Package ‘funGp’

April 22, 2020

Type Package

Title Gaussian Process Models for Scalar and Functional Inputs

Version 0.1.0

Maintainer Jose Betancourt <djbetancourt@uninorte.edu.co>

Description Construction and smart selection of Gaussian process models with emphasis on treatment of functional inputs. 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). 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.archives-ouvertes.fr/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/

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

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Collate '0_funGp_Doc.R' '0_show_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'
'8_outilsStats.R'

NeedsCompilation no

Author Jose Betancourt [cre, aut],
François Bachoc [aut],
Thierry Klein [aut],
Deborah Idier [ctb],
Jeremy Rohmer [ctb]

Repository CRAN

Date/Publication 2020-04-22 17:24:10 UTC

R topics documented:

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

Description

Construction and smart selection of Gaussian process models with emphasis on treatment of functional inputs.

Base functionalities

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

Model selection

- **Main methods**
  fgpm_factory: structural parameter optimization
- **Plotters pre-optimization**
  decay: regularized initial pheromones
  decay2probs: normalized initial pheromones
- **Plotters post-optimization**
  plotX: absolute and relative quality of the optimized model
  plotEvol: evolution of the algorithm

Useful material

- **Manual** Gaussian Process Regression for Scalar and Functional Inputs with funGp - The in-depth tour
- **Paper** Gaussian process metamodeling of functional-input code for coastal flood hazard assessment
- **Tech. report** Ant Colony Based Model Selection for Functional-Input Gaussian Process Regression

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.

---

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

Description

Register of model structures and their performance statistic, 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 and Thierry Klein

---

black-boxes — *Analytic black-boxes for the exploration of the funGp package*

Description

Set of black-box analytic functions for the discovering and testing of funGp functionalities.

Usage

```
## Own analytical function 1
## -------------------------
## x1 * sin(x2) + x1 * mean(f1) - x2^2 * diff(range(f2))
fgp_BB1(sIn, fIn, n.tr)


## Own analytical function 2
## -------------------------
## x1 * sin(x2) + mean(exp(x1 * t1) * f1) - x2^2 * mean(f2^2 * t2)
fgp_BB2(sIn, fIn, n.tr)
```

```
## First analytical example in Muehlenstaedt, Fruth & Roustant (2016)
## ----------------------------------------------------------------------
## x1 + 2 * x2 + 4 * mean(t1 * f1) + mean(f2)
fgp_BB3(sIn, fIn, n.tr)


## Second analytical example in preprint of Muehlenstaedt, Fruth & Roustant (2016)
## --------------------------------------------------------------------------------
## a = (x2 - (5/(4*pi^2)) * x1^2 + (5/pi) * x1 - 6)^2
## b = 10 * (1 - (1/(8*pi))) * cos(x1)
## c = 10
## d = (4/3) * pi * (42 * mean(f1*(1-t1)) + pi * (((x1+5)/5) + 15) * mean(t2*f2))
## a + b + c + d
fgp_BB4(sIn, fIn, n.tr)


## Second analytical example in final version of Muehlenstaedt, Fruth & Roustant (2016)
## -------------------------------------------------------------------------------------
## a = (x2 - (5/(4*pi^2)) * x1^2 + (5/pi) * x1 - 6)^2
## b = 10 * (1 - (1/(8*pi))) * cos(x1)
## c = 10
## d <- (4/3) * pi * (42 * mean(15*f1*(1-t1)-5) + pi * (((x1+5)/5) + 15) * mean(15*t2*f2))
## a + b + c + d
fgp_BB5(sIn, fIn, n.tr)


## Inspired by the analytical example in Nanty, Helbert, Marrel, Pérot, Prieur (2016)
## -----------------------------------------------------------------------------------
## 2 * x1^2 + 2 * mean(f1 + t1) + 2 * mean(f2 + t2) + max(f2) + x2
fgp_BB6(sIn, fIn, n.tr)


## Inspired by the second analytical example in final version of Muehlenstaedt et al (2016)
## -----------------------------------------------------------------------------------------
## a = (x2 + 4*x3 - (5/(4*pi^2)) * x1^2 + (5/pi) * x1 - 6)^2
## b = 10 * (1 - (1/(8*pi))) * cos(x1) * x2^2 * x5^3
## c = 10
## d <- (4/3) * pi * (42 * sin(x4) * mean(15*f1*(1-t1)-5) +
                                    pi * (((x1*x5+5)/5) + 15) * mean(15*t2*f2))
## a + b + c + d
fgp_BB7(sIn, fIn, n.tr)
```

Arguments

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.

**n.tr*: Object of class "numeric". The number of input points provided and correspondingly, the number of observations to produce.

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.

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. [SA-JUQ]

---

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.

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 and Thierry Klein

See Also

* decay 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.

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 and Thierry Klein

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 and Thierry Klein

`fgpKern-class` *S4 class for structures linked to the kernel of a funGp 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 and Thierry Klein

`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
)
```

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 to 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 are 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 and the in-depth package manual. 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 function 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, this 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_byindex" 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 from the optim function regarding the optimization of the hyperparameters should be printed to console. Default is TRUE.
pbars	an optional boolean indicating if progress bars should be displayed. Default is TRUE.

Value

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

Author(s)

José Betancourt, François Bachoc and Thierry Klein

References

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. [RESS] [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

* plotLOO for diagnostic plot of a funGp model;

* predict for predictions based on a funGp model;

* simulate for simulations based on a funGp model;

* update for post-creation updates on a funGp 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
plotLOO(ms)
plotLOO(mf)
plotLOO(msf)

# printing the three models
ms # equivalent to show(ms)
mf # equivalent to show(mf)
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
plotPreds(m1, preds = 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
plotSims(m1, 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
plotLOO(m1)
```

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


# 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)
```

`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: output estimation at new input points based on a funGp model
  simulate: random sampling from a funGp Gaussian process model
  update: modification of data and hyperparameters of a funGp model
- **Plotters**
  plotLOO: leave-one-out diagnostic plot for a funGp model
  plotPreds: plot for predictions of a funGp model
  plotSims: plot for simulations of a funGp 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, $LInvY = L^(-1) * sOut$.

Useful material

- **Manual** Gaussian Process Regression for Scalar and Functional Inputs with funGp - The in-depth tour

Author(s)

José Betancourt, François Bachoc and Thierry Klein

`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 as 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 = FALSE,
  pbars = 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 to 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 Q2, which comes from training the model with a subset of the points, and then estimate 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 Q2, 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 hyperparameters optimizations that should be conducted, nonetheless, fgpm_factory implements the virtual equations introduced by Dubrule (1983) for Gaussian processes, which require a single hyperparameters 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 index 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 index 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 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 iterations, 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 ≤ `ls_s.hyp[i]` ≤ 2*max(`sMs[[i]]`), for i in 1 to the number of scalar inputs, 1e-10 ≤ `ls_f.hyp[i]` ≤ 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 regarding the optimization of the hyperparameters should be printed to console. Default is FALSE.
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 and Thierry Klein

References

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. [RESS] [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

* plotX for diagnostic plots for a fgpm_factory output and selected model;
* plotEvol for a plot of the evolution of the model selection algorithm in fgpm_factory;
* predict for predictions based on a funGp model;
* simulate for simulations based on a funGp model;
* update for post-creation updates on a funGp model.

Examples

```
# calling fgpm_factory with the default arguments_________________________________________
# generating input and output data
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))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# optimizing the model structure with fgpm_factory (~12 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut)
plotLOO(xm@model) # plotting the model

# building the model with the default fgpm arguments to compare
m1 <- fgpm(sIn = sIn, fIn = fIn, sOut = sOut)
plotLOO(m1) # plotting the model

# assessing the quality of the model
# in the absolute and also w.r.t. the other explored models
plotX(xm)

# checking the evolution of the algorithm
plotEvol(xm)


# improving performance with more iterations_______________________________________________
# generating input and output data
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))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# default of 15 iterations (~12 seconds)
xm15 <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut)
```

```
# increasing to 25 iterations (~20 seconds)
xm25 <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, setup = list(n.iter = 25))

# plotting both models
plotLOO(xm15@model)
plotLOO(xm25@model)


# custom solution space_____________________________________________________________________
# generating input and output data
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))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# setting up the constraints
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

# calling the funGp factory with specific constraints (~17 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, ctraints = myctr)

# verifying constraints with the log of some successfully built models
cbind(xm@log.success@sols, "Q2" = xm@log.success@fitness)


# custom heuristic parameters_______________________________________________________________
# generating input and output data
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))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# defining the 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)

# calling the funGp factory with a custom heuristic setup (~17 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, setup = mysup)

# verifying heuristic setup through the details of the Xfgpm object
```

```
unlist(xm@details$param)


# stopping condition based on time_____________________________________________________________
# generating input and output data
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))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# setting up a sufficiently large number of iterations
mysup <- list(n.iter = 2000)

# defining time budget
mytlim <- 60

# calling the funGp factory with time limit (~60 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut, setup = mysup, time.lim = mytlim)


# passing fgpm arguments through fgpm_factory_________________________________________________
# generating input and output data
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))
sOut <- fgp_BB7(sIn, fIn, n.tr)

# calling the funGp factory with custom fgpm parameters (~25 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut,
                   nugget = 0, n.starts = 3, n.presample = 12)

# NOTE: in the run above, some models crash. This happens because we set the nugget to 0
#       and some input points become duplicates when some variables are removed from
#       the model. We strongly recommend to always run fgpm_factory with at least a
#       small nugget in order to prevent loss of configurations. By default fgpm_factory
#       runs with 1e-8, which is enough in most cases.
xm@log.crashes


# 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)
```

---

fgpProj-class *S4 class for structures linked to projections in a funGp 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 and Thierry Klein

---

format4pred *Preparation of inputs for predictions based on an fgpm modelCall*

Description

This function prepared input data structures according to the active inputs specified by a "`modelCall`" object. This function is intended to easily adapt the data structures to the requirements of a specific model delivered by the model factory function fgpm_factory.

Usage

```
format4pred(sIn.pr = NULL, fIn.pr = NULL, args)
```

Arguments

`sIn.pr`	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. The `"modelCall"` object provided through args will lead the extraction of only the active scalar inputs in the model.
`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. The `"modelCall"` object provided through args will lead the extraction of only the active functional inputs in the model.
`args`	an object of class `"modelCall"`, which specifies the set of active scalar and functional inputs.

Value

An object of class `"list"`, containing the input data structures with only the active inputs specified by *args*.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

References

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

* predict for predictions based on a funGp model;

* fgpm for creation of a funGp model;

* fgpm_factory for funGp heuristic model selection.

`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 and Thierry Klein

`plotEvol` *Plot for the evolution of model selection algorithm in funGp*

Description

This method 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 bellow 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'
plotEvol(x.model, ...)
```

Arguments

`x.model`	an object of class Xfgpm containing the output of the model selection algorithm in fgpm_factory.
`...`	additional arguments affecting the plot. The following typical graphics parameters are valid entries: *xlim*, *ylim*, *xlab*, *ylab*, *main*.

Value

None.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

References

Betancourt, J., Bachoc, F., and 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

* fgpm_factory for structural optimization of funGp models;

* plotX for diagnostic plots for a fgpm_factory output and selected model.

Examples

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

# optimizing the model structure with fgpm_factory (~5 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut)

# assessing the evolution of the algorithm
plotEvol(xm)
```

---

plotEvol-generic *Plot for the evolution of model selection algorithm*

Description

This method displays the evolution of an iterative algorithm for model selection.

Arguments

x.model	an object containing the data structures returned by the model selection algorithm.
...	additional arguments affecting the plot.

Value

None.

See Also

* plotEvol for a plot on the evolution of the model selection algorithm in fgpm_factory.

Examples

```
require(funGp) # a package with a plotEvol method implemented

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

# optimizing the model structure with fgpm_factory (~5 seconds)
xm <- fgpm_factory(sIn = sIn, fIn = fIn, sOut = sOut)

# assessing the evolution of the algorihtm
plotEvol(xm)
```

---

plotLOO *Leave-one-out calibration plot for a funGp model*

Description

This method provides a diagnostic plot for the validation of a funGp Gaussian process model. 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'
plotLOO(model, ...)
```

Arguments

`model`	an object of class fgpm corresponding to the funGp model to validate.
`...`	additional arguments affecting the plot. The following typical graphics parameters are valid entries: *xlim*, *ylim*, *xlab*, *ylab*, *main*.

Value

None.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

See Also

* fgpm for the construction of funGp models;

* plotPreds for prediction plots;

* plotSims for simulation plots.

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
plotLOO(m1)
```

`plotLOO-generic` *Leave-one-out calibration plot for regression models*

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.

Arguments

`model`	a model object for which the LOO calibration plot is to be made.
`...`	additional arguments affecting the plot.

Value

None.

See Also

* plotLOO for the diagnostic plot of a funGp model.

Examples

```
require(funGp) # a package with a plotLOO method implemented

# 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
plotLOO(m1)
```

`plotPreds` *Plot for predictions of a funGp model*

Description

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

Usage

```
## S4 method for signature 'fgpm'
plotPreds(model, preds, sOut.pr = NULL, calib = TRUE, sortp = TRUE, ...)
```

Arguments

`model`	a fgpm object for which the plot is to be made.
`preds`	a list containing the predictions and confidence bands. In funGp, this argument is just the data structure delivered by the predict method.
`sOut.pr`	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.
`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.

Value

None.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

See Also

* fgpm for the construction of funGp models;

* plotLOO for model diagnostic plots;

* simulate for simulations based on a funGp model;

* plotSims 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), matrix(runif(n.pr*22), ncol = 22))
m1.preds <- predict(m1, sIn.pr = sIn.pr, fIn.pr = fIn.pr)

# plotting predictions
plotPreds(m1, preds = 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), 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
plotPreds(m1, m1.preds, sOut.pr)

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

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

---

plotPreds-generic *Plot for predictions of regression models*

Description

This method displays the predicted output values delivered by some regression model. The plot might be constituted differently, depending on the type of model at hand.

Arguments

`model`	a model object for which the plot is to be made.
`preds`	data structure containing predictions. Depending on the type of model and the data structure used, it might also contain, for instance, the confidence bands at the prediction points.
`...`	additional arguments affecting the plot.

Value

None.

See Also

* plotPreds for the predictions plot of a funGp model.

Examples

```
require(funGp) # a package with a plotPreds method implemented
# 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)

# plotting predictions
plotPreds(m1, preds = m1.preds)
```

---

plotSims *Plot for simulations from a funGp model*

Description

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

Usage

```
## S4 method for signature 'fgpm'
plotSims(model, sims, detail = "full", ...)
```

Arguments

`model`	a fgpm object for which the plot is to be made.
`sims`	a list containing the simulated output values. In funGp, this argument is just the data structure delivered by the simulate method.
`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 include the simulated values, while a 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 dropped. See simulate 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.

Value

None.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

See Also

* fgpm for the construction of funGp models;

* plotLOO for model diagnostic plots;

* predict for predictions based on a funGp model;

* plotPreds 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), matrix(runif(n.sm*22), ncol = 22))
m1.sims <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm)

# plotting light simulations
plotSims(m1, m1.sims)


# 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), matrix(runif(n.sm*22), ncol = 22))
m1.sims <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm, detail = "full")
```

```
# plotting full simulations in full mode
plotSims(m1, m1.sims)

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

---

plotSims-generic *Plot for simulations of random processes*

Description

This method displays the simulated output values delivered by some random process model. The plot might be constituted differently, depending on the type of model at hand.

Arguments

`model`	a model object for which the plot is to be made.
`sims`	data structure containing simulations Depending on the type of model and the data structure used, it might also contain, for instance, the mean and confidence bands at the simulation points.
`...`	additional arguments affecting the plot.

Value

None.

See Also

* plotSims for the simulations plot of a funGp model.

Examples

```
require(funGp) # a package with a plotSims method implemented
# 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 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), matrix(runif(n.sm*22), ncol = 22))
m1.sims <- simulate(m1, nsim = 10, sIn.sm = sIn.sm, fIn.sm = fIn.sm)
```

```
# plotting simulations
plotSims(m1, m1.sims)
```

---

plotX *Diagnostic plots for funGp factory output*

Description

This method 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 the plotLOO function. 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.

Usage

```
## S4 method for signature 'Xfgpm'
plotX(x.model, calib = TRUE, fitp = TRUE, ...)
```

Arguments

`x.model`	an object of class Xfgpm containing the output of the model selection algorithm in fgpm_factory.
`calib`	a boolean indicating whether the calibration plot of the selected model should be included in the display. Default is TRUE.
`fitp`	a boolean indicating whether scatter plot of the quality of all explored models should be included in the display. 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 fitness of explored models, the list should be called *fitp.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.

Value

None.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

References

Betancourt, J., Bachoc, F., Klein, T., and Gamboa, F. (2020), Technical Report: "Ant Colony Based Model Selection for Functional-Input Gaussian Process Regression. Ref. B3D-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

* fgpm_factory for structural optimization of funGp models;

* plotEvol for a plot on the evolution of the model selection algorithm in fgpm_factory.

Examples

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

# optimizing the model structure with fgpm_factory (~5 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
plotX(xm)

# customizing some graphical parameters
plotX(xm, calib.gpars = list(xlim = c(800,1000), ylim = c(600,1200)),
          fitp.gpars = list(main = "Relative quality", legends = FALSE))
```

---

plotX-generic *Diagnostic plot for quality-enhanced models*

Description

This method provides plots for assessing the quality of regression models whose structure have been somehow optimized for predictability.

Arguments

x.model	an object containing the model for which the quality plot is to be made.
...	additional arguments affecting the plot.

Value

None.

See Also

* plotX for the diagnostic plot of a quality-enhanced funGp model.

Examples

```
require(funGp) # a package with a plotX method implemented

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

# optimizing the model structure with fgpm_factory (~5 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
plotX(xm)
```

---

predict — *Prediction from a funGp Gaussian process model*

Description

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

Usage

```
predict(object, ...)

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

Arguments

`object`	an object of class fgpm corresponding to the funGp model that should be used to predict the output.
`...`	not used.

`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 string specifying the extent of information that should be delivered by the method, to be chosen between "light" 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. Default is "light".

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 and Thierry Klein

See Also

***** plotPreds for the predictions plot of a funGp model;

***** simulate for simulations based on a funGp model;

***** plotSims for the simulations plot of a funGp 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
plotPreds(m1, preds = 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
plotPreds(m1, 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
plotPreds(m1, preds = m1.preds)
```

---

show *Printing methods for the funGp package*

Description

This set of method enables printing of the main objects defined in the funGp package. That corresponds to fgpm, fgpKern, fgpProj, and Xfgpm objects, representing funGp models, data structures related to the kernel of the model, data structures related to projection of inputs, and structures related to structural optimization of the model, respectively.

Usage

```
## S4 method for signature 'fgpKern'
show(object)

## S4 method for signature 'fgpProj'
show(object)

## S4 method for signature 'fgpm'
show(object)

## S4 method for signature 'Xfgpm'
show(object)
```

Arguments

object	either a fgpm, fgpKern, fgpProj, or Xfgpm object.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

---

simulate *Random sampling from a funGp Gaussian process model*

Description

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

Usage

```
simulate(object, nsim = 1, seed = NULL, ...)

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

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.
`...`	not used.
`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 string specifying the extent of information that should be delivered by the method, to be chosen between "light" 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. Default is "light".

Value

An object containing the data structures linked to simulations. For *light* simulations, the output will be a matrix with 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 and Thierry Klein

See Also

* plotSims for the simulations plot of a funGp model;

* predict for predictions based on a funGp model;

* plotPreds for the predictions plot of a funGp 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
```

```
plotSims(m1, 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
plotSims(m1, m1.sims_f)

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

update — *Easy update of funGp funGp Gaussian process models*

Description

This method enables the update of data or hyperparameters of a funGp Gaussian process 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

```
update(object, ...)

## 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,
  ...
)
```

Arguments

`object`	an object of class fgpm corresponding to the funGp model to update.
`...`	not used.
`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 to 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 to 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.

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, even 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 parameters 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 be fully up to date.

Value

An object of class fgpm representing the updated funGp model.

Author(s)

José Betancourt, François Bachoc and Thierry Klein

See Also

* fgpm for creation of a funGp model;
* predict for predictions based on a funGp model;
* simulate for simulations based on a funGp 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 output
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 output
m1up3 <- update(m1, sIn.sb = sIn.sb, sOut.sb = sOut.sb, ind.sb = ind.sb) # the scalar inputs
                                                                          # and the output


# 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
```

`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. Gaussian process models are useful statistical tools in the modeling of complex input-output relationships. 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 funGp models

- **Plotters**
  plotX: diagnostic plots for a fgpm_factory optimization and the selected model
  plotEvol: plot of the evolution of the model selection algorithm in funGp

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.

Useful material

- **Manual** Gaussian Process Regression for Scalar and Functional Inputs with funGp - The in-depth tour

Author(s)

José Betancourt, François Bachoc and Thierry Klein

Index