer

Note

This package was first developed in the frame of the RISCOPE research project, funded by the French Agence Nationale de la Recherche (ANR) for the period 2017-2021 (ANR, project No. 16CE04-0011, RISCOPE.fr), and certified by SAFE Cluster.

Author(s)

Maintainer: Jose Betancourt <fungp.rpack@gmail.com>

Authors:

- François Bachoc <francois.bachoc@math.univ-toulouse.fr>
- Thierry Klein <thierry.klein@math.univ-toulouse.fr>
- Jeremy Rohmer <J.Rohmer@brgm.fr>

Other contributors:

- Yves Deville <deville.yves@alpestat.com> [contributor]
- Deborah Idier <D.Idier@brgm.fr> [contributor]

See Also

Useful links:

- https://djbetancourt-gh.github.io/funGp/
- https://github.com/djbetancourt-gh/funGp
- Report bugs at https://github.com/djbetancourt-gh/funGp/issues

---

antsLog-class — *S4 class for log of models explored by ant colony in funGp*

Description

Register of model structures and their performance statistics, if available.

Slots

- `sols` Object of class `"data.frame"`. Compendium of model structures arranged by rows. Each column is linked to one structural parameter of the model such as the state of one variable (inactive, active) or the type of kernel function.
- `args` Object of class `"list"`. Compendium of model structures represented by objects of class `"modelCall"`.
- `fitness` Object of class `"numeric"`. Performance statistic of each model, if available.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

---

`black-boxes` *Analytic models for the exploration of the funGp package*

Description

Set of analytic functions that take functional variables as inputs. Since they run quickly, they can be used for testing of **funGp** functionalities as if they were black box computer models. They cover different situations (number of scalar inputs and complexity of the inputs-output mathematical relationship).

Usage

```
fgp_BB1(sIn, fIn, n.tr)

fgp_BB2(sIn, fIn, n.tr)

fgp_BB3(sIn, fIn, n.tr)

fgp_BB4(sIn, fIn, n.tr)

fgp_BB5(sIn, fIn, n.tr)

fgp_BB6(sIn, fIn, n.tr)

fgp_BB7(sIn, fIn, n.tr)
```

Arguments

`sIn`	Object with class `"matrix"`. The scalar input points. Variables are arranged by columns and coordinates by rows.
`fIn`	Object with class `"list"`. The functional inputs. Each element of the list must be a matrix containing the set of curves corresponding to one functional input.
`n.tr`	Object with class `"numeric"`. The number of input points provided and correspondingly, the number of observations to produce.

Details

For all the functions, the d_s scalar inputs x_i are in the real interval $[0,\ 1]$ and the d_f functional inputs $f_i(t_i)$ are defined on the interval $[0,\ 1]$. Expressions for the values are as follows.

- `fgp_BB1` With $d_s = 2$ $d_f = 2$

```
x1 * sin(x2) + x1 * mean(f1) - x2^2 * diff(range(f2))
```

- `fgp_BB2` With $d_s = 2$ and $d_f = 2$

```
x1 * sin(x2) + mean(exp(x1 * t1) * f1) - x2^2 * mean(f2^2 * t2)
```

- `fgp_BB3` With $d_s = 2$ and $d_f = 2$ is the first analytical example in Muehlenstaedt et al (2017)

```
x1 + 2 * x2 + 4 * mean(t1 * f1) + mean(f2)
```

- `fgp_BB4` With $d_s = 2$ and $d_f = 2$ is the second analytical example in *preprint* of Muehlenstaedt et al (2017)

```
(x2 - (5 / (4 * pi^2)) * x1^2 + (5 / pi) * x1 - 6)^2 +
    10 * (1 - (1 / (8 * pi))) * cos(x1) + 10 +
    (4 / 3) * pi * (42 * mean(f1 * (1 - t1)) +
                    pi * ((x1 + 5) / 5) + 15) * mean(t2 * f2))
```

- `fgp_BB5` With $d_s = 2$ and $d_f = 2$ is inspired by the second analytical example in *final version* of Muehlenstaedt et al (2017)

```
(x2 - (5 / (4 * pi^2)) * x1^2 + (5 / pi) * x1 - 6)^2 +
    10 * (1 - (1 / (8 * pi))) * cos(x1) + 10 +
    (4 / 3) * pi * (42 * mean(15 * f1 * (1 - t1) - 5) +
                    pi * ((x1 + 5) / 5) + 15) * mean(15 * t2 * f2))
```

- `fgp_BB6` With $d_s = 2$ and $d_f = 2$ is inspired by the analytical example in Nanty et al (2016)

```
2 * x1^2 + 2 * mean(f1 + t1) + 2 * mean(f2 + t2) + max(f2) + x2
```

- `fgp_BB7` With $d_s = 5$ and $d_f = 2$ is inspired by the second analytical example in *final version* of Muehlenstaedt et al (2017)

```
(x2 + 4 * x3 - (5 / (4 * pi^2)) * x1^2 + (5 / pi) * x1 - 6)^2 +
    10 * (1 - (1 / (8 * pi))) * cos(x1) * x2^2 * x5^3 + 10 +
    (4 / 3) * pi * (42 * sin(x4) * mean(15 * f1 * (1 - t1) - 5) +
                   pi * (((x1 * x5 + 5) / 5) + 15) * mean(15 * t2 * f2))
```

Value

An object of class `"matrix"` with the values of the output at the specified input coordinates.

Note

The functions listed above were used to validate the functionality and stability of this package. Several tests involving all main functions, plotters and getters were run for scalar-input, functional-input and hybrid-input models. In all cases, the output of the functions were correct from the statistical and programmatic perspectives. For an example on the kind of tests performed, the interested user is referred to the introductory funGp manual (doi:10.18637/jss.v109.i05).

References

Muehlenstaedt, T., Fruth, J., and Roustant, O. (2017), "Computer experiments with functional inputs and scalar outputs by a norm-based approach". *Statistics and Computing*, **27**, 1083-1097. [SC]

Nanty, S., Helbert, C., Marrel, A., Pérot, N., and Prieur, C. (2016), "Sampling, metamodeling, and sensitivity analysis of numerical simulators with functional stochastic inputs". *SIAM/ASA Journal on Uncertainty Quantification*, **4**(1), 636-659. doi:10.1137/15M1033319

`decay` *Decay functions for ant colony optimization in funGp*

Description

This function is intended to aid the selection of the heuristic parameters *tao0*, *delta* and *dispr* in the call to the model selection function fgpm_factory. The values computed by decay are the ones that would be used by the ant colony algorithm as initial pheromone load of the links pointing out to projection on each dimension. For more details, check the technical report explaining the ant colony algorithm implemented in funGp, and the manual of the package (doi:10.18637/jss.v109.i05).

Usage

```
decay(
  k,
  pmax = NULL,
  tao0 = 0.1,
  delta = 2,
  dispr = 1.4,
  doplot = TRUE,
  deliver = FALSE
)
```

Arguments

`k`	A number indicating the dimension of the functional input under analysis.
`pmax`	An optional number specifying the hypothetical maximum projection dimension of this input. The user will be able to set this value later in the call to fgpm_factory as a constraint. If not specified, it takes the value of k.
`tao0`	Explained in the description of *dispr*.
`delta`	Explained in the description of *dispr*.
`dispr`	The arguments *tao0*, *delta* and *dispr*, are optional numbers specifying the loss function that determines the initial pheromone load on the links pointing out to projection dimensions. Such a function is defined as $$tao = tao0 * exp(-.5 * ((p - delta - 1)^2/(-dispr^2/(2 * log(.5))),$$ with p taking the values of the projection dimensions. The argument *tao0* indicates the pheromone load in the links pointing out to the smallest dimensions;

delta specifies how many dimensions should preserve the maximum pheromone load; *dispr* determines how fast the pheromone load drops in dimensions further than $delta + 1$. If *pmax* = *k*, then the dimension 0, representing no projection, receives a pheromone load identical to that of dimension *k*. This, in order to represent the fact that both the representation of the function in its original dimension or a projection in a space of the same dimension, are equally heavy for the model. The default values of *tao0*, *delta* and *dispr*, are 0.1, 2 and 1.4, respectively, which match the default values used by the fgpm_factory function. Check this technical report for more details.

`doplot`	An optional boolean indicating if the pheromone loads should be plotted. Default = TRUE.
`deliver`	An optional boolean indicating if the pheromone loads should be returned. Default = FALSE.

Value

If deliver is TRUE, an object of class `"numeric"` containing the initial pheromone values corresponding to the specified projection dimensions. Otherwise, the function plots the pheromones and nothing is returned.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

See Also

* decay2probs for the function to generate the initial probability load;

* fgpm_factory for heuristic funGp model selection.

Examples

```
# using default decay arguments___________________________________________________________
# input of dimension 15 projected maximum in dimension 15
decay(15)

# input of dimension 15 projected maximum in dimension 8
decay(15, 8)


# playing with decay arguments____________________________________________________________
# input of dimension 15 projected maximum in dimension 15
decay(15)

# using a larger value of tao0
decay(15, tao0 = .3)

# using a larger value of tao0, keeping it fixed up to higher dimensions
decay(15, tao0 = .3, delta = 5)

# using a larger value of tao0, keeping it fixed up to higher dimensions, with slower decay
```

```
decay(15, tao0 = .3, delta = 5, dispr = 5.2)


# requesting pheromone values_________________________________________________________________
# input of dimension 15 projected maximum in dimension 15
decay(15, deliver = TRUE)
```

`decay2probs` *Probability functions for ant colony optimization in funGp*

Description

This function is intended to aid the selection of the heuristic parameters *tao0*, *delta* and *dispr* in the call to the model selection function fgpm_factory. The values computed by decay2probs are the ones that would be used by the ant colony algorithm as probability load of the links pointing out to projection on each dimension. These values result from the normalization of the initial pheromone loads delivered by the decay function, which are made to sum 1. For more details, check the technical report explaining the ant colony algorithm implemented in funGp, and the manual of the package (doi:10.18637/jss.v109.i05).

Usage

```
decay2probs(
  k,
  pmax = NULL,
  tao0 = 0.1,
  delta = 2,
  dispr = 1.4,
  doplot = TRUE,
  deliver = FALSE
)
```

Arguments

`k`	A number indicating the dimension of the functional input under analysis.
`pmax`	An optional number specifying the hypothetical maximum projection dimension of this input. The user will be able to set this value later in the call to fgpm_factory as a constraint. If not specified, it takes the value of k.
`tao0`	Explained in the description of *dispr*.
`delta`	Explained in the description of *dispr*.
`dispr`	The arguments *tao0*, *delta* and *dispr*, are optional numbers specifying the loss function that determines the initial pheromone load on the links pointing out to projection dimensions. Such a function is defined as

$$tao = tao0 * exp(-.5 * ((p - delta - 1)^2/(-dispr^2/(2 * log(.5))),$$

with p taking the values of the projection dimensions. The argument *tao0* indicates the pheromone load in the links pointing out to the smallest dimensions; *delta* specifies how many dimensions should preserve the maximum pheromone load; *dispr* determines how fast the pheromone load drops in dimensions further than $delta + 1$. If *pmax* = *k*, then the dimension 0, representing no projection, receives a pheromone load identical to that of dimension *k*. This, in order to represent the fact that both the representation of the function in its original dimension or a projection in a space of the same dimension, are equally heavy for the model. In order to obtain the probability loads, the initial pheromone values are normalized to sum 1. Note that the normalization makes the value of tao0 become irrelevant in the initial probability load. This does not mean that the effect of tao0 is completely removed from the algorithm. Despite the fact that tao0 does not have influence on the selection of the projection dimension during the first iteration, it will be protagonist during the global pheromone update and will have an impact on every further iteration. The argument tao0 is left active in the input just for a better comprehension of the functioning of the mechanisms defining the initial pheromone and probability loads. The default values of *tao0*, *delta* and *dispr*, are 0.1, 2 and 1.4, respectively, which match the default values used by the fgpm_factory function. Check this technical report for more details.

`doplot` An optional boolean indicating if the probability loads should be plotted. Default = TRUE.

`deliver` An optional boolean indicating if the probability loads should be returned. Default = FALSE.

Value

If deliver is TRUE, an object of class `"numeric"` containing the normalized initial pheromone values corresponding to the specified projection dimensions. Otherwise, the function plots the normalized pheromones and nothing is returned.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

See Also

* decay for the function to generate the initial pheromone load;

* fgpm_factory for heuristic model selection in funGp.

Examples

```
# using default decay arguments_____________________________________________________________
# input of dimension 15 projected maximum in dimension 15
decay(15) # initial pheromone load
decay2probs(15) # initial probability load

# input of dimension 15 projected maximum in dimension 8
decay(15, 8) # initial pheromone load
decay2probs(15, 8) # initial probability load
```

```
# playing with decay2probs arguments_________________________________________________________
# varying the initial pheromone load
decay(15) # input of dimension 15 projected maximum in dimension 15
decay(15, tao0 = .3) # larger value of tao0
decay(15, tao0 = .3, delta = 5) # larger tao0 kept to higher dimensions
decay(15, tao0 = .3, delta = 5, dispr = 5.2) # larger tao0 kept to higher dimensions
                                             # and slower decay

# varying the initial probability load
decay2probs(15) # input of dimension 15 projected maximum in dimension 15
decay2probs(15, tao0 = .3) # larger value of tao0 (no effect whatsoever)
decay2probs(15, tao0 = .3, delta = 5) # larger tao0 kept to higher dimensions
decay2probs(15, tao0 = .3, delta = 5, dispr = 5.2) # larger tao0 kept to higher dimensions
                                                   # and slower decay


# requesting probability values______________________________________________________________
# input of dimension 15 projected maximum in dimension 15
decay2probs(15, deliver = TRUE)
```

`factoryCall-class` *S4 class for fgpm_factory function calls*

Description

User reminder of the fgpm function call.

Slots

`string` Object of class `"character"`. User call reminder in string format.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

---

`fgpKern-class` *S4 class for structures linked to the kernel of a* `fgpm` *model*

Description

This is the formal representation for data structures linked to the kernel of a Gaussian process model within the funGp package.

Slots

`kerType` Object of class "character". Kernel type. To be set from {"gauss", "matern5_2", "matern3_2"}.

`f_disType` Object of class "character". Distance type. To be set from {"L2_bygroup", "L2_index"}.

`varHyp` Object of class "numeric". Estimated variance parameter.

`s_lsHyps` Object of class "numeric". Estimated length-scale parameters for scalar inputs.

`f_lsHyps` Object of class "numeric". Estimated length-scale parameters for functional inputs.

`f_lsOwners` Object of class "character". Index of functional input variable linked to each element in *f_lsHyps*

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

---

`fgpm` *Gaussian process models for scalar and functional inputs*

Description

This function enables fitting of Gaussian process regression models. The inputs can be either scalar, functional or a combination of both types.

Usage

```
fgpm(
  sIn = NULL,
  fIn = NULL,
  sOut,
  kerType = "matern5_2",
  f_disType = "L2_bygroup",
  f_pdims = 3,
  f_basType = "B-splines",
  var.hyp = NULL,
  ls_s.hyp = NULL,
  ls_f.hyp = NULL,
  nugget = 1e-08,
  n.starts = 1,
  n.presample = 20,
  par.clust = NULL,
  trace = TRUE,
  pbars = TRUE,
  control.optim = list(trace = TRUE),
  ...
)
```

Arguments

`sIn`	An optional matrix of scalar input values to train the model. Each column must match an input variable and each row a training point. Either scalar input coordinates (sIn), functional input coordinates (fIn), or both must be provided.
`fIn`	An optional list of functional input values to train the model. Each element of the list must be a matrix containing the set of curves corresponding to one functional input. Either scalar input coordinates (sIn), functional input coordinates (fIn), or both must be provided.
`sOut`	A vector (or 1-column matrix) containing the values of the scalar output at the specified input points.
`kerType`	An optional character string specifying the covariance structure to be used. To be chosen between "gauss", "matern5_2" and "matern3_2". Default is "matern5_2".
`f_disType`	An optional array of character strings specifying the distance function to be used for each functional coordinates within the covariance function of the Gaussian process. To be chosen between "L2_bygroup" and "L2_byindex". The L2_bygroup distance considers each curve as a whole and uses a single length-scale parameter per functional input variable. The L2_byindex distance uses as many length-scale parameters per functional input as discretization points it has. For instance an input discretized as a vector of size 8 will use 8 length-scale parameters when using L2_byindex. If dimension reduction of a functional input is requested, then L2_byindex uses as many length scale parameters as effective dimensions used to represent the input. A single character string can also be passed as a general selection for all the functional inputs of the model. More details in the reference article (doi:10.1016/j.ress.2020.106870) and the in-depth package manual (doi:10.18637/jss.v109.i05). Default is "L2_bygroup".
`f_pdims`	An optional array with the projection dimension for each functional input. For each input, the projection dimension should be an integer between 0 and its original dimension, with 0 denoting no projection. A single character string can also be passed as a general selection for all the functional inputs of the model. Default is 3.
`f_basType`	An optional array of character strings specifying the family of basis functions to be used in the projection of each functional input. To be chosen between "B-splines" and "PCA". A single character string can also be passed as a general selection for all the functional inputs of the model. This argument will be ignored for those inputs for which no projection was requested (i.e., for which f_pdims = 0). Default is "B-splines".
`var.hyp`	An optional number indicating the value that should be used as the variance parameter of the model. If not provided, it is estimated through likelihood maximization.
`ls_s.hyp`	An optional numeric array indicating the values that should be used as length-scale parameters for the scalar inputs. If provided, the size of the array should match the number of scalar inputs. If not provided, these parameters are estimated through likelihood maximization.
`ls_f.hyp`	An optional numeric array indicating the values that should be used as length-scale parameters for the functional inputs. If provided, the size of the array

should match the number of effective dimensions. Each input using the "L2_bygroup" distance will count 1 effective dimension, and each input using the "L2_byindex" distance will count as many effective dimensions as specified by the corresponding element of the f_pdims argument. For instance, two functional inputs of original dimensions 10 and 22, the first one projected onto a space of dimension 5 with "L2_byindex" distance, and the second one not projected with "L2_bygroup" distance will make up a total of 6 effective dimensions; five for the first functional input and one for second one. If this argument is not provided, the functional length-scale parameters are estimated through likelihood maximization.

`nugget` An optional variance value standing for the homogeneous nugget effect. A tiny nugget might help to overcome numerical problems related to the ill-conditioning of the covariance matrix. Default is 1e-8.

`n.starts` An optional integer indicating the number of initial points to use for the optimization of the hyperparameters. A parallel processing cluster can be exploited in order to speed up the evaluation of multiple initial points. More details in the description of the argument par.clust below. Default is 1.

`n.presample` An optional integer indicating the number of points to be tested in order to select the n.starts initial points. The n.presample points will be randomly sampled from the hyper-rectangle defined by:

1e-10 $\leq$ `ls_s.hyp[i]` $\leq$ 2*max(sMs[[i]]), for i in 1 to the number of scalar inputs,
1e-10 $\leq$ `ls_f.hyp[i]` $\leq$ 2*max(fMs[[i]]), for i in 1 to the number of functional inputs,

with sMs and fMs the lists of distance matrices for the scalar and functional inputs, respectively. The value of n.starts will be assigned to n.presample if this last is smaller. Default is 20.

`par.clust` An optional parallel processing cluster created with the `makeCluster` function of the parallel package. If not provided, multistart optimizations are done in sequence.

`trace` An optional boolean indicating if control messages native of the funGp package should be printed to console. Default is TRUE. For complementary control on the display of funGp-native progress bars and `optim` trace about the hyperparameter optimization process, have a look at the `pbars` and `control.optim` arguments, respectively.

`pbars` An optional boolean indicating if progress bars should be displayed. Default is TRUE.

`control.optim` An optional list to be passed as the `control` argument to `optim`, the function in charge of the non-linear optimization of the hyperparameters. Default is `list(trace = TRUE)`, equivalent to `list(trace = 1)`, which enables the printing of tracing information on the progress of the optimization. Before interacting with the `fgpm()` `control.optim` argument, please carefully check the documentation about the `control` argument provided in `optim` to ensure a coherent behavior and sound results. Note that: (i) at this time, only the `"L-BFGS-B"` method (Byrd et. al., 1995) is enabled in `fgpm()`; (ii) `control.optim$fnscale`

	should not be used since our optimization problem is strictly of minimization, not maximization.
`...`	Extra control parameters. Currently only used internally for some `update()` calls.

Value

An object of class fgpm containing the data structures representing the fitted funGp model.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

References

Betancourt, J., Bachoc, F., Klein, T., Idier, D., Rohmer, J., and Deville, Y. (2024), "funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs". *Journal of Statistical Software*, **109**, 5, 1–51. (doi:10.18637/jss.v109.i05)

Betancourt, J., Bachoc, F., Klein, T., Idier, D., Pedreros, R., and Rohmer, J. (2020), "Gaussian process metamodeling of functional-input code for coastal flood hazard assessment". *Reliability Engineering & System Safety*, **198**, 106870. (doi:10.1016/j.ress.2020.106870) [HAL]

Betancourt, J., Bachoc, F., Klein, T., and Gamboa, F. (2020), Technical Report: "Ant Colony Based Model Selection for Functional-Input Gaussian Process Regression. Ref. D3.b (WP3.2)". *RISCOPE project*. [HAL]

Betancourt, J., Bachoc, F., and Klein, T. (2020), R Package Manual: "Gaussian Process Regression for Scalar and Functional Inputs with funGp - The in-depth tour". *RISCOPE project*. [HAL]

See Also

* plot,fgpm-method: validation plot for a `fgpm` model;
* predict,fgpm-method for predictions based on a `fgpm` model;
* simulate,fgpm-method for simulations based on a `fgpm` model;
* update,fgpm-method for post-creation updates on a `fgpm` model;
* fgpm_factory for funGp heuristic model selection.

Examples

```
# creating funGp model using default fgpm arguments________________________________________
# generating input data for training
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))

# generating output data for training
sOut <- fgp_BB3(sIn, fIn, n.tr)

# building a scalar-input funGp model
```

```
ms <- fgpm(sIn = sIn, sOut = sOut)

# building a functional-input funGp model
mf <- fgpm(fIn = fIn, sOut = sOut)

# building a hybrid-input funGp model
msf <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# plotting the three models
plot(ms)
plot(mf)
plot(msf)

# printing the three models
summary(ms) # equivalent to show(ms)
summary(mf) # equivalent to show(mf)
summary(msf) # equivalent to show(msf)


# recovering useful information from a funGp model_________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# recovering data from model slots
m1@f_proj@coefs # list of projection coefficients for the functional inputs
m1@f_proj@basis # list of projection basis functions for the functional inputs
Map(function(a, b) a %*% t(b), m1@f_proj@coefs, m1@f_proj@basis) # list of projected
                                                                  # functional inputs
tcrossprod(m1@preMats$L) # training auto-covariance matrix


# making predictions based on a funGp model________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# generating input data for prediction
n.pr <- 100
sIn.pr <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.pr)),
                                x2 = seq(0,1,length = sqrt(n.pr))))
fIn.pr <- list(f1 = matrix(runif(n.pr*10), ncol = 10), matrix(runif(n.pr*22), ncol = 22))

# making predictions
m1.preds <- predict(m1, sIn.pr = sIn.pr, fIn.pr = fIn.pr)
```

```
# plotting predictions
plot(m1.preds)


# simulating from a funGp model_____________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# generating input data for simulation
n.sm <- 100
sIn.sm <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.sm)),
                                x2 = seq(0,1,length = sqrt(n.sm))))
fIn.sm <- list(f1 = matrix(runif(n.sm*10), ncol = 10), matrix(runif(n.sm*22), ncol = 22))

# making simulations
m1.sims <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm)

# plotting simulations
plot(m1.sims)


# creating funGp model using custom fgpm arguments________________________________________
# generating input and output data
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)

# original dimensions
# f1: 10
# f2: 22

# building a the model with the following structure
#    - Kernel: Gaussian
#    - f1: L2_byindex distance, no projection -> 10 length-scale parameters
#    - f2: L2_bygroup distance, B-spline basis of dimension 5 -> 1 length-scale parameter
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut,
           kerType = "gauss", f_disType = c("L2_byindex", "L2_bygroup"),
           f_pdims = c(0,5), f_basType = c(NA, "B-splines"))

# plotting the model
plot(m1)

# printing the model
m1 # equivalent to show(m1)
```

```
## Not run:
# multistart and parallelization in fgpm____________________________________________________
# generating input and output data
set.seed(100)
n.tr <- 243
sIn <- expand.grid(x1 = seq(0,1,length = n.tr^(1/5)), x2 = seq(0,1,length = n.tr^(1/5)),
                   x3 = seq(0,1,length = n.tr^(1/5)), x4 = seq(0,1,length = n.tr^(1/5)),
                   x5 = seq(0,1,length = n.tr^(1/5)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# calling fgpm with multistart in parallel
cl <- parallel::makeCluster(2)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut, n.starts = 10, par.clust = cl) # (~14 seconds)
parallel::stopCluster(cl)

# NOTE: in order to provide progress bars for the monitoring of time consuming processes
#      ran in parallel, funGp relies on the doFuture and future packages. Parallel processes
#       suddenly interrupted by the user tend to leave corrupt connections. This problem is
#      originated outside funGp, which limits our control over it. In the initial (unpublished)
#      version of the funGp manual, we provide a temporary solution to the issue and we remain
#      attentive in case it appears a more elegant way to handle it or a manner to suppress it.
#
#        funGp original (unpublished) manual: https://hal.science/hal-02536624

## End(Not run)
```

---

fgpm-class *S4 class for funGp Gaussian process models*

Description

This is the formal representation of Gaussian process models within the funGp package. Gaussian process models are useful statistical tools in the modeling of complex input-output relationships.

- **Main methods**
  fgpm: creation of funGp regression models
  predict,fgpm-method: output estimation at new input points based on a `fgpm` model
  simulate,fgpm-method: random sampling from a `fgpm` model
  update,fgpm-method: modification of data and hyperparameters of a `fgpm` model
- **Plotters**
  plot,fgpm-method: validation plot for a `fgpm` model
  plot.predict.fgpm: plot of predictions based on a `fgpm` model
  plot.simulate.fgpm: plot of simulations based on a `fgpm` model

Slots

`howCalled` Object of class "`modelCall`". User call reminder.

`type` Object of class "character". Type of model based on type of inputs. To be set from {"scalar", "functional", "hybrid"}.

`ds` Object of class "numeric". Number of scalar inputs.

`df` Object of class "numeric". Number of functional inputs.

`f_dims` Object of class "numeric". An array with the original dimension of each functional input.

`sIn` Object of class "matrix". The scalar input points. Variables are arranged by columns and coordinates by rows.

`fIn` Object of class "list". The functional input points. Each element of the list contains a functional input in the form of a matrix. In each matrix, curves representing functional coordinates are arranged by rows.

`sOut` Object of class "matrix". The scalar output values at the coordinates specified by sIn and/or fIn.

`n.tot` Object of class "integer". Number of observed points used to compute the training-training and training-prediction covariance matrices.

`n.tr` Object of class "integer". Among all the points loaded in the model, the amount used for training.

`f_proj` Object of class "fgpProj". Data structures related to the projection of functional inputs. Check fgpProj for more details.

`kern` Object of class "fgpKern". Data structures related to the kernel of the Gaussian process model. Check fgpKern for more details.

`nugget` Object of class "numeric". Variance parameter standing for the homogeneous nugget effect.

`preMats` Object of class "list". L and LInvY matrices pre-computed for prediction. L is a lower diagonal matrix such that $L'L$ equals the training auto-covariance matrix $K.tt$. On the other hand, $LInvY = L^(-1) * sOut$.

`convergence` Object of class "numeric". Integer code either confirming convergence or indicating an error. Check the convergence component of the Value returned by `optim`.

`negLogLik` Object of class "numeric". Negated log-likelihood obained by `optim` during hyperparameter optimization.

Useful material

- **Manual:** funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs (doi:10.18637/jss.v109.i05)

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer fgpm_factory *Structural optimization of Gaussian process models*

Description

This function enables the smart exploration of the solution space of potential structural configurations of a funGp model, and the consequent selection of a high quality configuration. funGp currently relies on an ant colony based algorithm to perform this task. The algorithm defines the solution space based on the levels of each structural parameter currently available in the fgpm function, and performs a smart exploration of it. More details on the algorithm are provided in a dedicated technical report. funGp might evolve in the future to include improvements in the current algorithm or alternative solution methods.

Usage

```
fgpm_factory(
  sIn = NULL,
  fIn = NULL,
  sOut = NULL,
  ind.vl = NULL,
  ctraints = list(),
  setup = list(),
  time.lim = Inf,
  nugget = 1e-08,
  n.starts = 1,
  n.presample = 20,
  par.clust = NULL,
  trace = TRUE,
  pbars = interactive()
)
```

Arguments

`sIn` An optional matrix of scalar input values to train the model. Each column must match an input variable and each row a training point. Either scalar input coordinates (sIn), functional input coordinates (fIn), or both must be provided.

`fIn` An optional list of functional input values to train the model. Each element of the list must be a matrix containing the set of curves corresponding to one functional input. Either scalar input coordinates (sIn), functional input coordinates (fIn), or both must be provided.

`sOut` A vector (or 1-column matrix) containing the values of the scalar output at the specified input points.

`ind.vl` An optional numerical matrix specifying which points in the three structures above should be used for training and which for validation. If provided, the optimization will be conducted in terms of the hold-out coefficient of determination Q^2, which comes from training the model with a subset of the points, and then estimating the prediction error in the remaining points. In that case, each column of *ind.vl* will be interpreted as one validation set, and the multiple columns will imply replicates. In the simplest case, *ind.vl* will be a one-column matrix or simply an array, meaning that a simple replicate should be used for each model configuration explored. If not provided, the optimization will be conducted in terms of the leave-one-out cross-validation Q^2, which for a total number of n observations, comes from training the model n times, each using n-1 points for training and the remaining one for validation. This procedure is typically costly due to the large number of hyperparameter optimizations that should be conducted, nonetheless, fgpm_factory implements the virtual equations introduced by Dubrule (1983) for Gaussian processes, which require a single hyperparameter optimization. See the reference below for more details.

`ctraints` An optional list specifying the constraints of the structural optimization problem. Valid entries for this list are:

*****_s_keepOn_: a numerical array indicating the scalar inputs that should remain active in the model. It should contain the indices of the columns of sIn corresponding to the inputs to keep active.

*****_f_keepOn_: a numerical array indicating the functional inputs that should remain active in the model. It should contain the indices of the elements of fIn corresponding to the inputs to keep active.

*****_f_disTypes_: a list specifying the set of distances that should be tested for some functional inputs. The values should be taken from the possibilities offered by the fgpm function for the argument *f_disType* therein. Valid choices at this time are "L2_bygroup" and "L2_byindex". Each element of the list should receive as name the index of a functional input variable, and should contain an array of strings with the name of the distances allowed for this input. All the available distances will be tried for any functional input not included in the list.

*****_f_fixDims_: a two-row matrix specifying a particular projection dimension for some functional inputs. For each input, the value should be a number between 0 and its original dimension, with 0 denoting no projection. The first row of the matrix should contain the index of each input, and the second row should contain the corresponding dimensions. All the possible dimensions will be tried for any functional input not included in the matrix (unless affected by the *f_maxDims* argument below).

*****_f_maxDims_: a two-row matrix specifying the largest projection dimension for some functional inputs. For each input, the value should be a number between 1 and its original dimension. The first row of the matrix should contain the index of each input, and the second row should contain the corresponding largest dimensions. All the possible dimensions will be tried for any functional input not included in the matrix (unless affected by the *f_fixDims* argument above).

*****_f_basTypes_: a list specifying the set of basis families that should be tested for some functional inputs. The values should be taken from the possibilities offered by the fgpm function for the argument *f_basType* therein. Valid choices at this time are "B-splines" and "PCA". Each element of the list should receive as name the index of a functional input variable, and should contain an array of strings with the name of the distances allowed for this input. All the available basis families will be tried for any functional input not included in the list.

******kerTypes*: an array of strings specifying the kernel functions allowed to be tested. The values should be taken from the possibilities offered by the fgpm function for the argument *kerType* therein. Valid choices at this time are "gauss", "matern5_2" and "matern3_2". If not provided, all the available kernel functions will be tried.

`setup` An optional list indicating the value for some parameters of the structural optimization algorithm. The ant colony optimization algorithm available at this time allows the following entries:

Initial pheromone load

******tao0*: a number indicating the initial pheromone load on links pointing out to the selection of a distance type, a projection basis or a kernel type. Default is 0.1.

******dop.s*: a number controlling how likely it is to activate a scalar input. It operates on a relation of the type $A = dop.s * I$, where A is the initial pheromone load of links pointing out to the activation of scalar inputs and I is the initial pheromone load of links pointing out to their inactivation. Default is 1.

******dop.f*: analogous to *dop.s* for functional inputs. Default is 1.

******delta.f and dispr.f*: two numbers used as shape parameters for the regularization function that determines the initial pheromone values on the links connecting the L2_byindex distance with the projection dimension. Default are 2 and 1.4, respectively.

Local pheromone update

******rho.l*: a number specifying the pheromone evaporation rate. Default is 0.1.

Global pheromone update

******u.gbest*: a boolean indicating if at each iteration, the pheromone load on the links of the best ant of the whole trial should be reinforced. Default is FALSE.

******n.ibest*: a number indicating how many top ants of each iteration should be used for pheromone reinforcement. Default is 1.

******rho.g*: a number specifying the learning reinforcement rate. Default is 0.1.

Population factors

**n.iter*: a number specifying the amount of iterations of the algorithm. Default is 15.

**n.pop*: a number specifying the amount of ants per iteration; each ant corresponds to one structural configuration for the model. Default is 10.

Bias strength

**q0*: ants use one of two rules to select their next node at each step. The first rule leads the ant through the link with higher pheromone load; the second rule works based on probabilities which are proportional to the pheromone load on the feasible links. The ants will randomly chose one of the two rules at each time. They will opt for rule 1 with probability *q0*. Default is 0.95.

`time.lim` An optional number specifying a time limit in seconds to be used as stopping condition for the structural optimization.

`nugget` An optional variance value standing for the homogeneous nugget effect. A tiny nugget might help to overcome numerical problems related to the ill-conditioning of the covariance matrix. Default is 1e-8.

`n.starts` An optional integer indicating the number of initial points to use for the optimization of the hyperparameters. A parallel processing cluster can be exploited in order to speed up the evaluation of multiple initial points. More details in the description of the argument par.clust below. Default is 1.

`n.presample` An optional integer indicating the number of points to be tested in order to select the n.starts initial points. The n.presample points will be randomly sampled from the hyper-rectangle defined by:

1e-10 $\leq$ `ls_s.hyp[i]` $\leq$ 2*max(`sMs[[i]]`), for i in 1 to the number of scalar inputs,
1e-10 $\leq$ `ls_f.hyp[i]` $\leq$ 2*max(`fMs[[i]]`), for i in 1 to the number of functional inputs,

with sMs and fMs the lists of distance matrices for the scalar and functional inputs, respectively. The value of n.starts will be assigned to n.presample if this last is smaller. Default is 20.

`par.clust` An optional parallel processing cluster created with the `makeCluster` function of the parallel package. If not provided, structural configurations are evaluated in sequence.

`trace` An optional boolean indicating if control messages native of the funGp package should be printed to console. Default is TRUE. For complementary control on the display of funGp-native progress bars, have a look at the `pbars` argument below.

`pbars` An optional boolean indicating if progress bars should be displayed. Default is TRUE.

Value

An object of class Xfgpm containing the data structures linked to the structural optimization of a funGp model. It includes as the main component an object of class fgpm corresponding to the optimized model. It is accessible through the `@model` slot of the Xfgpm object.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

References

Betancourt, J., Bachoc, F., Klein, T., Idier, D., Rohmer, J., and Deville, Y. (2024), "funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs". *Journal of Statistical Software*, **109**, 5, 1–51. (doi:10.18637/jss.v109.i05)

Betancourt, J., Bachoc, F., Klein, T., Idier, D., Pedreros, R., and Rohmer, J. (2020), "Gaussian process metamodeling of functional-input code for coastal flood hazard assessment". *Reliability Engineering & System Safety*, **198**, 106870. (doi:10.1016/j.ress.2020.106870) [HAL]

Betancourt, J., Bachoc, F., Klein, T., and Gamboa, F. (2020), Technical Report: "Ant Colony Based Model Selection for Functional-Input Gaussian Process Regression. Ref. D3.b (WP3.2)". *RISCOPE project*. [HAL]

Betancourt, J., Bachoc, F., and Klein, T. (2020), R Package Manual: "Gaussian Process Regression for Scalar and Functional Inputs with funGp - The in-depth tour". *RISCOPE project*. [HAL]

Dubrule, O. (1983), "Cross validation of kriging in a unique neighborhood". *Journal of the International Association for Mathematical Geology*, **15**, 687-699. [MG]

See Also

* plot,Xfgpm-method with `which = "evolution"` for visualizing the evolution of the ACO algorithm, or with `which = "diag"` for a diagnostic plot;

* get_active_in for post-processing of input data structures following a fgpm_factory call;

* predict,fgpm-method for predictions based on a funGp model;

* simulate,fgpm-method for simulations based on a funGp model;

* update,fgpm-method for post-creation updates on a funGp model.

Examples

```
#construction of a fgpm object
set.seed(100)
n.tr <- 32
x1 <- x2 <- x3 <- x4 <- x5 <- seq(0,1,length = n.tr^(1/5))
sIn <- expand.grid(x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = x5)
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
            f2 = matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# optimizing the model structure with fgpm_factory (~12 seconds)
## Not run:
```

```
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut)

## End(Not run)

# assessing the quality of the model
# in the absolute and also w.r.t. the other explored models
plot(xm, which = "diag")

# checking the evolution of the algorithm
plot(xm, which = "evol")

# Summary of the tested configurations
summary(xm)

# checking the log of crashed iterations
print(xm@log.crashes)

# building the model with the default fgpm arguments to compare
set.seed(100)
n.tr <- 32
x1 <- x2 <- x3 <- x4 <- x5 <- seq(0,1,length = n.tr^(1/5))
sIn <- expand.grid(x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = x5)
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
f2 <- matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB7(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)
plot(m1) # plotting the model

# improving performance with more iterations_______________________________________________
# call to fgpm_factory (~22 seconds)
## Not run:
xm25 <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut,
                     setup = list(n.iter = 25))

## End(Not run)

# assessing evolution and quality
plot(xm25, which = "evol")
plot(xm25, which = "diag")

# custom solution space____________________________________________________________________
myctr <- list(s_keepOn = c(1,2), # keep both scalar inputs always on
f_keepOn = c(2), # keep f2 always active
f_disTypes = list("2" = c("L2_byindex")), # only use L2_byindex distance for f2
f_fixDims = matrix(c(2,4), ncol = 1), # f2 projected in dimension 4
f_maxDims = matrix(c(1,5), ncol = 1), # f1 projected in dimension max 5
f_basTypes = list("1" = c("B-splines")), # only use B-splines projection for f1
kerTypes = c("matern5_2", "gauss")) # test only Matern 5/2 and Gaussian kernels
#
# call to fgpm_factory (~12 seconds)
## Not run:
xmc <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, ctraints = myctr)
```

```
## End(Not run)

# assessing evolution and quality
plot(xmc, which = "evol")
plot(xmc, which = "diag")

# verifying constraints with the log of some successfully built models
summary(xmc)

# custom heuristic parameters___________________________________________________________________
mysup <- list(n.iter = 30, n.pop = 12, tao0 = .15, dop.s = 1.2,
              dop.f = 1.3, delta.f = 4, dispr.f = 1.1, q0 = .85,
              rho.l = .2, u.gbest = TRUE, n.ibest = 2, rho.g = .08)

# call to fgpm_factory (~20 seconds)
## Not run:
xmh <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, setup = mysup)

## End(Not run)

# verifying heuristic setup through the details of the Xfgpm object
unlist(xmh@details$param)

# stopping condition based on time______________________________________________________________
mysup <- list(n.iter = 2000)
mytlim <- 60

# call to fgpm_factory (~60 seconds)
## Not run:
xms <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut,
                    setup = mysup, time.lim = mytlim)

## End(Not run)
summary(xms)

## Not run:
# parallelization in the model factory__________________________________________________________
# generating input and output data
set.seed(100)
n.tr <- 243
sIn <- expand.grid(x1 = seq(0,1,length = n.tr^(1/5)), x2 = seq(0,1,length = n.tr^(1/5)),
                   x3 = seq(0,1,length = n.tr^(1/5)), x4 = seq(0,1,length = n.tr^(1/5)),
                   x5 = seq(0,1,length = n.tr^(1/5)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# calling fgpm_factory in parallel
cl <- parallel::makeCluster(2)
xm.par <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, par.clust = cl) #  (~260 seconds)
parallel::stopCluster(cl)

# NOTE: in order to provide progress bars for the monitoring of time consuming processes
#     ran in parallel, funGp relies on the doFuture and future packages. Parallel processes
```

```
#      suddenly interrupted by the user tend to leave corrupt connections. This problem is
#     originated outside funGp, which limits our control over it. In the initial (unpublished)
#     version of the funGp manual, we provide a temporary solution to the issue and we remain
#     attentive in case it appears a more elegant way to handle it or a manner to suppress it.
#
#       funGp original (unpublished) manual: https://hal.science/hal-02536624

## End(Not run)
```

`fgpProj-class` *S4 class for structures linked to projections in a* `fgpm` *model*

Description

This is the formal representation for data structures linked to projection of inputs in a Gaussian process model within the funGp package.

Slots

`pdims` Object of class `"numeric"`. Projection dimension of each input.

`basType` Object of class `"character"`. To be chosen from {"PCA", "B-splines"}.

`basis` Object of class `"list"`. Projection basis. For functional inputs, each element (fDims_i x fpDims_i) contains the basis functions used for the projection of one functional input.

`coefs` Object of class `"list"`. Each element (n x fpDims_i) contains the coefficients used for the projection of one functional input.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

---

`get_active_in` *Extraction of active inputs in a given model structure*

Description

The fgpm_factory function returns an object of class `"Xfgpm"` with the function call of all the evaluated models stored in the `@log.success@args` and `@log.crashes@args` slots. The `get_active_in` function interprets the arguments linked to any structural configuration and returns a list with two elements: (i) a `matrix` of scalar input variables kept active; and (ii) a `list` of functional input variables kept active.

Usage

```
get_active_in(sIn = NULL, fIn = NULL, args)
```

Arguments

sIn	An optional matrix of scalar input coordinates with all the orignal scalar input variables.
fIn	An optional list of functional input coordinates with all the original functional input variables.
args	An object of class "modelCall", which specifies the model structure for which the active inputs should be extracted.

Value

An object of class "list", containing the following information extracted from the *args* parameter: (i) a matrix of scalar input variables kept active; and (ii) a list of functional input variables kept active.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

References

Betancourt, J., Bachoc, F., Klein, T., Idier, D., Rohmer, J., and Deville, Y. (2024), "funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs". *Journal of Statistical Software*, **109**, 5, 1–51. (doi:10.18637/jss.v109.i05)

Betancourt, J., Bachoc, F., and Klein, T. (2020), R Package Manual: "Gaussian Process Regression for Scalar and Functional Inputs with funGp - The in-depth tour". *RISCOPE project*. [HAL]

See Also

* which_on for details on how to obtain only the indices of the active inputs.

* modelCall for details on the *args* argument.

* fgpm_factory for funGp heuristic model selection.

* Xfgpm for details on object delivered by fgpm_factory.

Examples

```
# Use precalculated Xfgpm object named xm
# indices of active inputs in the best model
xm@log.success@args[[1]] # the full fgpm call
set.seed(100)
n.tr <- 32
sIn <- expand.grid(x1 = seq(0,1,length = n.tr^(1/5)), x2 = seq(0,1,length = n.tr^(1/5)),
x3 = seq(0,1,length = n.tr^(1/5)), x4 = seq(0,1,length = n.tr^(1/5)),
x5 = seq(0,1,length = n.tr^(1/5)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
which_on(sIn, fIn, xm@log.success@args[[1]]) # only the indices extracted by which_on

# data structures of active inputs
active <- get_active_in(sIn, fIn, xm@log.success@args[[1]])
```

```
active$sIn.on # scalar data structures
active$fIn.on # functional data structures
# identifying selected model and corresponding fgpm arguments
opt.model <- xm@model
opt.args <- xm@log.success@args[[1]]

# generating new input data for prediction
n.pr <- 243
sIn.pr <- expand.grid(x1 = seq(0,1,length = n.pr^(1/5)), x2 = seq(0,1,length = n.pr^(1/5)),
                     x3 = seq(0,1,length = n.pr^(1/5)), x4 = seq(0,1,length = n.pr^(1/5)),
                       x5 = seq(0,1,length = n.pr^(1/5)))
fIn.pr <- list(f1 = matrix(runif(n.pr*10), ncol = 10), f2 = matrix(runif(n.pr*22), ncol = 22))

# pruning data structures for prediction to keep only active inputs!!
active <- get_active_in(sIn.pr, fIn.pr, opt.args)

# making predictions
preds <- predict(opt.model, sIn.pr = active$sIn.on, fIn.pr = active$fIn.on)

# plotting predictions
plot(preds)


# preparing new data for simulation based on inputs kept active____________________________
opt.model <- xm@model
opt.args <- xm@log.success@args[[1]]

# generating new input data for simulation
n.sm <- 243
sIn.sm <- expand.grid(x1 = seq(0,1,length = n.pr^(1/5)), x2 = seq(0,1,length = n.pr^(1/5)),
                     x3 = seq(0,1,length = n.pr^(1/5)), x4 = seq(0,1,length = n.pr^(1/5)),
                       x5 = seq(0,1,length = n.pr^(1/5)))
fIn.sm <- list(f1 = matrix(runif(n.sm*10), ncol = 10), f2 = matrix(runif(n.sm*22), ncol = 22))

# pruning data structures for simulation to keep only active inputs!!
active <- get_active_in(sIn.sm, fIn.sm, opt.args)

# making light simulations
sims_l <- simulate(opt.model, nsim = 10, sIn.sm = active$sIn.on, fIn.sm = active$fIn.on)

# plotting light simulations
plot(sims_l)

## Not run:
# rebuilding of 3 best models using new data_______________________________________________
# NOTE: this example is of higher complexity than the previous ones. We recomend you run
#        the previous examples and understand the @log.success and @log.crashes slots in
#        the Xfgpm object delivered by fgpm_factory.
#
#       In the second example above we showed how to use get_active_in to prune the input
#       data structures for prediction based on the fgpm arguments of the best model found
#       by fgpm_factory. In this new example we generalize that concept by: (i) rebuilding
#        the 3 best models found by fgpm_factory using new data, (ii) pruning the input
```

```
#       data structures used for prediction with each of the models, and (iii) plotting
#       the predictions made by the three models. The key ingredient here is that the
#       three best models might have different scalar and functional inputs active. The
#       get_active_in function will allow to process the data structures in order to
#       extract only the scalar inputs required to re-build the model and then to make
#       predictions with each model. Check also the funGp manual for further details
#
#       funGp manual: https://doi.org/10.18637/jss.v109.i05


# <<<<<<< PART 1: calling fgpm_factory to perform the structural optimization >>>>>>>
#         -------------------------------------------------------------------
# this part is precalculated and loaded via data("precalculated_Xfgpm_objects")
summary(xm)

# <<<<<<< PART 2: re-building the three best models found by fgpm_factory >>>>>>>
#         ---------------------------------------------------------------
# recovering the fgpm arguments of the three best models
argStack <- xm@log.success@args[1:3]

# new data arrived, now we have 243 observations
n.nw <- 243 # more points!
sIn.nw <- expand.grid(x1 = seq(0,1,length = n.nw^(1/5)), x2 = seq(0,1,length = n.nw^(1/5)),
                    x3 = seq(0,1,length = n.nw^(1/5)), x4 = seq(0,1,length = n.nw^(1/5)),
                      x5 = seq(0,1,length = n.nw^(1/5)))
fIn.nw <- list(f1 = matrix(runif(n.nw*10), ncol = 10), f2 = matrix(runif(n.nw*22), ncol = 22))
sOut.nw <- fgp_BB7(sIn.nw, fIn.nw, n.nw)

# the second best model
modelDef(xm,2)
# re-building the three best models based on the new data (compact code with all 3 calls)
newEnv <- list(sIn = sIn.nw, fIn = fIn.nw, sOut = sOut.nw)
modStack <- lapply(1:3, function(i) eval(parse(text =  modelDef(xm,i)), env = newEnv))


# <<<<<<< PART 3: making predictions from the three best models found by fgpm_factory >>>>>>>
#         ---------------------------------------------------------------------------
# generating input data for prediction
n.pr <- 32
sIn.pr <- expand.grid(x1 = seq(0,1,length = n.pr^(1/5)), x2 = seq(0,1,length = n.pr^(1/5)),
                    x3 = seq(0,1,length = n.pr^(1/5)), x4 = seq(0,1,length = n.pr^(1/5)),
                      x5 = seq(0,1,length = n.pr^(1/5)))
fIn.pr <- list(f1 = matrix(runif(n.pr*10), ncol = 10), matrix(runif(n.pr*22), ncol = 22))

# making predictions based on the three best models (compact code with all 3 calls)
preds <- do.call(cbind, Map(function(model, args) {
  active <- get_active_in(sIn.pr, fIn.pr, args)
  predict(model, sIn.pr = active$sIn.on, fIn.pr = active$fIn.on)$mean
}, modStack, argStack))


# <<<<<<< PART 4: plotting predictions from the three best models found by fgpm_factory >>>>>>>
#         -----------------------------------------------------------------------------
```

```
# plotting predictions made by the three models
plot(1, xlim = c(1,nrow(preds)), ylim = range(preds), xaxt = "n",
     xlab = "Prediction point index", ylab = "Output",
     main = "Predictions with best 3 structural configurations")
axis(1, 1:nrow(preds))
for (i in seq_len(n.pr)) {lines(rep(i,2), range(preds[i,1:3]), col = "grey35", lty = 3)}
points(preds[,1], pch = 21, bg = "black")
points(preds[,2], pch = 23, bg = "red")
points(preds[,3], pch = 24, bg = "green")
legend("bottomleft", legend = c("Model 1", "Model 2", "Model 3"),
       pch = c(21, 23, 24), pt.bg = c("black", "red", "green"), inset = c(.02,.08))

## End(Not run)
```

`modelCall-class` *S4 class for calls to the fgpm function in funGp*

Description

User reminder of the fgpm function call.

Slots

`string` Object of class `"character"`. User call reminder in string format.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

---

`modelDef` *Retrieve a* `fgpm` *from within a* `Xfgpm` *object*

Description

Retrieve the `fgpm` model with index (or rank) `i` from within a `Xfgpm` object. By evaluating this code in an environment containing suitable objects `sIn`, `fIn` and `sOut` we can re-create a `fgpm` object.

Usage

```
modelDef(
  object,
  ind,
  trace = TRUE,
  pbars = TRUE,
  control.optim = list(trace = TRUE)
)
```

Arguments

`object`	A `Xfgpm` object as created by `fgpm_factory`.
`ind`	The index (or rank) of the model in `object`.
`trace`	An optional boolean indicating whether funGp-native progress messages should be displayed. Default is TRUE. See the `fgpm()` documentation for more details.
`pbars`	An optional boolean indicating whether progress bars managed by `fgpm()` should be displayed. Default is TRUE. See the `fgpm()` documentation for more details.
`control.optim`	An optional list to be passed as the control argument to `optim()`, the function in charge of the non-linear optimization of the hyperparameters. Default is list(trace = TRUE). See the `fgpm()` documentation for more details.

Details

The models are sorted by decreasing quality so `i = 1` extracts the definition of the best model.

Value

A parsed R code defining the `fgpm` model.

Note

Remind that the models are sorted by decreasing quality so `i = 1` extracts the definition of the best model.

See Also

The `[[,Xfgpm-method` that can also be used to re-create a `fgpm` object using *the same data* as that used to create the `Xfgpm` object in `object`.

Examples

```
## ===========================================================================
## Using the pre-calculated object `xm` to save time. See `?xm` to re-create
## this object.
## ===========================================================================

## 'xm@model' is the best 'fgpm' model in 'xm'
plot(xm@model)

## see the R code to use to recreate the model
modelDef(xm, i = 1)

## Not run:
    ## Define new data in a list. Using an environment would also work,
    ## including the global environment, which is the default in  `eval`.
    L <- list()
    set.seed(341)
    n.new <- 3^5
    x1 <- x2 <- x3 <- x4 <- x5 <- seq(0, 1, length = n.new^(1/5))
```

```
    ## create the data objects required to fit the model
    L$sIn <- as.matrix(expand.grid(x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = x5))
    L$fIn <- list(f1 = matrix(runif(n.new * 10), ncol = 10),
                  f2 = matrix(runif(n.new * 22), ncol = 22))
    L$sOut <- fgp_BB7(L$sIn, L$fIn, n.new)

    ## Now evaluate
    fgpm.new <- eval(modelDef(xm, i = 1), envir = L)
    plot(fgpm.new, main = "Re-created 'fgpm' model with different data")
    plot(xm[[1]], main = "Re-created 'fgpm' model with the same data")

## End(Not run)
```

`plot,fgpm-method` *Plot method for the class* `"fgpm"`

Description

This method provides a diagnostic plot for the validation of regression models. It displays a calibration plot based on the leave-one-out predictions of the output at the points used to train the model.

Usage

```
## S4 method for signature 'fgpm'
plot(x, y = NULL, ...)
```

Arguments

`x`	A `fgpm` object.
`y`	Not used.
`...`	Graphical parameters. These currently include • `xlim`, `ylim` to set the limits of the axes. • `pch`, `pt.col`, `pt.bg`, `pt.cex` to set the symbol used for the points and the related properties. • `line` to set the color used for the line. • `xlab`, `ylab`, `main` to set the labels of the axes and the main title. See **Examples**.

Details

Plot the Leave-One-Out (LOO) calibration.

Examples

```
# generating input and output data for training
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)),
                   x2 = seq(0, 1, length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10),
            f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)

# building the model
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# plotting the model
plot(m1)
# change some graphical parameters if wanted
plot(m1, line = "SpringGreen3" ,
     pch = 21, pt.col = "orangered", pt.bg = "gold",
     main = "LOO cross-validation")
```

---

plot,Xfgpm-method *Plot method for the class* "Xfgpm"

Description

Plot an object with class "Xfgpm" representing a collection of functional GP models corresponding to different structural parameters.

Two types of graphics can be shown depending on the choice of which. The choice which = "diag" is used to display diagnostics of the quality of the optimized model. Two types of diagnostic plots are shown as sub-plots by default, but each can be discarded if wanted. The choice which = "evol" is used to assess the quality of the fitted fgpm models on the basis of Leave-One-Out cross-validation.

The choice which = "diag" (default) provides two plots for assessing the quality of the output delivered by the model selection algorithm in the fgpm_factory function. The first one is a calibration plot similar to the one offered for fgpm objects by plot,fgpm-method. This plot allows to validate the absolute quality of the selected model. The second one displays the performance statistic of all the models successfully evaluated by the model selection algorithm. This provides a notion of the relative quality of the selected model with respect to the other models that can be made using the same data.

The choice which = "evol" displays the evolution of the quality of the configurations evaluated along the iterations, by the model selection algorithm in the fgpm_factory function. For each iteration, the performance statistic of all the evaluated models is printed, along with the corresponding median of the group. The plot also includes the global maximum, which corresponds to the best performance statistic obtained up to the current iteration. In this plot, it is typical to have some points falling relatively far from the maximum, even after multiple iterations. This happens mainly because we have multiple categorical features, whose alteration might change the performance statistic in a nonsmooth way. On the other hand, the points that fall below zero usually correspond to models whose hyperparameters were hard to optimize. This occurs sporadically during the log-likelihood optimization for Gaussian processes, due to the non-linearity of the objective function. As long as the maximum keeps improving and the median remains close to it, none of the two aforementioned phenomena is matter for worries. Both of them respond to the mechanism of exploration implemented in the algorithm, which makes it able to progressively move towards better model configurations.

Usage

```
## S4 method for signature 'Xfgpm'
plot(
  x,
  y = NULL,
  which = c("diag", "evol"),
  calib = TRUE,
  fitp = TRUE,
  horiz = FALSE,
  ...
)
```

Arguments

`x`	The `Xfgpm` object to plot.
`y`	Not used.
`which`	Character giving the type of plot wanted. Can take the value `"diag"` or `"evol"`. See **Examples**.
`calib`	Logical. If `TRUE` the calibration plot of the selected model will be included in the display in its "diagnostic" part if `which` is set to `"diag"`.
`fitp`	Logical. If `TRUE` a scatter plot of the quality of all explored models will be included in the display in its "diagnostic" part if `which` is set to `"diag"`.
`horiz`	Logical. Used only when `which` is `"diag"` and when both `calib` and `fitp` are `TRUE`. If `horiz` is `TRUE` the two subplots are displayed horizontally (on a same row) rather than vertically which is the default.
`...`	Other graphical parameters such as `main` of `xlab`. When `which` is `"diag"` and both `calib` and `fitp` are `TRUE`, the graphical parameters should be enclosed into a list and passed with the formal name `calib.gpars` or `fitp.gpars`.

See Also

* fgpm_factory for structural optimization of funGp models.

Examples

```
# generating input and output data
set.seed(100)
n.tr <- 2^5
```

```
x1 <- x2 <- x3 <- x4 <- x5 <- seq(0, 1, length = n.tr^(1/5))
sIn <- expand.grid(x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = x5)
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
            f2 = matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB7(sIn, fIn, n.tr)
## Not run:
# optimizing the model structure with 'fgpm_factory' (~10 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut)
# assessing the quality of the model - absolute and w.r.t. the other
# explored models
plot(xm, which = "evol")
# diagnostics (two subplots)
plot(xm, which = "diag")
plot(xm, which = "diag", horiz = TRUE)
# diagnostics (one plot)
plot(xm, which = "diag", fitp = FALSE)
plot(xm, which = "diag", calib = FALSE)
# customizing some graphical parameters
plot(xm, calib.gpars = list(xlim = c(800,1000), ylim = c(600,1200)),
     fitp.gpars = list(main = "Relative quality", legends = FALSE))


## End(Not run)
```

---

plot.predict.fgpm *Plot method for the predictions of a* fgpm *model*

Description

This method displays the predicted output values delivered by a funGp Gaussian process model.

Usage

```
## S3 method for class 'predict.fgpm'
plot(x, y = NULL, sOut.pr = NULL, calib = TRUE, sortp = TRUE, ...)
```

Arguments

x	An object with S3 class "predict.fgpm". This is a list containing the predictions and confidence bands as created by predict,fgpm-method for the S3 class "fgpm".
y	An optional vector (or 1-column matrix) containing the true values of the scalar output at the prediction points. If provided, the method will display two figures: (i) a calibration plot with true vs predicted output values, and (ii) a plot including the true and predicted output along with the confidence bands, sorted according to the increasing order of the true output. If not provided, only the second plot will be made, and the predictions will be arranged according to the increasing order of the predicted output.

sOut.pr — Alias of y, used for compatibility reasons.

calib — An optional boolean indicating if the calibration plot should be displayed. Ignored if sOut.pr is not provided. Default is TRUE.

sortp — An optional boolean indicating if the plot of sorted output should be displayed. Default is TRUE.

... — Additional arguments affecting the display. Since this method allows to generate two plots from a single function call, the extra arguments for each plot should be included in a list. For the calibration plot, the list should be called *calib.gpars*. For the plot of the output in increasing order, the list should be called *sortp.gpars*. The following typical graphics parameters are valid entries of both lists: *xlim*, *ylim*, *xlab*, *ylab*, *main*. The boolean argument *legends* can also be included in any of the two lists in order to control the display of legends in the corresponding plot.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

See Also

* fgpm for the construction of funGp models;

* plot,fgpm-method for model diagnostic plots;

* simulate,fgpm-method for simulations based on a funGp model;

* plot.simulate.fgpm for simulation plots.

Examples

```
# plotting predictions without the true output values_______________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0, 1, length = sqrt(n.tr)),
                   x2 = seq(0, 1, length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
            f2 = matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# making predictions
n.pr <- 100
sIn.pr <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.pr)),
                                x2 = seq(0,1,length = sqrt(n.pr))))
fIn.pr <- list(f1 = matrix(runif(n.pr * 10), ncol = 10),
               f2 = matrix(runif(n.pr * 22), ncol = 22))
m1.preds <- predict(m1, sIn.pr = sIn.pr, fIn.pr = fIn.pr)

# plotting predictions
plot(m1.preds)
```

```
# plotting predictions and true output values_______________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0, 1, length = sqrt(n.tr)),
                   x2 = seq(0, 1, length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
            f2 = matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# making predictions
n.pr <- 100
sIn.pr <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.pr)),
                                x2 = seq(0,1,length = sqrt(n.pr))))
fIn.pr <- list(f1 = matrix(runif(n.pr*10), ncol = 10),
               f2 = matrix(runif(n.pr*22), ncol = 22))
m1.preds <- predict(m1, sIn.pr = sIn.pr, fIn.pr = fIn.pr)

# generating output data for validation
sOut.pr <- fgp_BB3(sIn.pr, fIn.pr, n.pr)

# plotting predictions. Note that the 2-nd argument is the output, 'y'
plot(m1.preds, sOut.pr)

# only calibration plot
plot(m1.preds, sOut.pr = sOut.pr, sortp = FALSE)

# only sorted output plot
plot(m1.preds, sOut.pr = sOut.pr, calib = FALSE)
```

---

plot.simulate.fgpm *Plot method for the simulations of a* fgpm *model*

Description

This method displays the simulated output values delivered by a funGp Gaussian process model.

Usage

```
## S3 method for class 'simulate.fgpm'
plot(x, y = NULL, detail = NA, ...)
```

Arguments

x	An object with S3 class `simulate.fgpm` as created by simulate,fgpm-method.
y	Not used.

`detail` An optional character string specifying the data elements that should be included in the plot, to be chosen between "light" and "full". A *light* plot will include only the simulated values, while a *full* plot will also include the predicted mean and confidence bands at the simulation points. This argument will only be used if full simulations (including the mean and confidence bands) are provided, otherwise it will be ignored. See simulate,fgpm-method for more details on the generation of light and full simulations.

`...` Additional arguments affecting the display. The following typical graphics parameters are valid entries: *xlim*, *ylim*, *xlab*, *ylab*, *main*. The boolean argument *legends* can also be included in any of the two lists in order to control the display of legends in the corresponding plot.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

See Also

* fgpm for the construction of funGp models;

* plot,fgpm-method for model diagnostic plots;

* predict,fgpm-method for predictions based on a funGp model;

* plot.predict.fgpm for prediction plots.

Examples

```
# plotting light simulations_________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0, 1, length = sqrt(n.tr)),
                   x2 = seq(0, 1, length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
            f2 = matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# making light simulations
n.sm <- 100
sIn.sm <- as.matrix(expand.grid(x1 = seq(0, 1, length = sqrt(n.sm)),
                                x2 = seq(0, 1, length = sqrt(n.sm))))
fIn.sm <- list(f1 = matrix(runif(n.sm * 10), ncol = 10),
               f2 = matrix(runif(n.sm * 22), ncol = 22))
simsl <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm)

# plotting light simulations
plot(simsl)


# plotting full simulations__________________________________________________
# building the model
```

```
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0, 1, length = sqrt(n.tr)),
                   x2 = seq(0, 1, length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
            f2 = matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# making full simulations
n.sm <- 100
sIn.sm <- as.matrix(expand.grid(x1 = seq(0, 1, length = sqrt(n.sm)),
                                x2 = seq(0, 1 ,length = sqrt(n.sm))))
fIn.sm <- list(f1 = matrix(runif(n.sm * 10), ncol = 10),
               f2 = matrix(runif(n.sm * 22), ncol = 22))
simsf <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm,
                  detail = "full")

# plotting full simulations in "full" mode
plot(simsf)

# plotting full simulations in "light" mode
plot(simsf, detail = "light")
```

---

precalculated_Xfgpm_objects
Precalculated Xfgpm objects

Description

A dataset containing the results of the application of `fgpm_factory` to `fgp_BB7` analytic black-box function. See **Examples** for details.

Format

Five objects of class "`Xfgpm`":

xm With 32 training points and default parameters.

xm25 With 32 training points and 25 iterations of the algorithm.

xmc With 32 training points and customized solution space.

xmh With 32 training points and customized heuristic parameters.

xms With 32 training points and a time budget constraint and large number of iterations.

Examples

```
## Not run:

##################################################################
## Construction of xm object with default parameters (~12 seconds)
##################################################################
set.seed(100)
n.tr <- 32
x1 <- x2 <- x3 <- x4 <- x5 <- seq(0,1,length = n.tr^(1/5))
sIn <- expand.grid(x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = x5)
fIn <- list(f1 = matrix(runif(n.tr * 10), ncol = 10),
            f2 = matrix(runif(n.tr * 22), ncol = 22))
sOut <- fgp_BB7(sIn, fIn, n.tr)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut)

##################################################################
## Construction of xm25 object with 25 iterations (~20 seconds)
##################################################################
xm25 <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut,
                     setup = list(n.iter = 25))

##################################################################
## Construction of xmc object with customized solution space (~12 seconds)
##################################################################
myctr <- list(s_keepOn = c(1,2), # keep both scalar inputs always on
f_keepOn = c(2), # keep f2 always active
f_disTypes = list("2" = c("L2_byindex")), # only use L2_byindex distance for f2
f_fixDims = matrix(c(2,4), ncol = 1), # f2 projected in dimension 4
f_maxDims = matrix(c(1,5), ncol = 1), # f1 projected in dimension max 5
f_basTypes = list("1" = c("B-splines")), # only use B-splines projection for f1
kerTypes = c("matern5_2", "gauss")) # test only Matern 5/2 and Gaussian kernels
xmc <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, ctraints = myctr)

##################################################################
## Construction of xmc object with customized heuristic parameters (~15 seconds)
##################################################################
mysup <- list(n.iter = 30, n.pop = 12, tao0 = .15, dop.s = 1.2,
              dop.f = 1.3, delta.f = 4, dispr.f = 1.1, q0 = .85,
              rho.l = .2, u.gbest = TRUE, n.ibest = 2, rho.g = .08)
xmh <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, setup = mysup)

##################################################################
## Construction of xmc object with time budget constraint (~60 seconds)
##################################################################
mysup <- list(n.iter = 2000)
mytlim <- 60
xms <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut,
                    setup = mysup, time.lim = mytlim)


## End(Not run)
```

predict,fgpm-method *Prediction from a* fgpm *Gaussian process model*

Description

This method enables prediction based on a fgpm model, at any given set of points. Check fgpm for information on how to create fgpm models.

Usage

```
## S4 method for signature 'fgpm'
predict(object, sIn.pr = NULL, fIn.pr = NULL, detail = c("light", "full"), ...)
```

Arguments

object	An object of class fgpm corresponding to the funGp model that should be used to predict the output.
sIn.pr	An optional matrix of scalar input coordinates at which the output values should be predicted. Each column is interpreted as a scalar input variable and each row as a coordinate. Either scalar input coordinates (sIn.pr), functional input coordinates (fIn.pr), or both must be provided.
fIn.pr	An optional list of functional input coordinates at which the output values should be predicted. Each element of the list is interpreted as a functional input variable. Every functional input variable should be provided as a matrix with one curve per row. Either scalar input coordinates (sIn.pr), functional input coordinates (fIn.pr), or both must be provided.
detail	An optional character specifying the extent of information that should be delivered by the method, to be chosen between "light" (default) and "full". *Light* predictions produce a list including the predicted mean, standard deviation and limits of the 95% confidence intervals at the prediction points. *Full* predictions produce the same information as light ones, in addition to the training-prediction cross-covariance matrix and the prediction auto-covariance matrix.
...	Not used.

Value

An object of class "list" containing the data structures linked to predictions. For *light* predictions, the list will include the mean, standard deviation and limits of the 95% confidence intervals at the prediction points. For *full* predictions, it will include the same information, plus the training-prediction cross-covariance matrix and the prediction auto-covariance matrix.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

See Also

* plot.predict.fgpm for the prediction plot of a fgpm model;
* simulate,fgpm-method for simulations based on a fgpm model;
* plot.simulate.fgpm for the simulation plot of a fgpm model.

Examples

```
# light predictions______________________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# generating input data for prediction
n.pr <- 100
sIn.pr <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.pr)),
                                x2 = seq(0,1,length = sqrt(n.pr))))
fIn.pr <- list(f1 = matrix(runif(n.pr*10), ncol = 10), matrix(runif(n.pr*22), ncol = 22))

# making predictions
m1.preds <- predict(m1, sIn.pr = sIn.pr, fIn.pr = fIn.pr)

# checking content of the list
summary(m1.preds)

# ~R output:~
#         Length Class  Mode
# mean    100    -none- numeric
# sd      100    -none- numeric
# lower95 100    -none- numeric
# upper95 100    -none- numeric

# plotting predictions
plot(m1.preds)


# comparison against true output__________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# making predictions
n.pr <- 100
sIn.pr <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.pr)),
```

```
                                x2 = seq(0,1,length = sqrt(n.pr))))
fIn.pr <- list(f1 = matrix(runif(n.pr*10), ncol = 10), matrix(runif(n.pr*22), ncol = 22))
m1.preds <- predict(m1, sIn.pr = sIn.pr, fIn.pr = fIn.pr)

# generating output data for validation
sOut.pr <- fgp_BB3(sIn.pr, fIn.pr, n.pr)

# plotting predictions along with true output values
plot(m1.preds, sOut.pr)


# full predictions_________________________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# making full predictions
n.pr <- 100
sIn.pr <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.pr)),
                                x2 = seq(0,1,length = sqrt(n.pr))))
fIn.pr <- list(f1 = matrix(runif(n.pr*10), ncol = 10), matrix(runif(n.pr*22), ncol = 22))
m1.preds_f <- predict(m1, sIn.pr = sIn.pr, fIn.pr = fIn.pr, detail = "full")

# checking content of the list
summary(m1.preds_f)

# ~R output:~
#         Length Class  Mode
# mean      100  -none- numeric
# sd        100  -none- numeric
# K.tp     2500  -none- numeric
# K.pp    10000  -none- numeric
# lower95   100  -none- numeric
# upper95   100  -none- numeric

# plotting predictions
plot(m1.preds)
```

`simulate,fgpm-method` *Random sampling from a* `fgpm` *model*

Description

This method enables simulation of Gaussian process values at any given set of points based on a pre-built `fgpm` model. Check `fgpm` for information on how to create funGp models.

Usage

```
## S4 method for signature 'fgpm'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  sIn.sm = NULL,
  fIn.sm = NULL,
  nugget.sm = 0,
  detail = c("light", "full"),
  ...
)
```

Arguments

`object`	An object of class fgpm corresponding to the funGp model from which simulations must be performed.
`nsim`	An optional integer indicating the number of samples to produce. Default is 1.
`seed`	An optional value interpreted as an integer, that will be used as argument of `set.seed` just before simulating the response values.
`sIn.sm`	An optional matrix of scalar input coordinates at which the output values should be simulated. Each column is interpreted as a scalar input variable and each row as a coordinate. Either scalar input coordinates (sIn.sm), functional input coordinates (fIn.sm), or both must be provided.
`fIn.sm`	An optional list of functional input coordinates at which the output values should be simulated. Each element of the list is interpreted as a functional input variable. Every functional input variable should be provided as a matrix with one curve per row. Either scalar input coordinates (sIn.sm), functional input coordinates (fIn.sm), or both must be provided.
`nugget.sm`	An optional number corresponding to a numerical nugget effect. If provided, this number is added to the main diagonal of the simulation covariance matrix in order to prevent numerical instabilities during Cholesky decomposition. A small number in the order of 1e-8 is often enough. Default is 0.
`detail`	An optional character specifying the extent of information that should be delivered by the method, to be chosen between `"light"` (default) and `"full"`. *Light* simulations produce a matrix of simulated output values, with as many rows as requested random samples. *Full* simulations produce a list with the matrix of simulated output values, along with the predicted mean, standard deviation and limits of the 95% confidence intervals at the simulation points.
`...`	Not used.

Value

An object containing the data structures linked to simulations. For *light* simulations, the output will be a matrix of simulated output values, with as many rows as requested random samples. For *full* simulations, the output will be a list with the matrix of simulated output values, along with the predicted mean, standard deviation and limits of the 95% confidence intervals at the simulation points.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

See Also

* plot.simulate.fgpm for the simulation plot of a fgpm model;

* predict,fgpm-method for predictions based on a fgpm model;

* plot.predict.fgpm for the prediction plot of a fgpm model.

Examples

```
# light simulations _______________________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# generating input data for simulation
n.sm <- 100
sIn.sm <- as.matrix(expand.grid(x1 = seq(0,1,length = sqrt(n.sm)),
                                x2 = seq(0,1,length = sqrt(n.sm))))
fIn.sm <- list(f1 = matrix(runif(n.sm*10), ncol = 10), matrix(runif(n.sm*22), ncol = 22))

# making light simulations
m1.sims_l <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm)

# plotting light simulations
plot(m1.sims_l)


# full simulations ________________________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# making full simulations
m1.sims_f <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm, detail = "full")

# checking content of the list
summary(m1.sims_f)
```

```
# ~R output:~
#         Length Class  Mode
# sims    1000   -none- numeric
# mean     100   -none- numeric
# sd       100   -none- numeric
# lower95  100   -none- numeric
# upper95  100   -none- numeric

# plotting full simulations in full mode
plot(m1.sims_f)

# plotting full simulations in light mode
plot(m1.sims_f, detail = "light")
```

---

summary,fgpm-method   *Summary method for* `fgpm` *objects*

Description

Display the structure of a `fgpm` object and the value of the parameters (variance and length-scales).

Usage

```
## S4 method for signature 'fgpm'
summary(object, ...)
```

Arguments

`object`	An `fgpm` object.
`...`	Not used yet.

Note

This method is actually identical to the `show` method for this class which is called when the name of the object is entered in an interactive session.

Examples

```
m <- xm@model
class(m)
summary(m)
m
```

summary,Xfgpm-method *Summary method for* `Xfgpm` *objects*

Description

Display a summary of the structure of a `Xfgpm` object, with a short description of up to `n` `fgpm` objects visited during the ACO optimization.

Usage

```
## S4 method for signature 'Xfgpm'
summary(object, n = 24, ...)
```

Arguments

`object`	A `Xfgpm` object.
`n`	Maximal number of lines (`fgpm` objects) to show.
`...`	Not used yet.

Details

The displayed information depends on the number of candidate inputs, in order to maintain compact tables. The inputs are labelled with integer suffixes, the prefix being `"X"` for scalar inputs and `"F"` for functional inputs.

- With a small number of inputs, the list contains only one data frame. For each candidate input (either scalar or functional) a column with the input name indicates if the input is active (cross `x`) or not (white space) in the `fgpm` object corresponding to the row. For each functional variable also shown are: the distance used `D_`, the dimension `Bas_` after dimension reduction, the type of basis used `B_`. Remind that the kernel (`Kern`) is the same for all functional inputs. Also shown is the value of the Leave-One-Out coefficient `Q2`.
- With a large number of inputs, the list contains two data frames. The first one tells which inputs are active among the scalar and functional candidate inputs. The second data frame gives more details for functional inputs as before.

Value

An object inheriting from `list`, actually a list containing one or two data frames depending on the number of inputs. In each data frame, the `n` rows provide information on the best `fgpm` objects visited.

Examples

```
summary(xm)
```

---

update,fgpm-method *Easy update of* fgpm *models*

Description

This method enables the update of data or hyperparameters of a fgpm model. It corresponds to an object of the class fgpm. The method allows addition, subtraction and substitution of data points, as well as substitution and re-estimation of hyperparameters.

Usage

```
## S4 method for signature 'fgpm'
update(
  object,
  sIn.nw = NULL,
  fIn.nw = NULL,
  sOut.nw = NULL,
  sIn.sb = NULL,
  fIn.sb = NULL,
  sOut.sb = NULL,
  ind.sb = NULL,
  ind.dl = NULL,
  var.sb = NULL,
  ls_s.sb = NULL,
  ls_f.sb = NULL,
  var.re = FALSE,
  ls_s.re = FALSE,
  ls_f.re = FALSE,
  extend = FALSE,
  trace = TRUE,
  pbars = TRUE,
  control.optim = list(trace = TRUE),
  ...
)
```

Arguments

object	An object of class fgpm corresponding to the funGp model to update.
sIn.nw	An optional matrix of scalar input values to be added to the model. Each column must match an input variable and each row a scalar coordinate.
fIn.nw	An optional list of functional input values to be added to the model. Each element of the list must be a matrix containing the set of curves corresponding to one functional input.
sOut.nw	An optional vector (or 1-column matrix) containing the values of the scalar output at the new input points.

`sIn.sb`	An optional matrix of scalar input values to be used as substitutes of other scalar input values already stored in the model. Each column must match an input variable and each row a coordinate.
`fIn.sb`	An optional list of functional input values to be added to the model. Each element of the list must be a matrix containing the set of curves corresponding to one functional input.
`sOut.sb`	An optional vector (or 1-column matrix) containing the values of the scalar output at the substituting input points.
`ind.sb`	An optional numeric array indicating the indices of the input and output points stored in the model, that should be replaced by the values specified through sIn.sb, fIn.sb and/or sOut.sb.
`ind.dl`	An optional numeric array indicating the indices of the input and output points stored in the model that should be deleted.
`var.sb`	An optional number indicating the value that should be used to substitute the current variance parameter of the model.
`ls_s.sb`	An optional numerical array indicating the values that should be used to substitute the current length-scale parameters for the scalar inputs of the model.
`ls_f.sb`	An optional numerical array indicating the values that should be used to substitute the current length-scale parameters for the functional inputs of the model.
`var.re`	An optional boolean indicating whether the variance parameter should be re-estimated. Default is FALSE.
`ls_s.re`	An optional boolean indicating whether the length-scale parameters of the scalar inputs should be re-estimated. Default is FALSE.
`ls_f.re`	An optional boolean indicating whether the length-scale parameters of the functional inputs should be re-estimated. Default is FALSE.
`extend`	An optional boolean indicating whether the re-optimization should extend from the current hyperparameters of the model using them as initial points. Default is FALSE, meaning that the re-optimization picks brand new initial points in the way described in `fgpm()`. If both hyperparameter substitution and re-estimation are requested in a single `update()` call and `extend` is set to `TRUE`, the values used as initial points for the re-optimization are those stored by the model after the substitution step.
`trace`	An optional boolean indicating whether funGp-native progress messages and a summary update should be displayed. Default is TRUE. See the `fgpm()` documentation for more details.
`pbars`	An optional boolean indicating whether progress bars managed by `fgpm()` should be displayed (in case the update requires an `fgpm()` call). Default is TRUE. See the `fgpm()` documentation for more details.
`control.optim`	An optional list to be passed as the control argument to `optim()` (in case the update requires an `fgpm()` call), the function in charge of the non-linear optimization of the hyperparameters. Default is list(trace = TRUE). See the `fgpm()` documentation for more details.
`...`	Not used.

Details

The arguments listed above enable the completion of the following updating tasks:

- **Deletion** of data points: ind.dl;
- **Addition** of data points: sIn.nw, fIn.nw, sOut.nw;
- **Substitution** of data points: sIn.sb, fIn.sb, sOut.sb, ind.sb;
- **Substitution** of hyperparameters: var.sb, ls_s.sb, ls_f.sb;
- **Re-estimation** of hyperparameters: var.re, ls_s.re, ls_f.re.

All the arguments listed above are optional since any of these tasks can be requested without need to request any of the other tasks. In fact, most of the arguments can be used even if the other arguments related to the same task are not. For instance, the re-estimation of the variance can be requested via var.re without requiring re-estimation of the scalar or functional length-scale parameters. The only two exceptions are: (i) for data addition, the new output sOut.nw should always be provided and the new input points should correspond to the set of variables already stored in the fgpm object passed for update; and (ii) for data substitution, the argument ind.sb is always mandatory.

Conflicting task combinations:

- Data points deletion and substitution;
- Substitution and re-estimation of the same hyperparameter.

Note that the parameters of the model will not be updated after modifying the model unless explicitly requested through the var.re, ls_s.re and ls_f.re arguments. If, for instance, some points are added to the model without requesting parameter re-estimation, the new data will be included in the training-training and training-prediction covariance matrices, but the hyperparameters will not be updated. This allows to make updates in the data that might help to improve predictions, without the immediate need to perform a training procedure that could be time consuming. At any later time, the user is allowed to request the re-estimation of the hyperparameters, which will make the model fully up to date.

Value

An object of class fgpm representing the updated funGp model.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

See Also

* fgpm for creation of a funGp model;

* predict,fgpm-method for predictions based on a `fgpm` model;

* simulate,fgpm-method for simulations based on a `fgpm` model.

Examples

```
# deletion and addition of data points_____________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# deleting two points
ind.dl <- sample(1:m1@n.tot, 2)
m1up <- update(m1, ind.dl = ind.dl)

# adding five points
n.nw <- 5
sIn.nw <- matrix(runif(n.nw * m1@ds), nrow = n.nw)
fIn.nw <- list(f1 = matrix(runif(n.nw*10), ncol = 10), f2 = matrix(runif(n.nw*22), ncol = 22))
sOut.nw <- fgp_BB3(sIn.nw, fIn.nw, n.nw)
m1up <- update(m1, sIn.nw = sIn.nw, fIn.nw = fIn.nw, sOut.nw = sOut.nw)


# substitution of data points______________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# generating substituting input data for updating
n.sb <- 2
sIn.sb <- matrix(runif(n.sb * m1@ds), nrow = n.sb)
fIn.sb <- list(f1 = matrix(runif(n.sb*10), ncol = 10), f2 = matrix(runif(n.sb*22), ncol = 22))

# generating substituting output data for updating
sOut.sb <- fgp_BB3(sIn.sb, fIn.sb, n.sb)

# generating indices for substitution
ind.sb <- sample(1:(m1@n.tot), n.sb)

# updating all, the scalar inputs, functional inputs and the outputs
m1up <- update(m1, sIn.sb = sIn.sb, fIn.sb = fIn.sb, sOut.sb = sOut.sb, ind.sb = ind.sb)

# updating only some of the data structures
m1up1 <- update(m1, sIn.sb = sIn.sb, ind.sb = ind.sb) # only the scalar inputs
m1up2 <- update(m1, sOut.sb = sOut.sb, ind.sb = ind.sb) # only the outputs
m1up3 <- update(m1, sIn.sb = sIn.sb, sOut.sb = sOut.sb, ind.sb = ind.sb) # the scalar inputs
                                                                        # and the outputs
```

```
# substitution of hyperparameters_____________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# defining hyperparameters for substitution
var.sb <- 3
ls_s.sb <- c(2.44, 1.15)
ls_f.sb <- c(5.83, 4.12)

# updating the model
m1up <- update(m1, var.sb = var.sb, ls_s.sb = ls_s.sb, ls_f.sb = ls_f.sb)


# re-estimation of hyperparameters____________________________________________________________
# building the model
set.seed(100)
n.tr <- 25
sIn <- expand.grid(x1 = seq(0,1,length = sqrt(n.tr)), x2 = seq(0,1,length = sqrt(n.tr)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
sOut <- fgp_BB3(sIn, fIn, n.tr)
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)

# re-estimating the hyperparameters
m1up <- update(m1, var.re = TRUE) # only the variance
m1up <- update(m1, ls_s.re = TRUE) # only the scalar length-scale parameters
m1up <- update(m1, ls_s.re = TRUE, ls_f.re = TRUE) # all length-scale parameters
m1up <- update(m1, var.re = TRUE, ls_s.re = TRUE, ls_f.re = TRUE) # all hyperparameters

# same as above but now extending optimization from previously stored values
m1up <- update(m1, var.re = TRUE, extend = TRUE)
m1up <- update(m1, ls_s.re = TRUE, extend = TRUE)
m1up <- update(m1, ls_s.re = TRUE, ls_f.re = TRUE, extend = TRUE)
m1up <- update(m1, var.re = TRUE, ls_s.re = TRUE, ls_f.re = TRUE, extend = TRUE)
```

`which_on` *Indices of active inputs in a given model structure*

Description

The fgpm_factory function returns an object of class "`Xfgpm`" with the function calls of all the evaluated models stored in the `@log.success@args` and `@log.crashes@args` slots. The `which_on` function interprets the arguments linked to any structural configuration and returns a list with two elements: (i) an `array` of indices of the scalar inputs kept active; and (ii) an `array` of indices of the functional inputs kept active.

Usage

```
which_on(sIn = NULL, fIn = NULL, args)
```

Arguments

sIn	An optional matrix of scalar input coordinates with all the orignal scalar input variables. This is used only to know the total number of scalar input variables. Any `matrix` with as many columns as original scalar input variables could be used instead.
fIn	An optional list of functional input coordinates with all the original functional input variables. This is used only to know the total number of functional input variables. Any `list` with as many elements as original functional input variables could be used instead.
args	An object of class "`modelCall`", which specifies the model structure for which the active inputs should be extracted.

Value

An object of class "`list`", containing the following information extracted from the *args* parameter: (i) an array of indices of the scalar inputs kept active; and (ii) an array of indices of the functional inputs kept active.

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

References

Betancourt, J., Bachoc, F., Klein, T., Idier, D., Rohmer, J., and Deville, Y. (2024), "funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs". *Journal of Statistical Software*, **109**, 5, 1–51. (doi:10.18637/jss.v109.i05)

Betancourt, J., Bachoc, F., Klein, T., Idier, D., Rohmer, J., and Deville, Y. (2024), "funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs". *Journal of Statistical Software*, **109**, 5, 1–51. (doi:10.18637/jss.v109.i05)

Betancourt, J., Bachoc, F., and Klein, T. (2020), R Package Manual: "Gaussian Process Regression for Scalar and Functional Inputs with funGp - The in-depth tour". *RISCOPE project*. [HAL]

See Also

* get_active_in for details on how to obtain the data structures linked to the active inputs;

* modelCall for details on the *args* argument;

* fgpm_factory for funGp heuristic model selection;

* Xfgpm for details on object delivered by fgpm_factory.

Examples

```
# extracting the indices of the active inputs in an optimized model________________________
# use precalculated Xfgpm object named xm
# active inputs in the best model
xm@log.success@args[[1]] # the full fgpm call
set.seed(100)
n.tr <- 32
sIn <- expand.grid(x1 = seq(0,1,length = n.tr^(1/5)), x2 = seq(0,1,length = n.tr^(1/5)),
x3 = seq(0,1,length = n.tr^(1/5)), x4 = seq(0,1,length = n.tr^(1/5)),
x5 = seq(0,1,length = n.tr^(1/5)))
fIn <- list(f1 = matrix(runif(n.tr*10), ncol = 10), f2 = matrix(runif(n.tr*22), ncol = 22))
which_on(sIn, fIn, xm@log.success@args[[1]]) # only the indices extracted by which_on
```

`Xfgpm-class` *S4 class for funGp model selection data structures*

Description

This is the formal representation of the assembly of data structures delivered by the model selection routines in the funGp package. An `Xfgpm` object contains the trace of an optimization process, conducted to build Gaussian process models of outstanding performance.

- **Main methods**
  fgpm_factory: structural optimization of `fgpm` models, creator of the `"Xfgpm"` class.

- **Plotters**
  plot,Xfgpm-method: plot of the evolution of the algorithm with `which = "evolution"` or of the absolute and relative quality of the optimized model with `which = "diag"`.

Slots

`factoryCall` Object of class `"factoryCall"`. User call reminder.

`model` Object of class `"fgpm"`. Model selected by the heuristic structural optimization algorithm.

`stat` Object of class `"character"`. Performance measure optimized to select the model. To be set from "Q2loocv", "Q2hout".

`fitness` Object of class `"numeric"`. Value of the performance measure for the selected model.

`structure` Object of class `"data.frame"`. Structural configuration of the selected model.

`log.success` Object of class `"antsLog"`. Record of models successfully evaluated during the structural optimization. It contains the structural configuration both in data.frame and `"modelCall"` format, along with the fitness of each model. The models are sorted by fitness, starting with the best model in the first position.

`log.crashes` Object of class `"antsLog"`. Record of models crashed during the structural optimization. It contains the structural configuration of each model, both in data.frame and `"modelCall"` format.

`n.solspace` Object of class "numeric". Number of possible structural configurations for the optimization instance resolved.

`n.explored` Object of class "numeric". Number of structural configurations successfully evaluated by the algorithm.

`details` Object of class "list". Further information about the parameters of the ant colony optimization algorithm and the evolution of the fitness along the iterations.

`sIn` An object of class "matrix" containing a copy of the provided scalar inputs.

`fIn` An object of class "list" containing a copy of the provided functional inputs.

`sOut` An object of class "matrix" containing a copy of the provided outputs.

Useful material

- **Manual** funGp: An R Package for Gaussian Process Regression with Scalar and Functional Inputs (doi:10.18637/jss.v109.i05)

Author(s)

José Betancourt, François Bachoc, Thierry Klein and Jérémy Rohmer

---

`[[,Xfgpm-method` *Refit a* `fgpm` *model in a* `Xfgpm` *object*

Description

Refit a `fgpm` model as described in a `Xfgpm` object.

Usage

```
## S4 method for signature 'Xfgpm'
x[[i]]
```

Arguments

`x`	A `Xfgpm` object.
`i`	An integer giving the index of the model to refit. The models are in decreasing fit quality as assessed by the Leave-One-Out Q^2.

Caution

While the syntax may suggest that the function *extracts* a fitted `fgpm` model, this not true. The `fgpm` model is refitted using the call that was used when this model was assessed. The refitted `fgpm` model keeps the same structural parameters as the one assessed (active variables, kernel, ...), but since the optimization uses random initial values, the optimized hyper-parameters may differ from those of the corresponding `fgpm` in the `Xfgpm` object `x`. As a result, the model can be different and show a different LOO performance.

Note

The slot @model returns the best fgpm as assessed in a Xfgm model x. So this model can be expected to be close to the same as x[[1]]. Yet due to the refit, the two models x@model and x[[1]] can differ, see the explanations in the **Caution** section.

See Also

The modelDef function to extract the definition of a fgpm model e.g., to evaluate it using new data sIn, fIn and sOut.

Examples

```
## see `?xm` to see how to recreate the pre-caclulated `Xfgpm` object `xm`.
xm[[2]]
```

Index