Package 'ARCHISSUR'

June 28, 2025

Title Active Recovery of a Constrained and Hidden Set by Stepwise Uncertainty Reduction Strategy

Version 0.0.1

Description Stepwise Uncertainty Reduction criterion and algorithm for sequentially learning a Gaussian Process Classifier as described in Menz et al. (2025).

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Imports GPCsign, DiceKriging, KrigInv, future.apply, stats, TruncatedNormal, randtoolbox, rgenoud, graphics

Suggests future, DiceDesign, testthat, knitr

NeedsCompilation no

Author Morgane Menz [aut, cre], Miguel Munoz-Zuniga [aut], Delphine Sinoquet [aut], Naoual Serraji [ctb]

Maintainer Morgane Menz <morgane.menz@ifpen.fr>

Repository CRAN

Date/Publication 2025-06-28 17:00:07 UTC

Contents

archissur *Run ARCHISSUR algorithm*

Description

archissur adaptively enriches the Gaussian Process Classifier (GPC) using a learning criterion to achieve a precise approximation of the feasible area contour. This is done by iteratively adding the best learning point that minimizes future uncertainty over the feasible domain, following the Stepwise Uncertainty Reduction strategy (SUR).

Usage

```
archissur(
  design.init = NULL,
  cst.init = NULL,
  model = NULL,
  cst_function,
  lower = NULL,
  upper = NULL,
  n.ite = 10,
  seed = NULL,
  nb.integration = NULL,
  plot_2D_pn = FALSE,
  batchsize = 1,
  n_update = 1,
  gpc.options = NULL,
  optimcontrol = NULL,
  verbose = 1
)
```

Arguments

design.init	optional matrix representing the initial design of experiments (DoE). If not provided, you must provide a model of type gpcm.
cst.init	optional vector of binary observations {0,1} corresponding to the initial class labels. If not provided, it will be calculated using cst_function.
model	optional object of type gpcm to start archissur. If not provided, you must provide an initial DoE design.init.
cst_function	constraint function with binary outputs {0,1} to be learn.
lower	inputs lower bound of design.init.
upper	inputs upper bound of design.init.
n.ite	number of iterations of archissur.
seed	to fix the seed.
nb.integration	number of integration points. Default is d*1000.

plot_2D_pn	if TRUE and d = 2, plot class 1 probability map and learning points. Plots are available in '2D_plots' directory. Default is FALSE.
batchsize	number of points to be learned at each iteration. Default is 1.
n_update	number of iterations between hyperparameter updates by likelihood maximization. Default is 1.
gpc.options	list with GPC model options: covariance kernel, noise variance, number of initial points for MLE optimization, standardization of inputs, constrained latent GP mean sign.If NULL, default options are list(normalize = T, multistart = 1, covtype = "matern5_2", MeanTransform=NULL). Default is NULL.
optimcontrol	optional list of control parameters for enrichment criterion optimization. Default is NULL. The field "method" defines which optimization method is used: it can be either "multistart" (default) for an optimization with L-BFGS-B multistart, or "discrete" for an optimization over a specified discrete set, or "genoud" for an optimization using the genoud algorithm. (See details).
verbose	Level of verbosity for printing information during iterations. 0: No printing. 1: Print iteration number and best point found. 2: Print iteration number, best point found, criterion value, and model hyperparameters. Default is 1.

Details

If the field "method" is set to "genoud", one can set some parameters of this algorithm: pop.size (default: 50*d),* max.generations *(10*d), wait.generations (2), BFGSburnin (2) and the mutations P1, P2, up to P9 (see genoud). Numbers into brackets are the default values. If the field method is set to "discrete", one can set the field optim.points: p * d matrix corresponding to the p points where the criterion will be evaluated. If nothing is specified, 100*d points are chosen randomly. Finally, one can control the field optim.option in order to decide how to optimize the sampling criterion. If optim.option is set to 2 (default), batchsize sequential optimizations in dimension d are performed to find the optimum. If optim.option is set to 1, only one optimization in dimension batchsize*d is performed. This option is only available with "genoud". This option might provide more global and accurate solutions, but is a lot more expensive.

Value

A list containing:

Xf	DoE
f	binary observations corresponding to the class labels.
alpha	a scalar representing the Vorob'ev threshold.
vorob_expect	Vorob'ev expectation.
vorob_dev	current Vorob'ev deviation.
model	an object of class gpcm containing the GPC model at last iteration.

An '.Rds' file model_gp.Rds containing an object of class gpcm corresponding to current GPC model. The class 1 probability map. Plots are available in '2D_plots' directory.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Bachoc, F., Helbert, C. & Picheny, V. Gaussian process optimization with failures: classification and convergence proof. *J Glob Optim* **78**, 483–506 (2020). doi:10.1007/s10898020009200.

Examples

```
#------------------------------------------------------------------
#------------------------ archissur -------------------------------
#------------------------------------------------------------------

#  20-points DoE, and the corresponding response
d <- 2
nb_PX <- 20
x <- matrix(c(0.205293785978832, 0.0159983370750337,
              0.684774733109666, 0.125251417595962,
              0.787208786290006, 0.700475706055049,
              0.480507717105934, 0.359730889653793,
              0.543665267336735, 0.565974761807069,
              0.303412043992361, 0.471502352650857,
              0.839505250127309, 0.504914690245002,
              0.573294917143728, 0.784444726564573,
              0.291681289223421, 0.255053812451938,
              0.87233450888786, 0.947168337730927,
              0.648262257638515, 0.973264712407035,
              0.421877310273815, 0.0686662506387988,
              0.190976166753807, 0.810964668176754,
              0.918527262507395, 0.161973686467513,
              0.0188128700859558, 0.43522031347403,
              0.99902788789426, 0.655561821513544,
              0.741113863862512, 0.321050086076934,
              0.112003007565305, 0.616551317575545,
              0.383511473487687, 0.886611679106771,
              0.0749211435982952, 0.205805968972305),
            byrow = TRUE, ncol = d)

require(DiceKriging)
cst_function <- function(z){
  fx <- apply(z, 1, branin)
  f <- ifelse(fx < 14, 0, 1)
  return(f)}

## constraint function
s <- cst_function(x)

# archissur parameters

design.init <- x
cst.init <- s
```

```
n.ite <- 2
n_update <- 5
lower <- rep(0,d)
upper <- rep(1,d)
### GPC model options
gpc.options <- list()
gpc.options$noise.var <- 1e-6
gpc.options$multistart <- 1

res <-  archissur(design.init = design.init, cst.init = cst.init,
                  cst_function = cst_function, lower = lower, upper = upper,
                  n.ite = n.ite, n_update = n_update,  gpc.options = gpc.options)
unlink('model_gp.Rds')
```

`computeQuickgpccov` *Quick computation of Gaussian Process Classification (GPC) covariances*

Description

Computes conditional covariance matrix between some new points and many integration points.

Usage

```
computeQuickgpccov(object, integration.points, X.new, precalc.data, c.newdata)
```

Arguments

`object`	an object of class gpcm.
`integration.points`	p*d matrix of fixed integration points in the X space.
`X.new`	q*d matrix of new points.
`precalc.data`	list containing precalculated data. This list can be generated using the `precomputeUpdateData` function.
`c.newdata`	the (unconditional) covariance between `X.new` and the design points.

Value

Conditional covariance matrix between `integration.points` and `X.new`.

Author(s)

Morgane MENZ, Delphine SINOQUET, Miguel MUNOZ-ZUNIGA. Contributors: Naoual SERRAJI.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Bachoc, F., Helbert, C. & Picheny, V. Gaussian process optimization with failures: classification and convergence proof. *J Glob Optim* **78**, 483–506 (2020). doi:10.1007/s10898020009200.

Examples

```
#------------------------------------------------------------------
#------------------- computeQuickgpccov ---------------------------
#------------------------------------------------------------------

## 20-points DoE, and the corresponding response
d <- 2
nb_PX <- 20
x <- matrix(c(0.205293785978832, 0.0159983370750337,
              0.684774733109666, 0.125251417595962,
              0.787208786290006, 0.700475706055049,
              0.480507717105934, 0.359730889653793,
              0.543665267336735, 0.565974761807069,
              0.303412043992361, 0.471502352650857,
              0.839505250127309, 0.504914690245002,
              0.573294917143728, 0.784444726564573,
              0.291681289223421, 0.255053812451938,
              0.87233450888786, 0.947168337730927,
              0.648262257638515, 0.973264712407035,
              0.421877310273815, 0.0686662506387988,
              0.190976166753807, 0.810964668176754,
              0.918527262507395, 0.161973686467513,
              0.0188128700859558, 0.43522031347403,
              0.99902788789426, 0.655561821513544,
              0.741113863862512, 0.321050086076934,
              0.112003007565305, 0.616551317575545,
              0.383511473487687, 0.886611679106771,
              0.0749211435982952, 0.205805968972305),
            byrow = TRUE, ncol = d)
require(DiceKriging)
fx <- apply(x, 1, branin)
f <- ifelse(fx < 14, -1, 1)
Xf <- as.matrix(x)

require(future) # load future package for parallelization while building gpcm model
plan(multisession) # activate parallel calculations (with available cores automatic detection)

## gpcm object
require(GPCsign)
model <- gpcm(f, Xf, coef.m = -1.25, coef.cov = c(1.17,0.89))

## prediction at X.new
X.new <-  matrix(c(0.1,0.2),ncol=2,byrow=TRUE)
```

```
gpc <- predict(object = model, newdata = X.new)
c.newdata <- gpc$c

## prediction at integration points
require(randtoolbox)
nb.integration <- d * 100
integration.points <- sobol(n = nb.integration, dim = d, scrambling = 0)
gpc_int <- predict(object = model, newdata = integration.points)

precalc.data <- list(c.K = crossprod(gpc_int$c , model@invK))
c.newdata <- gpc$c

integration.points <- scale(x = integration.points, center = model@X.mean, scale = model@X.std)
kn <-  computeQuickgpccov(object = model, integration.points = integration.points,
                          X.new = X.new, precalc.data = precalc.data,
                          c.newdata = c.newdata )
plan(sequential) # deactivate parallel calculations: back to sequential mode
```

`computeVorobTerms` *Compute conditional probabilities*

Description

`computeVorobTerms` compute the future uncertainty.

Usage

```
computeVorobTerms(
  i,
  object,
  intpoints.oldmean,
  krig2,
  sk.new,
  alpha,
  gpc,
  X.new,
  seed = NULL
)
```

Arguments

`i`	the index for which conditional probabilities are computed. It ranges from 1 to $2^{nrow(X.new)}$ which represents all possible combinations.
`object`	an object of class `gpcm`.
`intpoints.oldmean`	vectors of size `ncol(integration.points)` corresponding to the mean at the integration points before adding the batchsize points `x` to the design of experiments.

krig2	a list containing the output of the function predict_update_gpc_parallel.
sk.new	conditional standard deviations vector at the points intpoints.
alpha	a scalar representing the Vorob'ev threshold.
gpc	a list containing the output of the predict function predict at X.new.
X.new	input vector of size batchsize*d at which one wants to evaluate the criterion.
seed	to fix the seed.

Value

a list with:

term2	Vector equal to $p_{n+q}p(y)$ where p_{n+q} is the updated feasibility probability at integration.points and $p(y)$ is the probability of outcomes y at new points X.new
term1	Vector equal to the product of term2 with the indicator of $p_{n+q} > \alpha$ where α is the Vorob'ev threshold
term3	Vector equal to the product of p_y with the indicator of $p_{n+q} < \alpha$

Author(s)

Morgane MENZ, Delphine SINOQUET, Miguel MUNOZ-ZUNIGA. Contributors: Naoual SERRAJI.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Examples

```
#------------------------------------------------------------------
#-------------------------- computeVorobTerms ---------------------------
#------------------------------------------------------------------

## 20-points DoE, and the corresponding response
d <- 2
nb_PX <- 20
x <- matrix(c(0.205293785978832, 0.0159983370750337,
              0.684774733109666, 0.125251417595962,
              0.787208786290006, 0.700475706055049,
              0.480507717105934, 0.359730889653793,
              0.543665267336735, 0.565974761807069,
              0.303412043992361, 0.471502352650857,
              0.839505250127309, 0.504914690245002,
              0.573294917143728, 0.784444726564573,
              0.291681289223421, 0.255053812451938,
              0.87233450888786, 0.947168337730927,
              0.648262257638515, 0.973264712407035,
              0.421877310273815, 0.0686662506387988,
```

```
              0.190976166753807, 0.810964668176754,
              0.918527262507395, 0.161973686467513,
              0.0188128700859558, 0.43522031347403,
              0.99902788789426, 0.655561821513544,
              0.741113863862512, 0.321050086076934,
              0.112003007565305, 0.616551317575545,
              0.383511473487687, 0.886611679106771,
              0.0749211435982952, 0.205805968972305),
            byrow = TRUE, ncol = d)
require(DiceKriging)
fx <- apply(x, 1, branin)
f <- ifelse(fx < 14, -1, 1)
Xf <- as.matrix(x)

require(future)  # load future package for parallelization
plan(multisession) # activate parallel calculations (with available cores automatic detection)

## gpcm object
require(GPCsign)
model <- gpcm(f, Xf, coef.m = -1.25, coef.cov = c(1.17,0.89))

##
n.grid <- 20
x.grid <- seq(0,1,length=n.grid)
newdata <- expand.grid(x.grid,x.grid)
newdata <- as.matrix(newdata)
pred1 <- predict(object=model,newdata=newdata)
precalc.data <- list()
precalc.data$c.K <- crossprod(pred1$c, model@invK)
newdata.oldsd <- sqrt(pred1$Zsimu_var)

# new points added
new.x <- matrix(c(0.1,0.2),ncol=2,byrow=TRUE)

# predicion at new points
pred2 <- predict(object=model,newdata=new.x)
Sigma.r <- pred2$cov

newdata <- scale(x = newdata, center = model@X.mean, scale = model@X.std)
new.x <- scale(x = new.x, center = model@X.mean, scale = model@X.std)
kn <- computeQuickgpccov(object = model,
                         integration.points = newdata,
                         X.new = new.x,
                         precalc.data = precalc.data,
                         c.newdata = pred2$c)

updated.predictions <- predict_update_gpc_parallel(Sigma.r = Sigma.r,
                                                   newdata.oldsd = newdata.oldsd,
                                                   kn = kn)
# parameters for comp_term function
i <- 1
intpoints.oldmean <- pred1$Zsimu_mean
sk.new <- updated.predictions$sd
```

```
pn <- pred1$prob
alpha <- KrigInv::vorob_threshold(pn)

result <- computeVorobTerms(i = i, object = model,
                    intpoints.oldmean = intpoints.oldmean,
                    krig2 = updated.predictions,
                    sk.new = sk.new,
                    alpha = alpha,
                    gpc = pred2,
                    X.new = new.x)
plan(sequential) # deactivate parallel calculations: back to sequential mode
```

`max_vorob_parallel_gpc` *Minimizer of the parallel vorob criterion*

Description

Minimization of the Vorob'ev criterion for a batch of candidate sampling points.

Usage

```
max_vorob_parallel_gpc(
  lower,
  upper,
  optimcontrol = NULL,
  batchsize,
  integration.param,
  object,
  new.noise.var = 0,
  seed = NULL
)
```

Arguments

`lower`	vector containing the lower bounds of the design space.
`upper`	vector containing the upper bounds of the design space.
`optimcontrol`	optional list of control parameters for the optimization of the sampling criterion. The field `"method"` defines which optimization method is used: it can be either `"multistart"` (default) for an optimization with `L-BFGS-B` multistart, or `"discrete"` for an optimization over a specified discrete set, or `"genoud"` for an optimization using the genoud algorithm. (See details)
`batchsize`	number of points to sample simultaneously. The sampling criterion will return batchsize points at a time for sampling.

`integration.param`
: optional list of control parameter for the computation of integrals, containing the fields `integration.points`: a p*d matrix corresponding to p integration points. `integration.weights`: a vector of size p corresponding to the weights of these integration points, and `alpha`: the Vorob'ev threshold.

`object`
: an object of class `gpcm` .

`new.noise.var`
: optional scalar value of the noise variance of the new observations. Default is 0.

`seed`
: to fix the seed.

Details

If the field `method` is set to `"genoud"`, one can set some parameters of this algorithm: `pop.size` (default: 50*d),* `max.generations` *(10*d), `wait.generations` (2), `BFGSburnin` (2) and the mutations `P1`, `P2`, up to `P9` (see `genoud`). Numbers into brackets are the default values. If the field `method` is set to `"discrete"`, one can set the field `optim.points`: p * d matrix corresponding to the p points where the criterion will be evaluated. If nothing is specified, 100*d points are chosen randomly. Finally, one can control the field `optim.option` in order to decide how to optimize the sampling criterion. If `optim.option` is set to 2 (default), `batchsize` sequential optimizations in dimension d are performed to find the optimum. If `optim.option` is set to 1, only one optimization in dimension `batchsize*d` is performed. This option is only available with `"genoud"`. This option might provide more global and accurate solutions, but is a lot more expensive.

Value

a list with components:

`par`
: the best set of parameters found.

`value`
: the value of the Vorob'ev criterion at `par`.

`allvalues`
: if an optimization on a discrete set of points is chosen, the value of the criterion at all these points.

`current.vorob`
: current vorob'ev deviation.

`alpha`
: the Vorob'ev thresold.

Author(s)

Morgane MENZ, Delphine SINOQUET, Miguel MUNOZ-ZUNIGA. Contributors: Naoual SERRAJI.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Chevalier, C. Fast uncertainty reduction strategies relying on Gaussian process models PhD Thesis. University of Bern (2013).

Bachoc, F., Helbert, C. & Picheny, V. Gaussian process optimization with failures: classification and convergence proof. *J Glob Optim* **78**, 483–506 (2020). doi:10.1007/s10898020009200.

Examples

```
#------------------------------------------------------------------
#------------------- max_vorob_optim_parallel_gpc------------------
#------------------------------------------------------------------

## 20-points DoE, and the corresponding response
d <- 2
nb_PX <- 20
x <- matrix(c(0.205293785978832, 0.0159983370750337,
              0.684774733109666, 0.125251417595962,
              0.787208786290006, 0.700475706055049,
              0.480507717105934, 0.359730889653793,
              0.543665267336735, 0.565974761807069,
              0.303412043992361, 0.471502352650857,
              0.839505250127309, 0.504914690245002,
              0.573294917143728, 0.784444726564573,
              0.291681289223421, 0.255053812451938,
              0.87233450888786, 0.947168337730927,
              0.648262257638515, 0.973264712407035,
              0.421877310273815, 0.0686662506387988,
              0.190976166753807, 0.810964668176754,
              0.918527262507395, 0.161973686467513,
              0.0188128700859558, 0.43522031347403,
              0.99902788789426, 0.655561821513544,
              0.741113863862512, 0.321050086076934,
              0.112003007565305, 0.616551317575545,
              0.383511473487687, 0.886611679106771,
              0.0749211435982952, 0.205805968972305),
            byrow = TRUE, ncol = d)
require(DiceKriging)
fx <- apply(x, 1, branin)
f <- ifelse(fx < 14, -1, 1)
Xf <- as.matrix(x)

require(future) # load future package for parallelization
plan(multisession)  # activate parallel calculations (with available cores automatic detection)
## gpcm object
require(GPCsign)
model <- gpcm(f, Xf, coef.m = -1.25, coef.cov = c(1.17,0.89))

# parameters for max_vorob_parallel_gpc function
lower <- rep(0,d)
upper <- rep(1,d)
batchsize = 1
integration.param <- list()
require(randtoolbox)
nb.integration <- d*100
integration.points <- sobol(n = nb.integration, dim = d, scrambling = 0)
integration.param$integration.points <- rep(upper-lower,each=nb.integration) *
   matrix(integration.points, nrow=nb.integration) +
   matrix(rep(lower,each=nb.integration), nrow=nb.integration)
integration.param$integration.weights <- NULL
```

```
integration.param$alpha <- NULL
optimcontrol <- list()
optimcontrol$method <- "multistart"
crit <- max_vorob_parallel_gpc(lower = lower, upper = upper,
                               batchsize = batchsize,
                               integration.param = integration.param,
                               object = model, optimcontrol = optimcontrol, seed=1)
plan(sequential) # deactivate parallel calculations: back to sequential mode
```

`precomputeUpdateData` *Useful precomputations to quickly update GPC mean and variance*

Description

Useful precomputations to quickly update GPC mean and variance

Usage

```
precomputeUpdateData(model, integration.points)
```

Arguments

`model`	an object of class gpcm.
`integration.points`	p*d matrix of fixed integration points in the X space.

Value

a list with components:

`c.K`	Vector equal to $t(c)K^{-1}$ where K^{-1} is the inverse covariance matrix `invK` returned by `object` and `c` the unconditional covariance matrix between `newdata` and design points `Xf`
`lambda.intpoints`	Vector equal to $K^{-1}c$ where K^{-1} is the inverse covariance matrix `invK` returned by `object` and `c` the unconditional covariance matrix between `newdata` and design points `Xf`
`pn.intpoints`	the (averaged) probability of class 1 at `integration.points`.
`intpoints.oldmean`	conditional mean matrix of the latent GP at `integration.points`
`intpoints.oldmean`	conditional variance vector of the latent GP at `integration.points`

Author(s)

Morgane MENZ, Delphine SINOQUET, Miguel MUNOZ-ZUNIGA. Contributors: Naoual SERRAJI.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Bachoc, F., Helbert, C. & Picheny, V. Gaussian process optimization with failures: classification and convergence proof. *J Glob Optim* **78**, 483–506 (2020). doi:10.1007/s10898020009200.

Examples

```
#-------------------------------------------------------------------
#------------------- precomputeUpdateData ---------------------------
#-------------------------------------------------------------------

## 20-points DoE, and the corresponding response
d <- 2
nb_PX <- 20
x <- matrix(c(0.205293785978832, 0.0159983370750337,
              0.684774733109666, 0.125251417595962,
              0.787208786290006, 0.700475706055049,
              0.480507717105934, 0.359730889653793,
              0.543665267336735, 0.565974761807069,
              0.303412043992361, 0.471502352650857,
              0.839505250127309, 0.504914690245002,
              0.573294917143728, 0.784444726564573,
              0.291681289223421, 0.255053812451938,
              0.87233450888786, 0.947168337730927,
              0.648262257638515, 0.973264712407035,
              0.421877310273815, 0.0686662506387988,
              0.190976166753807, 0.810964668176754,
              0.918527262507395, 0.161973686467513,
              0.0188128700859558, 0.43522031347403,
              0.99902788789426, 0.655561821513544,
              0.741113863862512, 0.321050086076934,
              0.112003007565305, 0.616551317575545,
              0.383511473487687, 0.886611679106771,
              0.0749211435982952, 0.205805968972305),
            byrow = TRUE, ncol = d)
require(DiceKriging)
fx <- apply(x, 1, branin)
f <- ifelse(fx < 14, -1, 1)
Xf <- as.matrix(x)


## gpcm object
require(GPCsign)
require(randtoolbox)
model <- gpcm(f, Xf, coef.m = -1.25, coef.cov = c(1.17,0.89))

nb.integration <- d * 100
integration.points <- sobol(n = nb.integration, dim = d, scrambling = 0)
integration.points <- scale(x = integration.points, center = model@X.mean, scale = model@X.std)
```

```
precalc.data <- precomputeUpdateData(model = model, integration.points = integration.points)
```

`predict_update_gpc_parallel` *Quick update of conditional variances when one or many new points are added to the DoE.*

Description

The functions uses Gaussian Process Classification (GPC) update formulas to quickly compute conditional variances at points newdata, when `r` new points `newX` are added.

Usage

```
predict_update_gpc_parallel(Sigma.r, newdata.oldsd, kn)
```

Arguments

`Sigma.r`	an r*r conditional covariance matrix at `r` points `newX`(x_(n+1),...,x_(n+r)). It represents the covariance matrix `cov` obtained through the `predict` function at `newX`.
`newdata.oldsd`	conditional standard deviations vector at the points `newdata`, (before adding x_(n+1),...,x_(n+r)).
`kn`	conditional covariances between the points `newdata` and the `r` points `newX`.

Value

a list with:

`sd`	updated conditional standard deviation at points `newdata`.
`lambda`	new GPC weight of x_(n+1),...,x_(n+r) for the prediction at points `newdata`.

Author(s)

Morgane MENZ, Delphine SINOQUET, Miguel MUNOZ-ZUNIGA. Contributors: Naoual SERRAJI.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Chevalier, C., Ginsbourger, D. (2014). Corrected Kriging update formulae for batch-sequential data assimilation, in Pardo-Iguzquiza, E., et al. (Eds.) *Mathematics of Planet Earth*, pp 119-122

Examples

```
#------------------------------------------------------------------
#------------------- predict_update_gpc_parallel ------------------
#------------------------------------------------------------------

## 20-points DoE, and the corresponding response
d <- 2
nb_PX <- 20
x <- matrix(c(0.205293785978832, 0.0159983370750337,
              0.684774733109666, 0.125251417595962,
              0.787208786290006, 0.700475706055049,
              0.480507717105934, 0.359730889653793,
              0.543665267336735, 0.565974761807069,
              0.303412043992361, 0.471502352650857,
              0.839505250127309, 0.504914690245002,
              0.573294917143728, 0.784444726564573,
              0.291681289223421, 0.255053812451938,
              0.87233450888786, 0.947168337730927,
              0.648262257638515, 0.973264712407035,
              0.421877310273815, 0.0686662506387988,
              0.190976166753807, 0.810964668176754,
              0.918527262507395, 0.161973686467513,
              0.0188128700859558, 0.43522031347403,
              0.99902788789426, 0.655561821513544,
              0.741113863862512, 0.321050086076934,
              0.112003007565305, 0.616551317575545,
              0.383511473487687, 0.886611679106771,
              0.0749211435982952, 0.205805968972305),
            byrow = TRUE, ncol = d)
require(DiceKriging)
fx <- apply(x, 1, branin)
f <- ifelse(fx < 14, -1, 1)
Xf <- as.matrix(x)

require(future)
plan(multisession)

## gpcm object
require(GPCsign)
model <- gpcm(f, Xf, coef.m = -1.25, coef.cov = c(1.17,0.89))

##
n.grid <- 20
x.grid <- seq(0,1,length=n.grid)
newdata <- expand.grid(x.grid,x.grid)
newdata <- as.matrix(newdata)
pred1 <- predict(object=model,newdata=newdata)
precalc.data <- list()
precalc.data$c.K <- crossprod(pred1$c, model@invK)
newdata.oldsd <- sqrt(pred1$Zsimu_var)

# new points added
```

```
new.x <- matrix(c(0.1,0.2),ncol=2)

# predicion at new points
pred2 <- predict(object=model,newdata=new.x)
Sigma.r <- pred2$cov

newdata <- scale(x = newdata, center = model@X.mean, scale = model@X.std)
new.x <- scale(x = new.x, center = model@X.mean, scale = model@X.std)
kn <- computeQuickgpccov(object = model,
                         integration.points = newdata,
                         X.new = new.x,
                         precalc.data = precalc.data,
                         c.newdata = pred2$c)

updated.predictions <- predict_update_gpc_parallel(Sigma.r = Sigma.r,
                                                   newdata.oldsd = newdata.oldsd,
                                                   kn = kn)
plan(sequential)
```

---

vorob_optim_parallel2_gpc *Parallel Vorob'ev criterion*

Description

Evaluation of the Vorob'ev criterion for candidate points `x`, assuming that some other points are also going to be evaluated. To be used in optimization routines, like in `max_vorob_parallel_gpc`. To avoid numerical instabilities, the new points are evaluated only if they are not too close to an existing observation, or if there is some observation noise. The criterion is the integral of the posterior Vorob'ev uncertainty.

Usage

```
vorob_optim_parallel2_gpc(
  x,
  other.points,
  integration.points,
  integration.weights = NULL,
  intpoints.oldmean,
  intpoints.oldsd,
  precalc.data,
  object,
  new.noise.var = NULL,
  batchsize,
  alpha,
  current.vorob,
  seed = NULL
)
```

Arguments

x	input vector of size d at which one wants to evaluate the criterion.
other.points	vector giving the other batchsize-1 points at which one wants to evaluate the criterion.
integration.points	p*d matrix of points for numerical integration in the design space.
integration.weights	vector of size p corresponding to the weights of these integration points.
intpoints.oldmean	vector of size p corresponding to the latent GP mean at the integration points before adding x to the design of experiments.
intpoints.oldsd	vector of size p corresponding to the latent GP standard deviation at the integration points before adding x to the design of experiments.
precalc.data	list containing precalculated data. This list can be generated using the precomputeUpdateData function.
object	object of class gpcm.
new.noise.var	optional scalar value of the noise variance of the new observations.
batchsize	number of points to sample simultaneously. The sampling criterion will return batchsize points at a time for sampling.
alpha	a scalar representing the Vorob'ev threshold.
current.vorob	current value of the vorob criterion (before adding new observations).
seed	to fix the seed.

Value

Parallel Vorob'ev value

Author(s)

Morgane MENZ, Delphine SINOQUET, Miguel MUNOZ-ZUNIGA. Contributors: Naoual SERRAJI.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Chevalier, C. Fast uncertainty reduction strategies relying on Gaussian process models PhD Thesis. University of Bern (2013).

Bachoc, F., Helbert, C. & Picheny, V. Gaussian process optimization with failures: classification and convergence proof. *J Glob Optim* **78**, 483–506 (2020). doi:10.1007/s10898020009200.

See Also

max_vorob_parallel_gpc()

vorob_optim_parallel_gpc *Parallel Vorob'ev criterion*

Description

Evaluation of the parallel Vorob'ev criterion for some candidate points. To be used in optimization routines, like in max_vorob_parallel_gpc. To avoid numerical instabilities, the new points are evaluated only if they are not too close to an existing observation, or if there is some observation noise. The criterion is the integral of the posterior Vorob'ev uncertainty.

Usage

```
vorob_optim_parallel_gpc(
  x,
  integration.points,
  integration.weights = NULL,
  intpoints.oldmean,
  intpoints.oldsd,
  precalc.data,
  object,
  new.noise.var = NULL,
  batchsize,
  alpha,
  current.vorob,
  seed = NULL
)
```

Arguments

x	input vector of size batchsize*d at which one wants to evaluate the criterion. This argument is NOT a matrix.
integration.points	matrix of points for numerical integration in the design space.
integration.weights	vector of size ncol(integration.points) corresponding to the weights of these integration points.
intpoints.oldmean	(see below).
intpoints.oldsd	vectors of size ncol(integration.points) corresponding to the mean and standard deviation at the integration points before adding the batchsize points x to the design of experiments.
precalc.data	list containing precalculated data. This list can be generated using the precomputeUpdateData function.
object	object of class gpcm.

new.noise.var	optional scalar value of the noise variance for the new observations.
batchsize	number of points to sample simultaneously. The sampling criterion will return batchsize points at a time for sampling.
alpha	a scalar representing the Vorob'ev threshold
current.vorob	current value of the vorob criterion (before adding new observations).
seed	to fix the seed.

Value

Parallel vorob value

Author(s)

Morgane MENZ, Delphine SINOQUET, Miguel MUNOZ-ZUNIGA. Contributors: Naoual SERRAJI.

References

Menz, M., Munoz-Zuniga, M., Sinoquet, D. Estimation of simulation failure set with active learning based on Gaussian Process classifiers and random set theory (2023). https://hal.science/hal-03848238.

Chevalier, C. Fast uncertainty reduction strategies relying on Gaussian process models PhD Thesis. University of Bern (2013).

El Amri, M.R. Analyse d'incertitudes et de robustesse pour les modèles à entrées et sorties fonctionnelles. Theses, Université Grenoble Alpes (2019), https://theses.hal.science/tel-02433324.

El Amri, M.R., Helbert, C., Munoz-Zuniga, M. Feasible set estimation under functional uncertainty by gaussian process modelling (2023). 455:133,893. doi:10.1016/j.physd.2023.133893.

Bachoc, F., Helbert, C. & Picheny, V. Gaussian process optimization with failures: classification and convergence proof. *J Glob Optim* **78**, 483–506 (2020). doi:10.1007/s10898020009200.

Examples

```
#------------------------------------------------------------------
#------------------vorob_optim_parallel_gpc------------------------
#------------------------------------------------------------------

## 20-points DoE, and the corresponding response
d <- 2
nb_PX <- 20
x_ <- matrix(c(0.205293785978832, 0.0159983370750337,
               0.684774733109666, 0.125251417595962,
               0.787208786290006, 0.700475706055049,
               0.480507717105934, 0.359730889653793,
               0.543665267336735, 0.565974761807069,
               0.303412043992361, 0.471502352650857,
               0.839505250127309, 0.504914690245002,
               0.573294917143728, 0.784444726564573,
               0.291681289223421, 0.255053812451938,
```

```
              0.87233450888786, 0.947168337730927,
              0.648262257638515, 0.973264712407035,
              0.421877310273815, 0.0686662506387988,
              0.190976166753807, 0.810964668176754,
              0.918527262507395, 0.161973686467513,
              0.0188128700859558, 0.43522031347403,
              0.99902788789426, 0.655561821513544,
              0.741113863862512, 0.321050086076934,
              0.112003007565305, 0.616551317575545,
              0.383511473487687, 0.886611679106771,
              0.0749211435982952, 0.205805968972305),
            byrow = TRUE, ncol = d)
require(DiceKriging)
fx <- apply(x_, 1, branin)
f <- ifelse(fx < 14, -1, 1)
Xf <- as.matrix(x_)

require(future)
plan(multisession)
## gpcm object
require(GPCsign)
model <- gpcm(f, Xf, coef.m = -1.25, coef.cov = c(1.17,0.89))

## parameteres for vorob_optim_parallel_gpc function
batchsize <- 1
x <- matrix(c(0.1,0.2),ncol=2,byrow=TRUE)
require(randtoolbox)
nb.integration <- d * 1000
integration.points <- sobol(n = nb.integration, dim = d, scrambling = 0)
integration.points <- scale(x = integration.points, center = model@X.mean, scale = model@X.std)
precalc.data <- precomputeUpdateData(model=model, integration.points=integration.points)
intpoints.oldmean <- precalc.data$intpoints.oldmean
intpoints.oldsd <- precalc.data$intpoints.oldsd

pn <- precalc.data$pn

require(KrigInv)
alpha <- KrigInv::vorob_threshold(pn)
pn_bigger_than_alpha <- (pn>alpha)+0
pn_lower_than_alpha <- 1-pn_bigger_than_alpha
penalisation <- 1
current.vorob <- mean(pn*pn_lower_than_alpha + penalisation*(1-pn)*pn_bigger_than_alpha)

x <- scale(x = x, center = model@X.mean, scale = model@X.std)
criter <- vorob_optim_parallel_gpc(x = x, integration.points = integration.points,
                                   intpoints.oldmean = intpoints.oldmean,
                                   intpoints.oldsd = intpoints.oldsd,
                                   precalc.data = precalc.data,
                                   object = model,
                                   batchsize = batchsize,
                                   alpha = alpha,
                                   current.vorob = current.vorob)
plan(sequential)
```

Index