Package 'KrigInv'

January 27, 2015

Type Package

Title Kriging-based Inversion for Deterministic and Noisy Computer Experiments

Version 1.3.1

Date 2014-12-22

Author Clement Chevalier, Victor Picheny and David Ginsbourger

Maintainer Clement Chevalier <clement.chevalier@math.uzh.ch>

Depends DiceKriging, pbivnorm, rgenoud, randtoolbox

Imports

Suggests

Description Criteria and algorithms for sequentially estimating level sets of a multivariate numerical function, possibly observed with noise.

URL http://www.sciencedirect.com/science/article/pii/S0167947313001060, http://www.clementchevalier.com

License GPL-3

LazyLoad yes

Repository CRAN

Date/Publication 2014-12-23 00:30:22

NeedsCompilation no

R topics documented:

`KrigInv-package` *Kriging-based inversion of deterministic and stochastic computer codes*

Description

Sequential algorithms based on Kriging for computer experiments, meant to explore the subset of input parameters corresponding to a prescribed level of the output.

Details

Package:	KrigInv
Type:	Package
Version:	1.3
Date:	2012-08-01
License:	GPL version 3
LazyLoad:	yes

Note

A first prototype of this package was originally developed by D. Ginsbourger in the frame of a collaboration with IRSN (Institut de Radioprotection et de Surete Nucleaire), acting through Yann Richet.

The three main authors thank IRSN for sponsoring open source research, and allowing them to spread the present package and publish it on CRAN.

They also would like to warmly thank Yann Richet for numerous discussions concerning this package, and more!

Package `rgenoud` >=5.3.3. is recommended.

Important function (key to all proposed methods):

`EGI` Sequential Kriging-based inversion

Author(s)

C. Chevalier (IMSV, University of Bern, Switzerland and IRSN)

V. Picheny (Ecole Centrale Paris)

D. Ginsbourger (IMSV, University of Bern, Switzerland)

with contributions from Yann Richet (IRSN)

Maintainer: C. Chevalier (clement.chevalier@stat.unibe.ch)

References

Chevalier C., Picheny V., Ginsbourger D. (2012), *The KrigInv package: An efficient and user-friendly R implementation of Kriging-based inversion algorithms* , http://hal.archives-ouvertes.fr/hal-00713537/

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Picheny V., Improving accuracy and compensating for uncertainty in surrogate modeling, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

Vazquez, E., Bect, J.: A sequential Bayesian algorithm to estimate a probability of failure. In: Proceedings of the 15th IFAC Symposium on System Identification, (SYSID 2009), Saint-Malo, France (2009)

Bichon, B.J., Eldred, M.S., Swiler, L.P., Mahadevan, S., McFarland, J.M.: Efficient global reliability analysis for nonlinear implicit performance functions. AIAA Journal 46 (10), 2459-2468 (2008)

Ranjan, P., Bingham, D., Michailidis, G.: Sequential experiment design for contour estimation from complex computer codes. Technometrics 50(4), 527-541 (2008)

Rasmussen C.E., Williams C.K.I. (2006), *Gaussian Processes for Machine Learning*, the MIT Press, www.GaussianProcess.org/gpml

---

`bichon_optim` *Bichon et al.'s Expected Feasibility criterion*

Description

Evaluation of Bichon's Expected Feasibility criterion. To be used in optimization routines, like in `max_infill_criterion`.

Usage

```
bichon_optim(x, model, T, method.param = NULL)
```

Arguments

`x`	Input vector at which one wants to evaluate the criterion. This argument can be either a vector of size d (for an evaluation at a single point) or a p*d matrix (for p simultaneous evaluations of the criterion at p different points).
`model`	An object of class `km` (Kriging model).
`T`	Target value (scalar). The sampling algorithm and the underlying kriging model aim to find the points below (resp. over) T.
`method.param`	Scalar tolerance around the target T. If not provided, default value used is 1.

Value

Bichon EF criterion. When the argument `x` is a vector, the function returns a scalar. When the argument `x` is a p*d matrix, the function returns a vector of size p.

Author(s)

V. Picheny (CERFACS, Toulouse, France)

D. Ginsbourger (IMSV, University of Bern, Switzerland)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

Bichon, B.J., Eldred, M.S., Swiler, L.P., Mahadevan, S., McFarland, J.M.: Efficient global reliability analysis for nonlinear implicit performance functions. AIAA Journal 46 (10), 2459-2468 (2008)

See Also

EGI, max_infill_criterion

Examples

```
#bichon_optim

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

x <- c(0.5,0.4)#one evaluation of the bichon criterion
bichon_optim(x=x,T=T,model=model)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
bichon.grid <- bichon_optim(x=x,T=T,model=model)
z.grid <- matrix(bichon.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,25,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
```

```
i.best <- which.max(bichon.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of Bichon criterion (black) and of f(x)=T (blue)")
```

`computeAuxVariables_noChol` *Auxiliary variables for kriging*

Description

Computes or updates some auxiliary variables used for kriging (see below). This function is a copy of the computeAuxVariables function from the DiceKriging package, except that the calculation of the Cholesky decomposition is not performed, for cpu time savings.

Usage

```
computeAuxVariables_noChol(model)
```

Arguments

`model`	An object of class `km` with missing or non updated fields.

Value

An updated `km` objet, where the changes concern the following field:

`z`	Vector equal to t(T)^(-1)*(y - F*beta), where y, F and Beta are respectively the vector of response, the experimental matrix and the trend coefficients specified in `model`.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne

References

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

See Also

`chol`, `backsolve`.

Examples

```
#computeAuxVariables_noChol
set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

model2 <- computeAuxVariables_noChol(model)
#in this example model2 and model are the same!
```

`computeAuxVariables_update` *Auxiliary variables in an update_km procedure*

Description

Function similar to the `computeAuxVariables` of the DiceKriging package, with a quicker implementation.

Usage

```
computeAuxVariables_update(model)
```

Arguments

`model`	A Kriging model of `km` class.

Details

This function was introduced to optimize the calculation time of some expensive to evaluate integral criteria.

Value

An updated `km` model

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

See Also

computeAuxVariables

Examples

```
#computeAuxVariables_update

set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

model2 <- computeAuxVariables_update(model)
#in this example model2 and model are the same!
```

---

computeQuickKrigcov *Quick computation of kriging covariances*

Description

Computes kriging covariances between one new point and many integration points, using precomputed data.

Usage

```
computeQuickKrigcov(model,integration.points,X.new,
precalc.data, F.newdata , c.newdata)
```

Arguments

model	A Kriging model of km class.
integration.points	p*d matrix of points for numerical integration in the X space.
X.new	The new point where we calculate kriging covariances. The calculated covariances are the covariances between this new point and all the integration points.
precalc.data	List containing precalculated data. This list is generated using the function precomputeUpdateData
F.newdata	The value of the kriging trend basis function at point X.new
c.newdata	The (unconditional) covariance between X.new and the design points

Details

This function requires to use another function in order to generate the proper arguments. The argument `precalc.data` can be generated using `precomputeUpdateData`. The arguments `F.newdata` and `c.newdata` can be obtained using `predict_nobias_km`, which returns a field `F.newdata` and a field `c`.

Value

A vector containing kriging covariances

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

`precomputeUpdateData`, `predict_nobias_km`, `predict_update_km`

Examples

```
#computeQuickKrigcov

set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

#the points where we want to compute prediction
#if a point new.x is added to the doe
n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
newdata <- expand.grid(x.grid,y.grid)

#precalculation
precalc.data <- precomputeUpdateData(model=model,
```

```
integration.points=newdata)

#now we can compute very quickly kriging covariances
#between these data and any other points
other.x <- matrix(c(0.6,0.6),ncol=2)
pred <- predict_nobias_km(object=model,
newdata=other.x,type="UK",se.compute=TRUE)

kn <- computeQuickKrigcov(model=model,
integration.points=newdata,X.new=other.x,
        precalc.data=precalc.data,
F.newdata=pred$F.newdata,
        c.newdata=pred$c)

z.grid <- matrix(kn, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
contour(x=x.grid,y=y.grid,z=z.grid,levels=0,add=TRUE,col="blue",lwd=5)
points(design, col="black", pch=17, lwd=4,cex=2)
points(other.x, col="red", pch=17, lwd=4,cex=3)
title("Kriging covariances with the point (0.6,0.6), in red")
```

`computeRealVolumeConstant` *A constant used to calculate the expected excursion set's volume variance*

Description

This function computes a constant used to calculate exactly the value of the `"jn"` criterion at one point. Computing this constant does NOT change the optimum of the `"jn"` criterion. Therefore, its calculation is indicative only and is only necessary to know exactly (in expectation) the excursion set's volume variance.

Usage

```
computeRealVolumeConstant(model,integration.points,
integration.weights=NULL,T)
```

Arguments

`model`	A Kriging model of `km` class.
`integration.points`	p*d matrix of points for numerical integration in the X space.
`integration.weights`	(Optional) Vector of size p corresponding to the weights of these integration points. If not provided, all weights are set to 1.
`T`	the targeted (scalar) output value

Value

a scalar

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

precomputeUpdateData, predict_nobias_km, predict_update_km

Examples

```
#computeRealVolumeConstant

set.seed(8)
N <- 9 #number of observations
testfun <- branin
T <- 80

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

integcontrol <- list(n.points=50,distrib="sur",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,
lower=c(0,0),upper=c(1,1),
model=model,T=T)
integration.points <- obj$integration.points
integration.weights <- obj$integration.weights

## Not run:
computeRealVolumeConstant(model=model,
integration.points=integration.points,
        integration.weights=integration.weights,T=T)

## End(Not run)
```

EGI *Efficient Global Inversion: sequential inversion algorithm based on Kriging*

Description

Sequential sampling based on the optimization of a kriging-based criterion, with model update after each evaluation. Seven criteria are available for selecting experiments, three inexpensive ("bichon", "ranjan", and "tmse") and four expensive ones that require numerical integration ("imse", "timse", "sur" and "jn").

Usage

```
EGI(T, model, method = NULL, method.param = NULL,
fun, iter, lower, upper, new.noise.var = 0,
optimcontrol = NULL, kmcontrol = NULL, integcontrol = NULL, ...)
```

Arguments

T	Target value (a real number). The sampling algorithm and the underlying kriging model aim at finding the points for which the output is close to T.
model	A Kriging model of km class.
method	Criterion used for choosing observations. Available criteria are "ranjan" (default) , "bichon", "tmse", "timse", "imse", "sur" and "jn".
method.param	Optional tolerance value (scalar) for methods "ranjan", "bichon", "tmse" and "timse". If not provided, default value is used (1 for ranjan and bichon, 0 for tmse and timse).
fun	Objective function.
iter	Number of iterations (i.e. number of additional sampling points).
lower	Vector containing the lower bounds of the design space.
upper	Vector containing the upper bounds of the design space.
new.noise.var	Optional scalar value of the noise variance of the new observations.
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 "genoud" (default) for an optimisation using the genoud algorithm, or "discrete" for an optimisation over a specified discrete set. If the field method is set to "genoud", one can set some parameters of this algorithm: pop.size (default : 50*d), max.generations (default : 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.

kmcontrol — Optional list representing the control variables for the re-estimation of the kriging model once new points are sampled. The items are the same as in km.

integcontrol — Optional list specifying the procedure to build the integration points and weights, relevant only for the sampling criteria based on numerical integration: ("imse", "timse", "sur" or "jn"). Many options are possible. A) If nothing is specified, 100*d points are chosen using the Sobol sequence. B) One can directly set the field integration.points (a p * d matrix) for prespecified integration points. In this case these integration points and the corresponding vector integration.weights will be used for all the iterations of the algorithm. C) If the field integration.points is not set then the integration points are renewed at each iteration. In that case one can control the number of integration points n.points (default: 100*d) and a specific distribution distrib. Possible values for distrib are: "sobol", "MC", "timse", "imse", "sur" and "jn" (default: "sobol"). C.1) The choice "sobol" corresponds to integration points chosen with the Sobol sequence in dimension d (uniform weight). C.2) The choice "MC" corresponds to points chosen randomly, uniformly on the domain. C.3) The choices "timse", "imse", "sur" and "jn" correspond to importance sampling distributions (unequal weights). It is strongly recommended to use the importance sampling distribution corresponding to the chosen sampling criterion. When important sampling procedures are chosen, n.points points are chosen using importance sampling among a discrete set of n.candidates points (default: n.points*10) which are distributed according to a distribution init.distrib (default: "sobol"). Possible values for init.distrib are the space filling distributions "sobol" and "MC" or an user defined distribution "spec". The "sobol" and "MC" choices correspond to quasi random and random points in the domain. If the "spec" value is chosen the user must fill in manually the field init.distrib.spec to specify himself a n.candidates * d matrix of points in dimension d.

... — Other arguments of the target function fun.

Details

The function used to build the integration points and weights (based on the options specified in integcontrol) is the function integration_design

Value

A list with components:

par — The added observations (ite * d matrix)

value — The value of the function fun at the added observations (vector of size "ite")

nsteps — The number of added observations (=ite).

lastmodel — The current (last) kriging model of km class.

lastvalue — The value of the criterion at the last added point.

allvalues — If an optimization on a discrete set of points is chosen, the value of the criterion at all these points, for the last iteration.

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

Victor Picheny (CERFACS, Toulouse, France)

David Ginsbourger (IMSV, University of Bern, Switzerland)

References

Chevalier C., Picheny V., Ginsbourger D. (2012), *The KrigInv package: An efficient and user-friendly R implementation of Kriging-based inversion algorithms* , http://hal.archives-ouvertes.fr/hal-00713537/

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

max_sur, max_timse, max_infill_criterion

Examples

```
#EGI

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=50)
integcontrol <- list(distrib="sur",n.points=50)
iter <- 1

## Not run:
obj1 <- EGI(T=T,model=model,method="sur",fun=testfun,iter=iter,
           lower=lower,upper=upper,optimcontrol=optimcontrol,
           integcontrol=integcontrol)
```

```
obj2 <- EGI(T=T,model=model,method="ranjan",fun=testfun,iter=iter,
           lower=lower,upper=upper,optimcontrol=optimcontrol)


par(mfrow=c(1,3))
print_uncertainty_2d(model=model,T=T,main="probability of excursion",
type="pn",new.points=0,cex.points=2)

print_uncertainty_2d(model=obj1$lastmodel,T=T,
main="updated probability of excursion, sur sampling",
type="pn",new.points=iter,col.points.end="red",cex.points=2)

print_uncertainty_2d(model=obj2$lastmodel,T=T,
main="updated probability of excursion, ranjan sampling",
type="pn",new.points=iter,col.points.end="red",cex.points=2)

## End(Not run)
##############
#same example with noisy initial observations and noisy new observations
branin.noise <- function(x) return(branin(x)+rnorm(n=1,sd=30))

set.seed(8)
N <- 9;T <- 80
testfun <- branin.noise
lower <- c(0,0);upper <- c(1,1)

design <- data.frame( matrix(runif(2*N),ncol=2) )
response.noise <- apply(design,1,testfun)
response.noise - response


model.noise <- km(formula=~., design = design, response = response.noise,
covtype="matern3_2",noise.var=rep(30*30,times=N))

optimcontrol <- list(method="genoud",pop.size=50)
integcontrol <- list(distrib="sur",n.points=50)
iter <- 1

## Not run:
obj1 <- EGI(T=T,model=model.noise,method="sur",fun=testfun,iter=iter,
           lower=lower,upper=upper,optimcontrol=optimcontrol,
           integcontrol=integcontrol,new.noise.var=30*30)

obj2 <- EGI(T=T,model=model.noise,method="ranjan",fun=testfun,iter=iter,
           lower=lower,upper=upper,optimcontrol=optimcontrol,
            new.noise.var=30*30)

par(mfrow=c(1,3))
print_uncertainty_2d(model=model.noise,T=T,
main="probability of excursion, noisy obs.",
type="pn",new.points=0,cex.points=2)

print_uncertainty_2d(model=obj1$lastmodel,T=T,
```

```
main="probability of excursion, sur sampling, noisy obs.",
type="pn",new.points=iter,col.points.end="red",cex.points=2)

print_uncertainty_2d(model=obj2$lastmodel,T=T,
main="probability of excursion, ranjan sampling, noisy obs.",
type="pn",new.points=iter,col.points.end="red",cex.points=2)

## End(Not run)
```

`EGIparallel` *Efficient Global Inversion: Parallel version.*

Description

Sequential sampling based on the optimization of a kriging-based criterion, with model update after each evaluation. Similar to EGI, except that samples can be chosen in batches instead of one at a time. Three criteria are available for now.

Usage

```
EGIparallel(T, model, method = NULL, method.param=NULL,
fun, iter, batchsize = 1,
lower, upper, new.noise.var = 0,
optimcontrol = NULL, kmcontrol = NULL, integcontrol = NULL, ...)
```

Arguments

`T`	Target value (scalar). The sampling algorithm and the underlying kriging model at finding the points for which the output is close to T.
`model`	A Kriging model of `km` class.
`method`	Criterion used for choosing observations. The two sampling methods available in parallel version are `"sur"`, `"timse"` and `"vorob"`.
`method.param`	Optional tolerance value (a real number) for method `"timse"`.
`batchsize`	Number of points to sample simultaneously. The sampling criterion will return `batchsize` points at a time for sampling.
`new.noise.var`	Optional scalar value of the noise variance of the new observations.
`fun`	Objective function.
`iter`	Number of iterations.
`lower`	Vector containing the lower bounds of the variables to be optimized over.
`upper`	Vector containing the upper bounds of the variables to be optimized over.
`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 `"genoud"` (default) for an optimisation using the genoud algorithm, or `"discrete"` for an optimisation over a specified discrete set. If the field `method` is set to `"genoud"`, one can set some parameters of this algorithm: `pop.size`

(default : 50*d), max.generations (default : 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.

kmcontrol — Optional list representing the control variables for the re-estimation of the kriging model once new points are sampled. The items are the same as in km.

integcontrol — Optional list specifying the procedure to build the integration points and weights. Many options are possible. A) If nothing is specified, 100*d points are chosen using the Sobol sequence.
B) One can directly set the field integration.points (a p * d matrix) for prespecified integration points. In this case these integration points and the corresponding vector integration.weights will be used for all the iterations of the algorithm. C) If the field integration.points is not set then the integration points are renewed at each iteration. In that case one can control the number of integration points n.points (default: 100*d) and a specific distribution distrib. Possible values for distrib are: "sobol", "MC", "timse", "imse", "sur" and "jn" (default: "sobol"). C.1) The choice "sobol" corresponds to integration points chosen with the Sobol sequence in dimension d (uniform weight). C.2) The choice "MC" corresponds to points chosen randomly, uniformly on the domain. C.3) The choices "timse", "imse", "sur" and "jn" correspond to importance sampling distributions (unequal weights). It is strongly recommended to use the importance sampling distribution corresponding to the chosen sampling criterion. When important sampling procedures are chosen, n.points points are chosen using importance sampling among a discrete set of n.candidates points (default: n.points*10) which are distributed according to a distribution
init.distrib (default: "sobol"). Possible values for init.distrib are the space filling distributions "sobol" and "MC" or an user defined distribution "spec". The "sobol" and "MC" choices correspond to quasi random and random points in the domain. If the "spec" value is chosen the user must fill in manually the field init.distrib.spec to specify himself a n.candidates * d matrix of points in dimension d.

... — Other arguments of the target function fun.

Details

The function used to build the integration points and weights (based on the options specified in integcontrol) is the function integration_design

Value

A list with components:

par	The added observations ((iter*batchsize) * d matrix)
value	The value of fun at the added observations (size: iter*batchsize)
nsteps	The number of added observations (=iter*batchsize).
lastmodel	The current (last) kriging model of km class.
lastvalue	The value of the criterion at the last added batch of points.
allvalues	If an optimization on a discrete set of points is chosen, the value of the criterion at all these points, for the last iteration, for the last point of the batch.

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

Victor Picheny (CERFACS, Toulouse, France)

References

Chevalier C., Picheny V., Ginsbourger D. (2012), *The KrigInv package: An efficient and user-friendly R implementation of Kriging-based inversion algorithms* , http://hal.archives-ouvertes.fr/hal-00713537/

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI, max_sur_parallel, sur_optim_parallel

Examples

```
#EGIparallel

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
```

```
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=50)
integcontrol <- list(distrib="sur",n.points=50)
iter <- 1
batchsize <- 6

## Not run:
obj <- EGIparallel(T=T,model=model,method="sur",batchsize=batchsize,
fun=testfun,iter=iter,lower=lower,upper=upper,
optimcontrol=optimcontrol,integcontrol=integcontrol)

par(mfrow=c(1,2))
print_uncertainty_2d(model=model,T=T,main="probability of excursion",
type="pn",new.points=0,cex.points=2)

print_uncertainty_2d(model=obj$lastmodel,T=T,
main="probability of excursion, parallel sur sampling",
type="pn",new.points=iter*batchsize,col.points.end="red",cex.points=2)

## End(Not run)

###############
#same example with noisy initial observations and noisy new observations
branin.noise <- function(x) return(branin(x)+rnorm(n=1,sd=30))

set.seed(8)
N <- 9;T <- 80
testfun <- branin.noise
lower <- c(0,0);upper <- c(1,1)

design <- data.frame( matrix(runif(2*N),ncol=2) )
response.noise <- apply(design,1,testfun)
response.noise - response

model.noise <- km(formula=~., design = design, response = response.noise,
covtype="matern3_2",noise.var=rep(30*30,times=N))

optimcontrol <- list(method="genoud",pop.size=50)
integcontrol <- list(distrib="sur",n.points=50)
iter <- 1
batchsize <- 6

## Not run:
obj <- EGIparallel(T=T,model=model.noise,method="sur",batchsize=batchsize,
                   fun=testfun,iter=iter,lower=lower,upper=upper,
                   optimcontrol=optimcontrol,integcontrol=integcontrol,
                   new.noise.var=30*30)


par(mfrow=c(1,2))
```

```
print_uncertainty_2d(model=model.noise,T=T,
main="probability of excursion, noisy obs.",
type="pn",new.points=0,cex.points=2)

print_uncertainty_2d(model=obj$lastmodel,T=T,
main="probability of excursion, parallel sur sampling, noisy obs.",
type="pn",new.points=iter*batchsize,col.points.end="red",cex.points=2)

## End(Not run)
```

`integration_design` *Construction of a sample of integration points and weights*

Description

Generic function to build integration points for some sampling criterion. Available important sampling schemes are `"sur"`, `"jn"`, `"timse"`, `"vorob"` and `"imse"`. Each of them corresponds to a sampling criterion.

Usage

```
integration_design(integcontrol = NULL, d = NULL,
lower, upper, model = NULL, T = NULL,min.prob=0.001)
```

Arguments

`integcontrol` Optional list specifying the procedure to build the integration points and weights, relevant only for the sampling criteria based on numerical integration: (`"imse"`, `"timse"`, `"sur"` or `"jn"`). Many options are possible. A) If nothing is specified, 100*d points are chosen using the Sobol sequence. B) One can directly set the field `integration.points` (a p * d matrix) for prespecified integration points. In this case these integration points and the corresponding vector `integration.weights` will be used for all the iterations of the algorithm. C) If the field `integration.points` is not set then the integration points are renewed at each iteration. In that case one can control the number of integration points `n.points` (default: 100*d) and a specific distribution `distrib`. Possible values for `distrib` are: `"sobol"`, `"MC"`, `"timse"`, `"imse"`, `"sur"` and `"jn"` (default: `"sobol"`). C.1) The choice `"sobol"` corresponds to integration points chosen with the Sobol sequence in dimension d (uniform weight). C.2) The choice `"MC"` corresponds to points chosen randomly, uniformly on the domain. C.3) The choices `"timse"`, `"imse"`, `"sur"` and `"jn"` correspond to importance sampling distributions (unequal weights). It is strongly recommended to use the importance sampling distribution corresponding to the chosen sampling criterion. When important sampling procedures are chosen, `n.points` points are chosen using importance sampling among a discrete set of `n.candidates` points (default: `n.points*10`) which are distributed according to a distribution `init.distrib` (default: `"sobol"`). Possible values for `init.distrib` are the space filling distributions `"sobol"` and `"MC"` or an user defined distribution

"spec". The "sobol" and "MC" choices correspond to quasi random and random points in the domain. If the "spec" value is chosen the user must fill in manually the field init.distrib.spec to specify himself a n.candidates * d matrix of points in dimension d.

d — The dimension of the input set. If not provided d is set equal to the length of lower.

lower — Vector containing the lower bounds of the design space.

upper — Vector containing the upper bounds of the design space.

model — A Kriging model of km class.

T — Target value (scalar). The sampling algorithm and the underlying kriging model aim at finding the points for which the output is close to T.

min.prob — This argument applies only when importance sampling distributions are chosen (to compute integral criteria like "sur" or "timse"). For numerical reasons we give a minimum probability for a point to belong to the importance sample. This avoids probabilities equal to zero and importance sampling weights equal to infinity. In an importance sample of M points, the maximum weight becomes 1/min.prob * 1/M.

Details

The important sampling aims at improving the accuracy of the calculation of numerical integrals present in these criteria.

Value

A list with components:

integration.points — p x d matrix of p points used for the numerical calculation of integrals

integration.weights — a vector of size p corresponding to the weight of each point. If all the points are equally weighted, integration.weights is set to NULL

alpha — if the "vorob" important sampling schemes is chosen, the function also returns a scalar, alpha, being the calculated Vorob'ev threshold

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Picheny V., Ginsbourger D. (2012), *The KrigInv package: An efficient and user-friendly R implementation of Kriging-based inversion algorithms* , http://hal.archives-ouvertes.fr/hal-00713537/

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI, max_timse, max_sur

Examples

```
#integration_design

## Not run:
set.seed(8)
#when nothing is specified: integration points
#are chosen with the sobol sequence
integ.param <- integration_design(lower=c(0,0),upper=c(1,1))
plot(integ.param$integration.points)

## End(Not run)

#an example with pure random integration points
integcontrol <- list(distrib="MC",n.points=50)
integ.param <- integration_design(integcontrol=integcontrol,
lower=c(0,0),upper=c(1,1))
plot(integ.param$integration.points)

#an example with important sampling distributions
#these distributions are used to compute integral criterion like
#"sur","timse" or "imse"

#for these, we need a kriging model
N <- 14;testfun <- branin; T <- 80
lower <- c(0,0);upper <- c(1,1)
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

integcontrol <- list(distrib="timse",n.points=200,n.candidates=5000,init.distrib="MC")
integ.param <- integration_design(integcontrol=integcontrol,
lower=c(0,0),upper=c(1,1), model=model,T=T)

print_uncertainty_2d(model=model,T=T,type="timse",
col.points.init="red",cex.points=2,
main="timse uncertainty and one sample of integration points")

points(integ.param$integration.points,pch=17,cex=1)
```

jn_optim *jn criterion optimization*

Description

Evaluation of the "jn" criterion for a candidate point. To be used in the optimization routines max_sur with the argument real.volume.variance=TRUE. To avoid numerical instabilities, a new point is added to the design of experiments only if it is not too close to an existing observation, or if there is some observation noise. The criterion is the integral of the posterior expected jn uncertainty, which is the posterior expected variance of the excursion set's volume.

Usage

```
jn_optim(x, integration.points, integration.weights = NULL,
  intpoints.oldmean, intpoints.oldsd,
  precalc.data, model, T, new.noise.var = NULL,current.sur=0)
```

Arguments

x	Input vector of size d at which the criterion is evaluated.
integration.points	p*d matrix of points for numerical integration in the X space.
integration.weights	Vector of size p corresponding to the weights of these integration points. If not provided, uniform weight is used.
intpoints.oldmean	Vector of size p corresponding to the kriging mean at the integration points before adding x to the design of experiments.
intpoints.oldsd	Vector of size p corresponding to the kriging standard deviation at the integration points before adding x to the design of experiments.
precalc.data	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the precomputeUpdateData function.
model	Object of class km (Kriging model).
T	Target value (scalar).
new.noise.var	Optional scalar value of the noise variance for the new observations.
current.sur	Arbitrary value given to the "jn" criterion for candidate points that are too close to existing observations. This argument applies only if the noise variance is zero.

Value

jn value

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI, max_sur, sur_optim

Examples

```
#jn_optim

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="jn",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,lower=c(0,0),upper=c(1,1),
       model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
pred <- predict_nobias_km(object=model,newdata=integration.points,
                          type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)

x <- c(0.5,0.4)#one evaluation of the jn criterion
jn_optim(x=x,integration.points=integration.points,
         integration.weights=integration.weights,
         intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
         precalc.data=precalc.data,T=T,model=model)
#returns a negative value
```

```
#this is normal as a positive (constant) part of the criterion is not computed
#this constant part does not depend on x

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
jn.grid <- apply(X=x,FUN=jn_optim,MARGIN=1,integration.points=integration.points,
         integration.weights=integration.weights,
         intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
         precalc.data=precalc.data,T=T,model=model)
z.grid <- matrix(jn.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)

i.best <- which.min(jn.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of jn criterion (black) and of f(x)=T (blue)")
```

`max_infill_criterion` *Optimizer for the infill criteria*

Description

Global optimization, based on the package rgenoud (or on exhaustive search on a discrete set), of the chosen infill criterion (maximization or minimization, depending on the case)

Usage

```
max_infill_criterion(lower, upper, optimcontrol = NULL,
method, T, model, method.param = 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 `"genoud"` (default) for an optimisation using the genoud algorithm, or `"discrete"` for an optimisation over a specified discrete set. 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.
method	Criterion used for choosing observations: "ranjan" (default) , "bichon", or "tmse".
T	Target value (scalar).
model	A Kriging model of km class.
method.param	Optional tolerance value (scalar). Default value is 1 for "ranjan" and "bichon", and 0 for "tmse".

Value

A list with components:

par	The best set of parameters found.
value	The value of the chosen criterion at par.
allvalues	If an optimization on a discrete set of points is chosen, the value of the criterion at all these points.

Author(s)

Victor Picheny (CERFACS, Toulouse, France)

David Ginsbourger (IMSV, University of Bern, Switzerland)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Bichon, B.J., Eldred, M.S., Swiler, L.P., Mahadevan, S., McFarland, J.M.: Efficient global reliability analysis for nonlinear implicit performance functions. AIAA Journal 46 (10), 2459-2468 (2008)

Ranjan, P., Bingham, D., Michailidis, G.: Sequential experiment design for contour estimation from complex computer codes. Technometrics 50(4), 527-541 (2008)

See Also

EGI,ranjan_optim,tmse_optim,bichon_optim

Examples

```
#max_infill_criterion

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=50)

obj <- max_infill_criterion(lower=lower,upper=upper,optimcontrol=optimcontrol,
                            method="bichon",T=T,model=model)

obj$par;obj$value
new.model <- update_km(model=model,NewX=obj$par,testfun(obj$par),CovReEstimate=TRUE)

## Not run:
par(mfrow=c(1,2))
print_uncertainty(model=model,T=T,type="pn",lower=lower,upper=upper,
cex.points=2.5,main="probability of excursion")

print_uncertainty(model=new.model,T=T,type="pn",lower=lower,upper=upper,
new.points=1,col.points.end="red",cex.points=2.5,main="updated probability of excursion")

## End(Not run)
```

---

max_sur *Minimizer of the sur criterion*

Description

Minimization, based on the package rgenoud (or on exhaustive search on a discrete set), of the sur criterion.

Usage

```
max_sur(lower, upper, optimcontrol = NULL,
integration.param = NULL, T, model,
new.noise.var = 0, real.volume.variance = FALSE, real.volume.constant = FALSE)
```

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 "genoud" (default) for an optimisation using the genoud algorithm, or "discrete" for an optimisation over a specified discrete set. 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.
integration.param	Optional list of control parameter for the computation of integrals, containing the fields integration.points: a p*d matrix corresponding to p integrations points and integration.weights: a vector of size p corresponding to the weights of these integration points. If nothing is specified, default values are used (see: function integration_design for more details).
T	Target value (scalar).
model	A Kriging model of km class.
new.noise.var	Optional scalar value of the noise variance at the new observation.
real.volume.variance	Boolean to decide if the "jn" sampling criterion should be used instead of the "sur" criterion. When it is equal to FALSE (default), the "sur" criterion is used (faster computation).
real.volume.constant	When the"jn" criterion is chosen, this argument decides whether a constant part of the "jn" criterion should be computed or not. Computing this constant does NOT change the optimum of the "jn" criterion. Default value: FALSE. This argument is ignored if the argument real.volume.variance is set to FALSE.

Value

A list with components:

par	Best set of parameters found.
value	Value of the criterion at par.
allvalues	If an optimization on a discrete set of points is chosen, the value of the criterion at all these points
variance.volume	If the arguments real.volume.variance and real.volume.constant are both set to TRUE,the value of the computed constant

Author(s)

Victor Picheny (CERFACS, Toulouse, France)

David Ginsbourger (IMSV, University of Bern, Switzerland)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

Vazquez, E., Bect, J.: A sequential Bayesian algorithm to estimate a probability of failure. In: Proceedings of the 15th IFAC Symposium on System Identification, (SYSID 2009), Saint-Malo, France (2009)

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

See Also

EGI,sur_optim

Examples

```
#max_sur

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=50)
integcontrol <- list(distrib="sur",n.points=50,init.distrib="MC")
integration.param <- integration_design(integcontrol=integcontrol,d=2,
                                        lower=lower,upper=upper,model=model,
                                        T=T)

## Not run:
obj <- max_sur(lower=lower,upper=upper,optimcontrol=optimcontrol,T=T,
               model=model,integration.param=integration.param)
```

```
obj$par;obj$value
new.model <- update_km(model=model,NewX=obj$par,NewY=testfun(obj$par),
                       CovReEstimate=TRUE)

par(mfrow=c(1,2))
print_uncertainty(model=model,T=T,type="pn",lower=lower,upper=upper,
cex.points=2.5,main="probability of excursion")

print_uncertainty(model=new.model,T=T,type="pn",lower=lower,upper=upper,
new.points=1,col.points.end="red",cex.points=2.5,
main="updated probability of excursion")

## End(Not run)
```

`max_sur_parallel` *Minimizer of the parallel sur criterion*

Description

Minimization, based on the package rgenoud (or on exhaustive search on a discrete set), of the sur criterion for a batch of candidate sampling points.

Usage

```
max_sur_parallel(lower, upper, optimcontrol = NULL,
batchsize, integration.param, T,
model, new.noise.var = 0)
```

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 `"genoud"` (default) for an optimisation using the genoud algorithm, or `"discrete"` for an optimisation over a specified discrete set. 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.

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 integrations points and integration.weights: a vector of size p corresponding to the weights of these integration points. If nothing is specified, default values are used (see: function integration_design for more details).
T	Target value (scalar).
model	A Kriging model of km class.
new.noise.var	Optional scalar value of the noise variance of the new observations.

Value

A list with components:

par	the best set of points found.
value	the value of the sur criterion at par.
allvalues	If an optimization on a discrete set of points is chosen, the value of the criterion at all these points.

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

EGIparallel,sur_optim_parallel

Examples

```
#max_sur_parallel


set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)
```

```
#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=50,optim.option=1)
integcontrol <- list(distrib="sur",n.points=50,init.distrib="MC")
integration.param <- integration_design(integcontrol=integcontrol,d=2,
                                        lower=lower,upper=upper,model=model,
                                        T=T)
batchsize <- 5 #number of new points

## Not run:
obj <- max_sur_parallel(lower=lower,upper=upper,optimcontrol=optimcontrol,
                        batchsize=batchsize,T=T,model=model,
                        integration.param=integration.param)
                        #one optim in dimension 5*2 !

obj$par;obj$value #optimum in 5 new points
new.model <- update_km(model=model,NewX=obj$par,NewY=apply(obj$par,1,testfun),
                       CovReEstimate=TRUE)


par(mfrow=c(1,2))
print_uncertainty(model=model,T=T,type="pn",lower=lower,upper=upper,
cex.points=2.5,main="probability of excursion")

print_uncertainty(model=new.model,T=T,type="pn",lower=lower,upper=upper,
new.points=batchsize,col.points.end="red",cex.points=2.5,
main="updated probability of excursion")

## End(Not run)
```

`max_timse` *Minimizer of the IMSE or targeted IMSE criterion*

Description

Minimization, based on the package rgenoud (or on exhaustive search on a discrete set), of the targeted imse (or imse) criterion.

Usage

```
max_timse(lower, upper, optimcontrol = NULL,
integration.param = NULL, T, model,
new.noise.var = 0, epsilon = 0, imse = FALSE)
```

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 "genoud" (default) for an optimisation using the genoud algorithm, or "discrete" for an optimisation over a specified discrete set. 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.
integration.param	Optional list of control parameter for the computation of integrals, containing the fields integration.points: a p*d matrix corresponding to p integrations points and integration.weights: a vector of size p corresponding to the weights of these integration points. If nothing is specified, default values are used (see: function integration_design for more details).
T	Target value (scalar).
model	A Kriging model of km class.
new.noise.var	Optional scalar value of the noise variance of the new observation.
epsilon	Optional tolerance value (a real positive number). Default value is 0.
imse	Optional boolean to decide if the "imse" criterion should be used instead of "timse". Default: FALSE.

Value

A list with components:

par	the best set of parameters found.
value	the value of the criterion at par.
allvalues	if an optimization on a discrete set of points is chosen, the value of the criterion at all these points

Author(s)

Victor Picheny (CERFACS, Toulouse, France)

David Ginsbourger (IMSV, University of Bern, Switzerland)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Picheny V., Improving accuracy and compensating for uncertainty in surrogate modeling, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

See Also

EGI,timse_optim

Examples

```
#max_timse

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=50)
integcontrol <- list(distrib="timse",n.points=50,init.distrib="MC")
integration.param <- integration_design(integcontrol=integcontrol,d=2,
                                        lower=lower,upper=upper,model=model,
                                        T=T)

## Not run:
obj <- max_timse(lower=lower,upper=upper,optimcontrol=optimcontrol,T=T,
                model=model,integration.param=integration.param)

obj$par;obj$value
new.model <- update_km(model=model,NewX=obj$par,NewY=testfun(obj$par),
                       CovReEstimate=TRUE)

par(mfrow=c(1,2))
print_uncertainty(model=model,T=T,type="pn",lower=lower,upper=upper,
cex.points=2.5,main="probability of excursion")
```

```
print_uncertainty(model=new.model,T=T,type="pn",lower=lower,upper=upper,
new.points=1,col.points.end="red",cex.points=2.5,
main="updated probability of excursion")

## End(Not run)
```

`max_timse_parallel` *Minimizer of the parallel timse criterion*

Description

Minimization, based on the package rgenoud (or on exhaustive search on a discrete set), of the timse criterion for a batch of candidate sampling points.

Usage

```
max_timse_parallel(lower, upper, optimcontrol = NULL,
batchsize, integration.param, T,
model, new.noise.var = 0,
epsilon=0,imse=FALSE)
```

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 `"genoud"` (default) for an optimisation using the genoud algorithm, or `"discrete"` for an optimisation over a specified discrete set. 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.
`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 integrations points and integration.weights: a vector of size p corresponding to the weights of these integration points. If nothing is specified, default values are used (see: function integration_design for more details).

T
: Target value (scalar).

model
: A Kriging model of km class.

new.noise.var
: Optional scalar value of the noise variance of the new observations.

epsilon
: Optional tolerance value (a real positive number). Default value is 0.

imse
: Optional boolean to decide if the "imse" criterion should be used instead of "timse". default: FALSE.

Value

A list with components:

par
: the best set of parameters found.

value
: the value of the sur criterion at par.

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

Author(s)

Victor Picheny (CERFACS, Toulouse, France) Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Picheny V., Improving accuracy and compensating for uncertainty in surrogate modeling, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

See Also

EGIparallel,max_sur_parallel

Examples

```
#max_timse_parallel


set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=200,optim.option=2)
integcontrol <- list(distrib="timse",n.points=400,init.distrib="MC")
integration.param <- integration_design(integcontrol=integcontrol,d=2,
                                        lower=lower,upper=upper,model=model,
                                        T=T)
batchsize <- 5 #number of new points

## Not run:
obj <- max_timse_parallel(lower=lower,upper=upper,optimcontrol=optimcontrol,
                        batchsize=batchsize,T=T,model=model,
                        integration.param=integration.param,epsilon=0,imse=FALSE)
                        #5 optims in dimension 2 !

obj$par;obj$value #optimum in 5 new points
new.model <- update_km(model=model,NewX=obj$par,NewY=apply(obj$par,1,testfun),
                       CovReEstimate=TRUE)

par(mfrow=c(1,2))
print_uncertainty(model=model,T=T,type="pn",lower=lower,upper=upper,
cex.points=2.5,main="probability of excursion")

print_uncertainty(model=new.model,T=T,type="pn",lower=lower,upper=upper,
new.points=batchsize,col.points.end="red",cex.points=2.5,
main="updated probability of excursion")


## End(Not run)
```

`max_vorob_parallel` *Minimizer of the parallel vorob criterion*

Description

Minimization, based on the package rgenoud (or on exhaustive search on a discrete set), of the Vorob'ev criterion for a batch of candidate sampling points.

Usage

```
max_vorob_parallel(lower, upper, optimcontrol = NULL,
batchsize, integration.param, T,
model, new.noise.var = 0)
```

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 `"genoud"` (default) for an optimisation using the genoud algorithm, or `"discrete"` for an optimisation over a specified discrete set. 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.
`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 integrations points and `integration.weights`: a vector of size p corresponding to the weights of these integration points. If nothing is specified, default values are used (see: function `integration_design` for more details).
`T`	Target value (scalar).
`model`	A Kriging model of `km` class.
`new.noise.var`	Optional scalar value of the noise variance of the new observations.

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.

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

EGIparallel,max_sur_parallel

Examples

```
#max_vorob_parallel


set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

optimcontrol <- list(method="genoud",pop.size=200,optim.option=2)
integcontrol <- list(distrib="timse",n.points=400,init.distrib="MC")
integration.param <- integration_design(integcontrol=integcontrol,d=2,
                                        lower=lower,upper=upper,model=model,
                                        T=T)
batchsize <- 5 #number of new points

## Not run:
obj <- max_vorob_parallel(lower=lower,upper=upper,optimcontrol=optimcontrol,
                          batchsize=batchsize,T=T,model=model,
```

```
                        integration.param=integration.param)
                        #5 optims in dimension 2 !

obj$par;obj$value #optimum in 5 new points
new.model <- update_km(model=model,NewX=obj$par,NewY=apply(obj$par,1,testfun),
                       CovReEstimate=TRUE)

par(mfrow=c(1,2))
print_uncertainty(model=model,T=T,type="pn",lower=lower,upper=upper,vorobmean=TRUE,
cex.points=2.5,main="probability of excursion")

print_uncertainty(model=new.model,T=T,type="pn",lower=lower,upper=upper,vorobmean=TRUE,
new.points=batchsize,col.points.end="red",cex.points=2.5,
main="updated probability of excursion")


## End(Not run)
```

`precomputeUpdateData` *Useful data to quickly update kriging mean and variance*

Description

This function is used in combination with `computeQuickKrigcov` and computes an output list that serves as input in that function.

Usage

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

Arguments

`model`	A Kriging model of `km` class.
`integration.points`	p*d matrix of points for numerical integration in the X space.

Value

A list with components:

`Kinv.c.olddata`	Matrix equal to K^(-1)*c where K is the non conditional covariance matrix at the design points and c is the non conditional covariances between the design points and the integration points.
`Kinv.F`	Matrix equal to K^(-1)*F where F is a matrix with the values of the trend functions at the design points.
`first.member`	Matrix with a complicated expression.

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

computeQuickKrigcov, predict_nobias_km, predict_update_km

Examples

```
#precomputeUpdateData

set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

#the points where we want to compute prediction (if a point new.x is added to the doe)
n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
newdata <- expand.grid(x.grid,y.grid)
precalc.data <- precomputeUpdateData(model=model,integration.points=newdata)

#now we can compute very quickly kriging covariances
#between these data and any other points
other.x <- matrix(c(0.6,0.6),ncol=2)
pred <- predict_nobias_km(object=model,newdata=other.x,type="UK",se.compute=TRUE)

kn <- computeQuickKrigcov(model=model,integration.points=newdata,X.new=other.x,
                    precalc.data=precalc.data,F.newdata=pred$F.newdata,
                    c.newdata=pred$c)

z.grid <- matrix(kn, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
```

```
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
contour(x=x.grid,y=y.grid,z=z.grid,levels=0,add=TRUE,col="blue",lwd=5)
points(design, col="black", pch=17, lwd=4,cex=2)
points(other.x, col="red", pch=17, lwd=4,cex=3)
title("Kriging covariances with the point (0.6,0.6), in red")
```

`predict_nobias_km` *Kriging predictions without any adjustment factor for the kriging variances*

Description

This function is a simple modification of the predict.km function from the DiceKriging package. The only change lies in the adjustment factor for the kriging variances when universal kriging is used. Here this adjustment factor is simply removed. The rest of the function remains exactly as in the function predict.km of the DiceKriging package.

Usage

```
predict_nobias_km(object, newdata, type = "UK",
se.compute = TRUE, cov.compute = FALSE, low.memory=FALSE,...)
```

Arguments

`object`	A Kriging model of `km` class.
`newdata`	Vector, matrix or data frame containing the points where to perform predictions.
`type`	Character string corresponding to the kriging family, to be chosen between simple kriging ("SK"), or universal kriging ("UK").
`se.compute`	Optional boolean. If `FALSE`, only the kriging mean is computed. If `TRUE`, the kriging standard deviation and confidence intervals are computed too.
`cov.compute`	Optional boolean. If `TRUE` the conditional covariance matrix is computed.
`low.memory`	Optional boolean. If set to `TRUE` the function will only return kriging means and standard deviations.
`...`	No other arguments.

Details

When `type = "UK"`, the estimated variance and covariance are no longer multiplied by n/(n-p), where `n` and `p` are respectively the number of rows and the number of columns of the design matrix F.

Value

mean	kriging mean (including the trend) computed at newdata.
sd	kriging standard deviation computed at newdata. Not computed if se.compute=FALSE.
cov	kriging conditional covariance matrix. Not computed if cov.compute=FALSE (default).
lower95, upper95	bounds of the 95 % confidence interval computed at newdata (to be interpreted with special care when parameters are estimated, see description above). Not computed if se.compute=FALSE.
c	an auxiliary matrix, containing all the covariances between newdata and the initial design points.
Tinv.c	an auxiliary vector, equal to T^(-1)*c.
F.newdata	value of the trend function at newdata.

Warning

Beware that the only consistency check between newdata and the experimental design is to test whether they have same number of columns. In that case, the columns of newdata are interpreted in the same order as the initial design.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

A.G. Journel and C.J. Huijbregts (1978), *Mining Geostatistics*, Academic Press, London.

D.G. Krige (1951), A statistical approach to some basic mine valuation problems on the witwatersrand, *J. of the Chem., Metal. and Mining Soc. of South Africa*, **52** no. 6, 119-139.

J.D. Martin and T.W. Simpson (2005), Use of kriging models to approximate deterministic computer models, *AIAA Journal*, **43** no. 4, 853-863.

G. Matheron (1963), Principles of geostatistics, *Economic Geology*, **58**, 1246-1266.

G. Matheron (1969), Le krigeage universel, *Les Cahiers du Centre de Morphologie Mathematique de Fontainebleau*, **1**.

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, http://www.GaussianProcess.org/gpml

J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn (1989), Design and analysis of computer experiments, *Statistical Science*, **4**, 409-435.

See Also

predict.km, km

Examples

```
#predict_nobias_km
set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

n.grid <- 100
x.grid <- y.grid <- seq(0,1,length=n.grid)

#points where we want to compute prediction (if a point new.x is added to the doe)
newdata <- expand.grid(x.grid,y.grid)
pred <- predict_nobias_km(object=model,newdata=newdata,type="UK",se.compute=TRUE)

z.grid1 <- matrix(pred$mean, n.grid, n.grid)
z.grid2 <- matrix(pred$sd, n.grid, n.grid)

par(mfrow=c(1,2))

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid1,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid1,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
title("Kriging mean")

image(x=x.grid,y=y.grid,z=z.grid2,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid2,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
title("Kriging standard deviation")
```

---

predict_update_km *Quick computation of updated kriging means and variances.*

Description

This function uses kriging update formula to quickly compute kriging mean and variances at points newdata, when a point newX is added. The required additional informations are the old kriging

mean and variance at point newX, the output value f(newX), the old kriging mean and variances at points newdata and the kriging covariance between newX and newdata.

Usage

```
predict_update_km(newXmean, newXvar, newXvalue,
newdata.oldmean, newdata.oldsd, kn)
```

Arguments

newXmean	Scalar: old kriging mean at newX (before adding newX to the design).
newXvar	Scalar: old kriging variance at newX (before adding newX to the design).
newXvalue	Scalar: value of the objective function at newX, f(newX).
newdata.oldmean	Vector: old kriging mean at the points newdata (before adding newX to the design)
newdata.oldsd	Vector: old kriging standard deviations at the points newdata (before adding newX to the design)
kn	Kriging covariances between the points newdata and newX. These covariances can be computed using the function computeQuickKrigcov

Value

A list with the following fields:

mean	Updated kriging mean at points newdata
sd	Updated kriging standard deviation at points newdata
lambda	New kriging weight of newX for the prediction at points newdata

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

precomputeUpdateData, predict_nobias_km, computeQuickKrigcov

Examples

```
#predict_update_km

set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

#points where we want to compute prediction (if a point new.x is added to the doe)
n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
newdata <- expand.grid(x.grid,y.grid)
precalc.data <- precomputeUpdateData(model=model,integration.points=newdata)
pred2 <- predict_nobias_km(object=model,newdata=newdata,type="UK",se.compute=TRUE)
newdata.oldmean <- pred2$mean; newdata.oldsd <- pred2$sd

new.x <- matrix(c(0.6,0.6),ncol=2)   #the point that we are going to add
pred1 <- predict_nobias_km(object=model,newdata=new.x,type="UK",se.compute=TRUE)
newXmean <- pred1$mean; newXvar <- pred1$sd^2; newXvalue <- pred1$mean + 2*pred1$sd

kn <- computeQuickKrigcov(model=model,integration.points=newdata,X.new=new.x,
                    precalc.data=precalc.data,F.newdata=pred1$F.newdata,
                    c.newdata=pred1$c)

updated.predictions <- predict_update_km(newXmean=newXmean,newXvar=newXvar,
newXvalue=newXvalue,newdata.oldmean=newdata.oldmean,
newdata.oldsd=newdata.oldsd,kn=kn)

#the new kriging variance is usually lower than the old one
updated.predictions$sd - newdata.oldsd

z.grid1 <- matrix(newdata.oldsd, n.grid, n.grid)
z.grid2 <- matrix(updated.predictions$sd, n.grid, n.grid)

par(mfrow=c(1,2))

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid1,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid1,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
title("Kriging standard deviation")

image(x=x.grid,y=y.grid,z=z.grid2,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid2,15,add=TRUE)
```

```
points(design, col="black", pch=17, lwd=4,cex=2)
points(new.x, col="red", pch=17, lwd=4,cex=2)
title("updated Kriging standard deviation")
```

`predict_update_km_parallel` *Quick update of kriging means and variances when many new points are added.*

Description

This function is the parallel version of the function `predict_update_km`. It uses kriging update formula to quickly compute kriging mean and variances at points newdata, when `r` new points `newX` are added.

Usage

```
predict_update_km_parallel(newXmean, newXvar, newXvalue,
Sigma.r, newdata.oldmean, newdata.oldsd, kn)
```

Arguments

`newXmean`	Vector of size r: old kriging mean at points x_(n+1),...,x_(n+r).
`newXvar`	Vector of size r: kriging variance at points x_(n+1),...,x_(n+r).
`newXvalue`	Vector of size r: value of the objective function at x_(n+1),...,x_(n+r).
`Sigma.r`	An r*r matrix: kriging covariances between the points x_(n+1),...,x_(n+r).
`newdata.oldmean`	Vector: old kriging mean at the points `newdata` (before adding x_(n+1),...,x_(n+r))
`newdata.oldsd`	Vector: old kriging standard deviations at the points `newdata` (before adding x_(n+1),...,x_(n+r))
`kn`	Kriging covariances between the points `newdata` and the `r` points `newX`. These covariances can be computed using the function `computeQuickKrigcov`

Value

A list with the following fields:

`mean`	Updated kriging mean at points `newdata`
`sd`	Updated kriging standard deviation at points `newdata`
`lambda`	New kriging weight of x_(n+1),...,x_(n+r) for the prediction at points `newdata`

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

EGIparallel, max_sur_parallel, sur_optim_parallel

Examples

```
#predict_update_km_parallel

set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

#points where we want to compute prediction (if a point new.x is added to the doe)
n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
newdata <- expand.grid(x.grid,y.grid)
precalc.data <- precomputeUpdateData(model=model,integration.points=newdata)
pred2 <- predict_nobias_km(object=model,newdata=newdata,type="UK",se.compute=TRUE)
newdata.oldmean <- pred2$mean; newdata.oldsd <- pred2$sd

#the point that we are going to add
new.x <- matrix(c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8),ncol=2,byrow=TRUE)
pred1 <- predict_nobias_km(object=model,newdata=new.x,type="UK",
se.compute=TRUE,cov.compute=TRUE)
newXmean <- pred1$mean; newXvar <- pred1$sd^2; newXvalue <- pred1$mean + 2*pred1$sd
Sigma.r <- pred1$cov

kn <- computeQuickKrigcov(model=model,integration.points=newdata,X.new=new.x,
                    precalc.data=precalc.data,F.newdata=pred1$F.newdata,
                    c.newdata=pred1$c)

updated.predictions <- predict_update_km_parallel(newXmean=newXmean,newXvar=newXvar,
                                        newXvalue=newXvalue,Sigma.r=Sigma.r,
                                        newdata.oldmean=newdata.oldmean,
                                        newdata.oldsd=newdata.oldsd,kn=kn)
```

```
#the new kriging variance is usually lower than the old one
updated.predictions$sd - newdata.oldsd

z.grid1 <- matrix(newdata.oldsd, n.grid, n.grid)
z.grid2 <- matrix(updated.predictions$sd, n.grid, n.grid)

par(mfrow=c(1,2))

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid1,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid1,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
title("Kriging standard deviation")

image(x=x.grid,y=y.grid,z=z.grid2,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid2,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
points(new.x, col="red", pch=17, lwd=4,cex=2)
title("updated Kriging standard deviation")
```

`print_uncertainty` *Prints a measure of uncertainty for a function of any dimension.*

Description

This function prints in the whole input domain the value of a given measure of uncertainty. Possible measures are `"pn"` (the probability of excursion) and measures specific to a sampling criterion: `"sur"`, `"timse"` and `"imse"`. This function can be used to print relevant outputs after having used the function `EGI`.

Usage

```
print_uncertainty(model, T, type = "pn", ...)
```

Arguments

`model`	Kriging model of `km` class.
`T`	Target value (scalar).
`type`	Type of uncertainty that the user wants to print. Possible values are `"pn"` (probability of excursion), or `"sur"`, `"imse"`, `"timse"`, `"vorob"` if we print a measure of uncertainty corresponding to one criterion.
`...`	Other arguments of the function `print_uncertainty_1d`, `2d` or `nd`.

Value

the integrated uncertainty

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI

Examples

```
#print_uncertainty

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

#you could do many plots, but only one is run here
print_uncertainty(model=model,T=T,main="probability of excursion",type="pn")
#print_uncertainty(model=model,T=T,main="Vorob'ev uncertainty",type="vorob")
#print_uncertainty(model=model,T=T,main="imse uncertainty",type="imse")
#print_uncertainty(model=model,T=T,main="timse uncertainty",type="timse")
#print_uncertainty(model=model,T=T,main="sur uncertainty",type="sur")
#print_uncertainty(model=model,T=T,main="probability of excursion",type="pn",
#vorobmean=TRUE)
```

`print_uncertainty_1d` *Prints a measure of uncertainty for 1d function.*

Description

This function draws the value of a given measure of uncertainty over the whole input domain (1D). Possible measures are "pn" (beeing the probability of excursion) and measures specific to a sampling criterion: "sur", "timse" and "imse". This function can be used to print relevant outputs after having used the function EGI.

Usage

```
print_uncertainty_1d(model, T, type = "pn",
lower = 0, upper = 1, resolution = 500,
new.points = 0, xlab = "", ylab = "", main = "",
xscale = c(0, 1), show.points = TRUE, cex.main = 1, cex.lab = 1,
cex.points = 1, cex.axis = 1, pch.points.init = 17, pch.points.end = 17,
col.points.init = "black", col.points.end = "red", xaxislab = NULL,
yaxislab = NULL, xaxispoint = NULL, yaxispoint = NULL,
xdecal = 3, ydecal = 3,DiceViewplot=FALSE,vorobmean=FALSE)
```

Arguments

model	Kriging model of km class.
T	Target value (scalar).
type	Type of uncertainty that the user wants to print. Possible values are "pn" (probability of excursion), or "sur", "imse", "timse", "vorob" if we print a measure of uncertainty corresponding to one criterion.
lower	Lower bound for the input domain.
upper	Upper bound for the input domain.
resolution	Number of points to discretize the interval (lower,upper).
new.points	Number of new observations. These observations are the last new.points observations and can be printed in another color and the initial observations (see argument: col.points.end).
xlab	Label for the x axis.
ylab	Label for the y axis.
main	Title of the graph.
xscale	If one wants to rescale the input domain on another interval it is possible to set this vector of size 2. The new interval will be translated by xscale[1] and expanded by a factor xscale[2] - xscale[1].
show.points	Boolean: should we show the observations on the graph ?
cex.main	Multiplicative factor for the size of the title.
cex.lab	Multiplicative factor for the size of titles of the axis.
cex.points	Multiplicative factor for the size of the points.
cex.axis	Multiplicative factor for the size of the axis graduations.
pch.points.init	Symbol for the n-new.points first observations.

pch.points.end	Symbol for the new.points last observations.
col.points.init	Color for the n-new.points first observations.
col.points.end	Color for the new.points last observations.
xaxislab	Optional new labels that will replace the normal levels on x axis.
yaxislab	Optional new labels that will replace the normal levels on y axis.
xaxispoint	Position of these new labels on x axis.
yaxispoint	Position of these new labels on y axis.
xdecal	Optional position shifting of the titles of the x axis.
ydecal	Optional position shifting of the titles of the y axis.
DiceViewplot	Optional boolean. When it is set to TRUE (default) a second plot is added, generated with the DiceView package. This plot shows the kriging mean and confidence intervals on the whole input domain.
vorobmean	Optional boolean. When it is set to TRUE the Vorob'ev expectation is plotted. It corresponds to the averaged excursion set, using the definition of Vorob'ev. Here, the estimated set is the set above the Vorob'ev threshold (plotted in blue).

Value

the integrated uncertainty

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI

Examples

```
#print_uncertainty_1d

set.seed(8)
N <- 9 #number of observations
T <- 1 #threshold
testfun <- fundet
lower <- c(0)
upper <- c(1)

#a 9 points initial design
```

```
design <- data.frame( matrix(runif(N),ncol=1) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

print_uncertainty_1d(DiceViewplot=FALSE,model=model,T=T,
main="probability of excursion",cex.points=1.5,col.points.init="red",vorobmean=TRUE)
```

`print_uncertainty_2d` *Prints a measure of uncertainty for 2d function.*

Description

This function draws the value of a given measure of uncertainty over the whole input domain (2D). Possible measures are `"pn"` (probability of excursion) and measures specific to a sampling criterion: `"sur"`, `"timse"` and `"imse"`. This function can be used to print relevant outputs after having used the function `EGI`.

Usage

```
print_uncertainty_2d(model, T, type = "pn",
lower = c(0, 0), upper = c(1, 1), resolution = 200,
new.points = 0, xlab = "", ylab = "", main = "",
xscale = c(0, 1), yscale = c(0, 1), show.points = TRUE,
cex.main = 1, cex.lab = 1, cex.contourlab = 1, cex.points = 1,
cex.axis = 1, pch.points.init = 17, pch.points.end = 17,
col.points.init = "black", col.points.end = "red", nlevels = 10,
levels = NULL, xaxislab = NULL, yaxislab = NULL,
xaxispoint = NULL, yaxispoint = NULL, xdecal = 3, ydecal = 3,
krigmeanplot=FALSE,vorobmean=FALSE)
```

Arguments

`model`	Kriging model of `km` class.
`T`	Target value (scalar).
`type`	Type of uncertainty that the user wants to print. Possible values are `"pn"` (probability of excursion), or `"sur"`, `"imse"`, `"timse"`, `"vorob"` if we print a measure of uncertainty corresponding to one criterion.
`lower`	Vector containing the lower bounds of the input domain.
`upper`	Vector containing the upper bounds of the input domain.
`resolution`	Number of points to discretize the domain. This discretization is used in each dimension, so that the total number of points is `resolution^2`.

new.points	Number of new observations. These observations are the last new.points observations and can be printed in another color and the initial observations (see argument: col.points.end).
xlab	Label for the x axis.
ylab	Label for the y axis.
main	Title of the graph.
xscale	If one wants to rescale the input domain on another interval it is possible to set this vector of size 2. The new interval will be translated by xscale[1] and expanded by a factor xscale[2] - xscale[1].
yscale	see: xscale.
show.points	Boolean: should we show the observations on the graph ?
cex.main	Multiplicative factor for the size of the title.
cex.lab	Multiplicative factor for the size of titles of the axis.
cex.contourlab	Multiplicative factor for the size of labels of the contour plot.
cex.points	Multiplicative factor for the size of the points.
cex.axis	Multiplicative factor for the size of the axis graduations.
pch.points.init	Symbol for the n-new.points first observations.
pch.points.end	Symbol for the new.points last observations.
col.points.init	Color for the n-new.points first observations.
col.points.end	Color for the new.points last observations.
nlevels	Integer corresponding to the number of levels of the contour plot.
levels	Array: one can directly set the levels of the contour plot.
xaxislab	Optional new labels that will replace the normal levels on x axis.
yaxislab	Optional new labels that will replace the normal levels on y axis.
xaxispoint	Position of these new labels on x axis.
yaxispoint	Position of these new labels on y axis.
xdecal	Optional position shifting of the titles of the x axis.
ydecal	Optional position shifting of the titles of the y axis.
krigmeanplot	Optional boolean. When it is set to FALSE (default) the contour plot corresponds to the uncertainty selected. When it is set to TRUE the contour plot gives the kriging mean.
vorobmean	Optional boolean. When it is set to TRUE the Vorob'ev expectation is plotted. It corresponds to the averaged excursion set, using the definition of Vorob'ev.

Value

the integrated uncertainty

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI

Examples

```
#print_uncertainty_2d

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

print_uncertainty_2d(model=model,T=T,main="probability of excursion",
                     type="pn",krigmeanplot=TRUE)

#print_uncertainty_2d(model=model,T=T,main="vorob uncertainty",
#type="vorob",krigmeanplot=FALSE)

#print_uncertainty_2d(model=model,T=T,main="imse uncertainty",
#type="imse",krigmeanplot=FALSE)

#print_uncertainty_2d(model=model,T=T,main="timse uncertainty",
#type="timse",krigmeanplot=FALSE)

#print_uncertainty_2d(model=model,T=T,main="sur
#uncertainty",type="sur",krigmeanplot=FALSE)

#print_uncertainty_2d(model=model,T=T,main="probability of excursion",
#                     type="pn",krigmeanplot=TRUE,vorobmean=TRUE)
```

`print_uncertainty_nd` *Print a measure of uncertainty for functions with dimension d strictly higher than 2.*

Description

This function draws projections on various plans of a given measure of uncertainty. Possible measures are `"pn"` (probability of excursion) and measures specific to a sampling criterion: `"sur"`, `"timse"` and `"imse"`. This function can be used to print relevant outputs after having used the function `EGI`.

Usage

```
print_uncertainty_nd(model,T,type="pn",lower=NULL,upper=NULL,
   resolution=20, nintegpoints=400,main="",
cex.main=1,cex.lab=1,cex.contourlab=1,cex.axis=1,
nlevels=10,levels=NULL,
xdecal=3,ydecal=3, option="mean")
```

Arguments

`model`	Kriging model of `km` class.
`T`	Target value (a real number). The sampling algorithm and the underlying kriging model aim to find the points below (resp. over) T.
`type`	Type of uncertainty that the user wants to print. Possible values are `"pn"` (probability of excursion), or `"sur"`, `"imse"`, `"timse"` if we print a measure of uncertainty corresponding to one criterion.
`lower`	Vector containing the lower bounds of the input domain. If nothing is set we use a vector of 0.
`upper`	Vector containing the upper bounds of the input domain. If nothing is set we use a vector of 1.
`resolution`	Number of points to discretize a plan included in the domain. For the moment, we cannot use values higher than 40.
`nintegpoints`	to do
`main`	Title of the graph.
`cex.main`	Multiplicative factor for the size of the title.
`cex.lab`	Multiplicative factor for the size of titles of the axis.
`cex.contourlab`	Multiplicative factor for the size of labels of the contour plot.
`cex.axis`	Multiplicative factor for the size of the axis graduations.
`nlevels`	Integer corresponding to the number of levels of the contour plot.
`levels`	Array: one can directly set the levels of the contour plot.
`xdecal`	Optional position shifting of the titles of the x axis.
`ydecal`	Optional position shifting of the titles of the y axis.
`option`	Optional argument (a string). The 3 possible values are `"mean"` (default), `"max"` and `"min"`.

Value

the integrated uncertainty

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI

Examples

```
#print_uncertainty_nd

set.seed(8)
N <- 25 #number of observations
T <- -1 #threshold
testfun <- hartman3
#The hartman3 function is defined over the domain [0,1]^3.

hartman3(runif(3))

lower <- rep(0,times=3)
upper <- rep(1,times=3)

#a 9 points initial design (LHS in 3 dimensions)
design <- data.frame( matrix(runif(3*N),ncol=3) )
response <- apply(design,1,testfun)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

## Not run:
print_uncertainty_nd(model=model,T=T,main="average probability of excursion",type="pn",
                     option="mean")

print_uncertainty_nd(model=model,T=T,main="maximum probability of excursion",type="pn",
                      option="max")


## End(Not run)
```

ranjan_optim *Ranjan et al.'s Expected Improvement criterion*

Description

Evaluation of Ranjan's Expected Feasibility criterion. To be used in optimization routines, like in max_infill_criterion.

Usage

```
ranjan_optim(x, model, T, method.param = NULL)
```

Arguments

x	Input vector at which one wants to evaluate the criterion. This argument can be either a vector of size d (for an evaluation at a single point) or a p*d matrix (for p simultaneous evaluations of the criterion at p different points).
model	An object of class km (Kriging model).
T	Target value (scalar).
method.param	Scalar tolerance around the target T (default value is 1).

Value

Ranjan EI criterion. When the argument x is a vector the function returns a scalar. When the argument x is a p*d matrix the function returns a vector of size p.

Author(s)

V. Picheny (CERFACS, Toulouse, France)

D. Ginsbourger (IMSV, University of Bern, Switzerland)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Ranjan, P., Bingham, D., Michailidis, G.: Sequential experiment design for contour estimation from complex computer codes. Technometrics 50(4), 527-541 (2008)

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI, max_infill_criterion

Examples

```
##############################################################################
#ranjan_optim

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

x <- c(0.5,0.4)#one evaluation of the ranjan criterion
ranjan_optim(x=x,T=T,model=model)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
ranjan.grid <- ranjan_optim(x=x,T=T,model=model)
z.grid <- matrix(ranjan.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,25,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)

i.best <- which.max(ranjan.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of Ranjan criterion (black) and of f(x)=T (blue)")
```

`sur_optim` *sur criterion*

Description

Evaluation of the `"sur"` criterion for a candidate point. To be used in optimization routines, like in `max_sur`. To avoid numerical instabilities, the new point is evaluated only if it is not too close to an existing observation, or if there is some observation noise. The criterion is the integral of the posterior sur uncertainty.

Usage

```
sur_optim(x, integration.points, integration.weights = NULL,
intpoints.oldmean, intpoints.oldsd,
precalc.data, model, T,
new.noise.var = NULL,current.sur=1e6)
```

Arguments

x	Input vector of size d at which one wants to evaluate the criterion.
integration.points	p*d matrix of points for numerical integration in the X space.
integration.weights	Vector of size p corresponding to the weights of these integration points.
intpoints.oldmean	Vector of size p corresponding to the kriging mean at the integration points before adding x to the design of experiments.
intpoints.oldsd	Vector of size p corresponding to the kriging standard deviation at the integration points before adding x to the design of experiments.
precalc.data	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the precomputeUpdateData function.
model	Object of class km (Kriging model).
T	Target value (scalar)
new.noise.var	Optional scalar value of the noise variance for the new observations.
current.sur	Arbitrary value given to the "sur" criterion for candidate points that are too close to existing observations. This argument applies only if the noise variance is zero.

Value

sur value

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

Victor Picheny (CERFACS, Toulouse, France)

David Ginsbourger (IMSV, University of Bern, Switzerland)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

EGI, max_sur

Examples

```
#sur_optim

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="sur",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,lower=c(0,0),upper=c(1,1),
model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
pred <- predict_nobias_km(object=model,newdata=integration.points,
type="UK",se.compute=TRUE)

intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)


x <- c(0.5,0.4)#one evaluation of the sur criterion
sur_optim(x=x,integration.points=integration.points,
         integration.weights=integration.weights,
         intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
         precalc.data=precalc.data,T=T,model=model)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
sur.grid <- apply(X=x,FUN=sur_optim,MARGIN=1,integration.points=integration.points,
         integration.weights=integration.weights,
         intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
         precalc.data=precalc.data,T=T,model=model)#takes ~15seconds to run
z.grid <- matrix(sur.grid, n.grid, n.grid)
```

```
#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)

i.best <- which.min(sur.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of sur criterion (black) and of f(x)=T (blue)")
```

`sur_optim_parallel` *Parallel sur criterion*

Description

Evaluation of the parallel sur criterion for some candidate points. To be used in optimization routines, like in `max_sur_parallel`. 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 sur uncertainty.

Usage

```
sur_optim_parallel(x, integration.points, integration.weights = NULL,
intpoints.oldmean, intpoints.oldsd,
precalc.data, model, T,
new.noise.var = NULL, batchsize, current.sur)
```

Arguments

`x`	Input vector of size batchsize*d at which one wants to evaluate the criterion. This argument is NOT a matrix.
`integration.points`	p*d matrix of points for numerical integration in the X space.
`integration.weights`	Vector of size p corresponding to the weights of these integration points.
`intpoints.oldmean`	Vector of size p corresponding to the kriging mean at the integration points before adding the batchsize points `x` to the design of experiments.
`intpoints.oldsd`	Vector of size p corresponding to the kriging standard deviation at the integration points before adding the batchsize points `x` to the design of experiments.
`precalc.data`	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the `precomputeUpdateData` function.

model	Object of class km (Kriging model).
T	Target value (scalar).
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.
current.sur	Current value of the sur criterion (before adding new observations)

Details

The first argument x has been chosen to be a vector of size batchsize*d (and not a matrix with batchsize rows and d columns) so that an optimizer like genoud can optimize it easily. For example if d=2, batchsize=3 and x=c(0.1,0.2,0.3,0.4,0.5,0.6), we will evaluate the parallel criterion at the three points (0.1,0.2),(0.3,0.4) and (0.5,0.6). The last argument current.sur is used as a default value for the sur criterion when the new points x are too close to existing observations.

Value

Parallel sur value

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

EGIparallel, max_sur_parallel

Examples

```
#sur_optim_parallel

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
```

```
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="sur",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,
lower=c(0,0),upper=c(1,1),model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
pred <- predict_nobias_km(object=model,newdata=integration.points,
type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)

batchsize <- 4
x <- c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8)
#one evaluation of the sur_optim_parallel criterion
#we calculate the expectation of the future "sur" uncertainty
#when 4 points are added to the doe
#the 4 points are (0.1,0.2) , (0.3,0.4), (0.5,0.6), (0.7,0.8)
sur_optim_parallel(x=x,integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,
          batchsize=batchsize,current.sur=Inf)


#the function max_sur_parallel will help to find the optimum:
#ie: the batch of 4 minimizing the expectation of the future uncertainty
```

`sur_optim_parallel2` *Parallel sur criterion*

Description

Evaluation of the parallel sur criterion for some candidate points, assuming that some other points are also going to be evaluated. To be used in optimization routines, like in `max_sur_parallel`. 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 sur uncertainty.

Usage

```
sur_optim_parallel2(x, other.points,
integration.points, integration.weights = NULL,
```

```
intpoints.oldmean, intpoints.oldsd, precalc.data,
model, T, new.noise.var = NULL,
batchsize, current.sur)
```

Arguments

x	Input vector of size d at which one wants to evaluate the criterion. This argument corresponds to only ONE point.
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 X space.
integration.weights	Vector of size p corresponding to the weights of these integration points.
intpoints.oldmean	Vector of size p corresponding to the kriging mean at the integration points before adding x to the design of experiments.
intpoints.oldsd	Vector of size p corresponding to the kriging standard deviation at the integration points before adding x to the design of experiments.
precalc.data	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the precomputeUpdateData function.
model	Object of class km (Kriging model).
T	Target value (scalar).
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.
current.sur	Current value of the sur criterion (before adding new observations)

Details

The first argument x has been chosen to be a vector of size d so that an optimizer like genoud can optimize it easily. The second argument other.points is a vector of size (batchsize-1)*d corresponding to the batchsize-1 other points. The last argument current.sur is used as a default value for the sur criterion when the new points x are too close to existing observations.

Value

Parallel sur value

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

EGIparallel, max_sur_parallel

Examples

```
#sur_optim_parallel2

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="sur",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,lower=c(0,0),upper=c(1,1),
model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
pred <- predict_nobias_km(object=model,newdata=integration.points,
                          type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)

batchsize <- 4
other.points <- c(0.7,0.5,0.5,0.9,0.9,0.8)
x <- c(0.1,0.2)
#one evaluation of the sur_optim_parallel criterion2
#we calculate the expectation of the future "sur" uncertainty when
#1+3 points are added to the doe
#the 1+3 points are (0.1,0.2) and (0.7,0.5), (0.5,0.9), (0.9,0.8)
```

```
sur_optim_parallel2(x=x,other.points,integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,
          batchsize=batchsize,current.sur=Inf)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
sur_parallel.grid <- apply(X=x,FUN=sur_optim_parallel2,MARGIN=1,other.points,
          integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,
          batchsize=batchsize,current.sur=Inf)
z.grid <- matrix(sur_parallel.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
points(matrix(other.points,ncol=2,byrow=TRUE), col="red", pch=17, lwd=4,cex=2)

i.best <- which.min(sur_parallel.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of sur_parallel criterion (black) and of f(x)=T (blue)")
```

`timse_optim` *Targeted IMSE criterion*

Description

Evaluation of the `"timse"` criterion for a candidate point. To be used in optimization routines, like in `max_timse`. To avoid numerical instabilities, the new point is evaluated only if it is not too close to an existing observation, or if there is some observation noise. The criterion is the integral of the posterior timse uncertainty.

Usage

```
timse_optim(x, integration.points, integration.weights = NULL,
intpoints.oldmean = NULL, intpoints.oldsd = NULL,
precalc.data, model, T, new.noise.var = 0, weight = NULL)
```

Arguments

x	Input vector of size d at which one wants to evaluate the criterion.
integration.points	p*d matrix of points for numerical integration in the X space.
integration.weights	Vector of size p corresponding to the weights of these integration points.
intpoints.oldmean	Vector of size p corresponding to the kriging mean at the integration points before adding x to the design of experiments.
intpoints.oldsd	Vector of size p corresponding to the kriging standard deviation at the integration points before adding x to the design of experiments.
precalc.data	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the precomputeUpdateData function.
model	Object of class km (Kriging model).
T	Target value (scalar)
new.noise.var	Optional scalar value of the noise variance for the new observations.
weight	Vector of weight function (length must be equal to the number of lines of the matrix integration.points). If nothing is set, the imse criterion is used instead if timse. It corresponds to equal weights.

Value

targeted imse value

Author(s)

Victor Picheny (CERFACS, Toulouse, France)

David Ginsbourger (IMSV, University of Bern, Switzerland)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Picheny V., Improving accuracy and compensating for uncertainty in surrogate modeling, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

See Also

EGI, max_timse

Examples

```
#timse_optim

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="timse",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,lower=c(0,0),
upper=c(1,1),model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
pred <- predict_nobias_km(object=model,newdata=integration.points,
type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)

#we also need to compute weights. Otherwise the (more simple)
#imse criterion will be evaluated
weight <- 1/sqrt(2*pi*intpoints.oldsd^2) *
exp(-0.5*((intpoints.oldmean-T)/sqrt(intpoints.oldsd^2))^2)
weight[is.nan(weight)] <- 0

x <- c(0.5,0.4)#one evaluation of the timse criterion
timse_optim(x=x,integration.points=integration.points,
         integration.weights=integration.weights,
         intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
         precalc.data=precalc.data,T=T,model=model,weight=weight)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
timse.grid <- apply(X=x,FUN=timse_optim,MARGIN=1,integration.points=integration.points,
         integration.weights=integration.weights,
         intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
         precalc.data=precalc.data,T=T,model=model,weight=weight)
```

```
z.grid <- matrix(timse.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)

i.best <- which.min(timse.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of timse criterion (black) and of f(x)=T (blue)")
```

`timse_optim_parallel` *Parallel targeted IMSE criterion*

Description

Evaluation of the `"timse"` criterion for some candidate points. To be used in optimization routines, like in `max_timse_parallel`. 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 timse uncertainty.

Usage

```
timse_optim_parallel(x, integration.points, integration.weights = NULL,
intpoints.oldmean = NULL, intpoints.oldsd = NULL,
precalc.data, model, T, new.noise.var = 0, weight = NULL,
batchsize, current.timse)
```

Arguments

`x`	Input vector of size d at which one wants to evaluate the criterion.
`integration.points`	p*d matrix of points for numerical integration in the X space.
`integration.weights`	Vector of size p corresponding to the weights of these integration points.
`intpoints.oldmean`	Vector of size p corresponding to the kriging mean at the integration points before adding `x` to the design of experiments.
`intpoints.oldsd`	Vector of size p corresponding to the kriging standard deviation at the integration points before adding `x` to the design of experiments.
`precalc.data`	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the `precomputeUpdateData` function.

model	Object of class km (Kriging model).
T	Target value (scalar).
new.noise.var	Optional scalar value of the noise variance of the new observations.
weight	Vector of weight function (length must be equal to the number of lines of the matrix integration.points). If nothing is set, the imse criterion is used instead of timse. It corresponds to equal weights.
batchsize	Number of points to sample simultaneously. The sampling criterion will return batchsize points at a time for sampling.
current.timse	Current value of the timse criterion (before adding new observations)

Value

targeted imse value

Author(s)

Victor Picheny (CERFACS, Toulouse, France)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Picheny V., Improving accuracy and compensating for uncertainty in surrogate modeling, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

See Also

EGIparallel, max_timse_parallel

Examples

```
#timse_optim_parallel

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
```

```
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=1000,distrib="timse",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,lower=c(0,0),
upper=c(1,1),model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
pred <- predict_nobias_km(object=model,newdata=integration.points,
type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)

#we also need to compute weights. Otherwise the (more simple)
#imse criterion will be evaluated
weight <- 1/sqrt(2*pi*intpoints.oldsd^2) *
exp(-0.5*((intpoints.oldmean-T)/sqrt(intpoints.oldsd^2))^2)
weight[is.nan(weight)] <- 0

batchsize <- 4
x <- c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8)
#one evaluation of the timse_optim_parallel criterion
#we calculate the expectation of the future "timse"
#uncertainty when 4 points are added to the doe
#the 4 points are (0.1,0.2) , (0.3,0.4), (0.5,0.6), (0.7,0.8)
timse_optim_parallel(x=x,integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,weight=weight,
          batchsize=batchsize,current.timse=Inf)

#the function max_timse_parallel will help to find the optimum:
#ie: the batch of 4 minimizing the expectation of the future uncertainty
```

`timse_optim_parallel2` *Parallel timse criterion*

Description

Evaluation of the parallel timse criterion for some candidate points, assuming that some other points are also going to be evaluated. To be used in optimization routines, like in `max_timse_parallel`. 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 timse uncertainty.

Usage

```
timse_optim_parallel2(x, other.points,
integration.points, integration.weights = NULL,
intpoints.oldmean, intpoints.oldsd, precalc.data,
model, T, new.noise.var = NULL,weight = NULL,
batchsize, current.timse)
```

Arguments

x	Input vector of size d at which one wants to evaluate the criterion. This argument corresponds to only ONE point.
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 X space.
integration.weights	Vector of size p corresponding to the weights of these integration points.
intpoints.oldmean	Vector of size p corresponding to the kriging mean at the integration points before adding x to the design of experiments.
intpoints.oldsd	Vector of size p corresponding to the kriging standard deviation at the integration points before adding x to the design of experiments.
precalc.data	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the precomputeUpdateData function.
model	Object of class km (Kriging model).
T	Target value (scalar).
new.noise.var	Optional scalar value of the noise variance for the new observations.
weight	Vector of weight function (length must be equal to the number of lines of the matrix integration.points). If nothing is set, the imse criterion is used instead of timse. It corresponds to equal weights.
batchsize	Number of points to sample simultaneously. The sampling criterion will return batchsize points at a time for sampling.
current.timse	Current value of the timse criterion (before adding new observations)

Details

The first argument x has been chosen to be a vector of size d so that an optimizer like genoud can optimize it easily. The second argument other.points is a vector of size (batchsize-1)*d corresponding to the batchsize-1 other points. The last argument current.timse is used as a default value for the timse criterion when the new points x are too close to existing observations.

Value

Parallel timse value

Author(s)

Victor Picheny (CERFACS, Toulouse, France)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Picheny V., Improving accuracy and compensating for uncertainty in surrogate modeling, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

See Also

EGIparallel, max_timse_parallel

Examples

```
#timse_optim_parallel2

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=1000,distrib="timse",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,lower=c(0,0),upper=c(1,1),
model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
pred <- predict_nobias_km(object=model,newdata=integration.points,
type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
```

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

#we also need to compute weights. Otherwise the (more simple)
#imse criterion will be evaluated
weight <- 1/sqrt(2*pi*intpoints.oldsd^2) *
exp(-0.5*((intpoints.oldmean-T)/sqrt(intpoints.oldsd^2))^2)
weight[is.nan(weight)] <- 0

batchsize <- 4
other.points <- c(0.7,0.5,0.5,0.9,0.9,0.8)
x <- c(0.1,0.2)
#one evaluation of the timse_optim_parallel criterion2
#we calculate the expectation of the future "timse" uncertainty
#when 1+3 points are added to the doe
#the 1+3 points are (0.1,0.2) and (0.7,0.5), (0.5,0.9), (0.9,0.8)
timse_optim_parallel2(x=x,other.points,integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,weight=weight,
          batchsize=batchsize,current.timse=Inf)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
timse_parallel.grid <- apply(X=x,FUN=timse_optim_parallel2,MARGIN=1,other.points,
          integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,weight=weight,
          batchsize=batchsize,current.timse=Inf)
z.grid <- matrix(timse_parallel.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
points(matrix(other.points,ncol=2,byrow=TRUE), col="red", pch=17, lwd=4,cex=2)

i.best <- which.min(timse_parallel.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of timse_parallel criterion (black) and of f(x)=T (blue)")
```

`tmse_optim` *Targeted MSE criterion*

Description

Evaluation of the Targeted MSE criterion. To be used in optimization routines, like in max_infill_criterion

Usage

```
tmse_optim(x, model, T, method.param = NULL)
```

Arguments

x	Input vector at which one wants to evaluate the criterion. This argument can be either a vector of size d (for an evaluation at a single point) or a p*d matrix (for p simultaneous evaluations of the criterion at p different points).
model	An object of class km (Kriging model).
T	Target value (scalar).
method.param	Scalar tolerance around the target T.

Value

targeted MSE value. When the argument x is a vector the function returns a scalar. When the argument x is a p*d matrix the function returns a vector of size p.

Author(s)

V. Picheny (Ecole Centrale Paris)

D. Ginsbourger (IMSV, University of Bern, Switzerland)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R.T., Adaptive designs of experiments for accurate approximation of a target region, J. Mech. Des. - July 2010 - Volume 132, Issue 7, http://dx.doi.org/10.1115/1.4001873

Picheny V., Improving accuracy and compensating for uncertainty in surrogate modeling, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

See Also

EGI, max_infill_criterion

Examples

```
#tmse_optim

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin
```

```
#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

x <- c(0.5,0.4)#one evaluation of the tmse criterion
tmse_optim(x=x,T=T,model=model)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
tmse.grid <- tmse_optim(x=x,T=T,model=model)
z.grid <- matrix(tmse.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,25,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)

i.best <- which.max(tmse.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of tmse criterion (black) and of f(x)=T (blue)")
```

`tsee_optim` *Two Sided Expected Exceedance criterion*

Description

Evaluation of a two-sided Expected Exceedance criterion. To be used in optimization routines, like in `max_infill_criterion`.

Usage

```
tsee_optim(x, model, T)
```

Arguments

x	Input vector at which one wants to evaluate the criterion. This argument can be either a vector of size d (for an evaluation at a single point) or a p*d matrix (for p simultaneous evaluations of the criterion at p different points).

model	An object of class km (Kriging model).
T	Target value (scalar).

Value

tsee criterion. When the argument x is a vector the function returns a scalar. When the argument x is a p*d matrix the function returns a vector of size p.

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France) Yann Richet (IRSN, France)

See Also

EGI, max_infill_criterion

Examples

```
#tsee_optim

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

x <- c(0.5,0.4)#one evaluation of the tsee criterion
tsee_optim(x=x,T=T,model=model)

n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
tsee.grid <- tsee_optim(x=x,T=T,model=model)
z.grid <- matrix(tsee.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,25,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)

i.best <- which.max(tsee.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
```

```
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of tsee criterion (black) and of f(x)=T (blue)")
```

`update_km` *Update of a kriging model*

Description

Update a `km` object when one or many new observations are added. Many, but not all, fields of the `km` object need to be recalculated when new observations are added. It is also possible to modify the k last (existing) observations.

Usage

```
update_km(model, NewX, NewY, NewX_AllreadyExist = FALSE,
CovReEstimate = FALSE, new.noise.var = NULL,
kmcontrol = NULL, F.newdata = NULL)
```

Arguments

`model`	Kriging model of `km` class.
`NewX`	Matrix with `model@d` columns and r rows corresponding to the r locations of the observations to be updated. These locations can be new locations or existing ones.
`NewY`	Matrix with one column and r rows corresponding to the r responses at the r locations NewX.
`NewX_AllreadyExist`	Boolean: indicate whether the locations NewX are all news or not.
`CovReEstimate`	Boolean to decide whether the covariance parameters of the `km` object should be re-estimated.
`new.noise.var`	Vector containing the noise variance at each new observations.
`kmcontrol`	Optional list representing the control variables for the re-estimation of the kriging model once new points are sampled. The items are the same as in `km`
`F.newdata`	Optional matrix containing the value of the trend at the new locations. Setting this argument avoids a call to an expensive function.

Value

Updated km object

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

See Also

km

Examples

```
#update_km

set.seed(8)
N <- 9 #number of observations
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")
model@covariance

new.x <- matrix(c(0.4,0.5),ncol=2)#the point that we are going to add in the km object
new.y <- testfun(new.x)
new.model <- update_km(model=model,NewX=new.x,NewY=new.y,CovReEstimate=TRUE)
new.model@covariance
```

---

vorob_optim_parallel *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. 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(x, integration.points, integration.weights = NULL,
intpoints.oldmean, intpoints.oldsd,
precalc.data, model, T,
new.noise.var = NULL, batchsize, alpha, current.vorob)
```

Arguments

x	Input vector of size batchsize*d at which one wants to evaluate the criterion. This argument is NOT a matrix.
integration.points	p*d matrix of points for numerical integration in the X space.
integration.weights	Vector of size p corresponding to the weights of these integration points.
intpoints.oldmean	Vector of size p corresponding to the kriging mean at the integration points before adding the batchsize points x to the design of experiments.
intpoints.oldsd	Vector of size p corresponding to the kriging standard deviation at the integration points before adding the batchsize points x to the design of experiments.
precalc.data	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the precomputeUpdateData function.
model	Object of class km (Kriging model).
T	Target value (scalar).
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	The Vorob'ev threshold.
current.vorob	Current value of the vorob criterion (before adding new observations)

Details

The first argument x has been chosen to be a vector of size batchsize*d (and not a matrix with batchsize rows and d columns) so that an optimizer like genoud can optimize it easily. For example if d=2, batchsize=3 and x=c(0.1,0.2,0.3,0.4,0.5,0.6), we will evaluate the parallel criterion at the three points (0.1,0.2),(0.3,0.4) and (0.5,0.6). The last argument current.vorob is used as a default value for the vorob criterion when the new points x are too close to existing observations.

Value

Parallel vorob value

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

EGIparallel, max_vorob_parallel

Examples

```
#vorob_optim_parallel

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="vorob",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,
lower=c(0,0),upper=c(1,1),model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
alpha <- obj$alpha
pred <- predict_nobias_km(object=model,newdata=integration.points,
type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)

batchsize <- 4
x <- c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8)
#one evaluation of the vorob_optim_parallel criterion
#we calculate the expectation of the future "vorob" uncertainty
#when 4 points are added to the doe
#the 4 points are (0.1,0.2) , (0.3,0.4), (0.5,0.6), (0.7,0.8)
```

```
vorob_optim_parallel(x=x,integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,
          batchsize=batchsize,alpha=alpha,current.vorob=Inf)


#the function max_vorob_parallel will help to find the optimum:
#ie: the batch of 4 minimizing the expectation of the future uncertainty
```

`vorob_optim_parallel2` *Parallel Vorob'ev criterion*

Description

Evaluation of the Vorob'ev criterion for some candidate points, assuming that some other points are also going to be evaluated. To be used in optimization routines, like in `max_vorob_parallel`. 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(x, other.points,
integration.points, integration.weights = NULL,
intpoints.oldmean, intpoints.oldsd, precalc.data,
model, T, new.noise.var = NULL,
batchsize, alpha, current.vorob)
```

Arguments

`x`	Input vector of size d at which one wants to evaluate the criterion. This argument corresponds to only ONE point.
`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 X space.
`integration.weights`	Vector of size p corresponding to the weights of these integration points.
`intpoints.oldmean`	Vector of size p corresponding to the kriging mean at the integration points before adding `x` to the design of experiments.
`intpoints.oldsd`	Vector of size p corresponding to the kriging standard deviation at the integration points before adding `x` to the design of experiments.

precalc.data	List containing useful data to compute quickly the updated kriging variance. This list can be generated using the precomputeUpdateData function.
model	Object of class km (Kriging model).
T	Target value (scalar).
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	The Vorob'ev threshold.
current.vorob	Current value of the vorob criterion (before adding new observations)

Details

The first argument x has been chosen to be a vector of size d so that an optimizer like genoud can optimize it easily. The second argument other.points is a vector of size (batchsize-1)*d corresponding to the batchsize-1 other points. The last argument current.vorob is used as a default value for the vorob criterion when the new points x are too close to existing observations.

Value

Parallel Vorob'ev value

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Chevalier C., Ginsbourger D. (2012), *Corrected Kriging update formulae for batch-sequential data assimilation* ,http://arxiv.org/pdf/1203.6452.pdf

See Also

EGIparallel, max_vorob_parallel

Examples

```
#vorob_optim_parallel2

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
```

```
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="vorob",init.distrib="MC")
obj <- integration_design(integcontrol=integcontrol,
lower=c(0,0),upper=c(1,1),model=model,T=T)

integration.points <- obj$integration.points
integration.weights <- obj$integration.weights
alpha <- obj$alpha
pred <- predict_nobias_km(object=model,newdata=integration.points,
                          type="UK",se.compute=TRUE)
intpoints.oldmean <- pred$mean ; intpoints.oldsd<-pred$sd

#another precomputation
precalc.data <- precomputeUpdateData(model,integration.points)

batchsize <- 4
other.points <- c(0.7,0.5,0.5,0.9,0.9,0.8)
x <- c(0.1,0.2)
#one evaluation of the vorob_optim_parallel criterion2
#we calculate the expectation of the future "vorob" uncertainty when
#1+3 points are added to the doe
#the 1+3 points are (0.1,0.2) and (0.7,0.5), (0.5,0.9), (0.9,0.8)
vorob_optim_parallel2(x=x,other.points,integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,
          batchsize=batchsize,alpha=alpha,current.vorob=Inf)


n.grid <- 20 #you can run it with 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
x <- expand.grid(x.grid, y.grid)
vorob_parallel.grid <- apply(X=x,FUN=vorob_optim_parallel2,MARGIN=1,other.points,
          integration.points=integration.points,
          integration.weights=integration.weights,
          intpoints.oldmean=intpoints.oldmean,intpoints.oldsd=intpoints.oldsd,
          precalc.data=precalc.data,T=T,model=model,
          batchsize=batchsize,alpha=alpha,current.vorob=Inf)
z.grid <- matrix(vorob_parallel.grid, n.grid, n.grid)

#plots: contour of the criterion, doe points and new point
image(x=x.grid,y=y.grid,z=z.grid,col=grey.colors(10))
contour(x=x.grid,y=y.grid,z=z.grid,15,add=TRUE)
points(design, col="black", pch=17, lwd=4,cex=2)
points(matrix(other.points,ncol=2,byrow=TRUE), col="red", pch=17, lwd=4,cex=2)
```

```
i.best <- which.min(vorob_parallel.grid)
points(x[i.best,], col="blue", pch=17, lwd=4,cex=3)

#plots the real (unknown in practice) curve f(x)=T
testfun.grid <- apply(x,1,testfun)
z.grid.2 <- matrix(testfun.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid.2,levels=T,col="blue",add=TRUE,lwd=5)
title("Contour lines of vorob_parallel criterion (black) and of f(x)=T (blue)")
```

---

vorob_threshold *Calculation of the Vorob'ev threshold*

Description

Evaluation of the Vorob'ev threshold given an excursion probability vector. This threshold is such that the volume of the set (x : pn(x) > threshold) is equal to the integral of pn.

Usage

```
vorob_threshold(pn)
```

Arguments

pn	Input vector of arbitrary size containing the excursion probabilities pn(x).

Details

In this function, all the points x are supposed tobe equaly weighted.

Value

a scalar: the Vorob'ev thresold

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set* ,http://hal.archives-ouvertes.fr/hal-00641108/

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

See Also

max_vorob_parallel, vorob_optim_parallel

Examples

```
#vorob_threshold

set.seed(8)
N <- 9 #number of observations
T <- 80 #threshold
testfun <- branin

#a 9 points initial design
design <- data.frame( matrix(runif(2*N),ncol=2) )
response <- testfun(design)

#km object with matern3_2 covariance
#params estimated by ML from the observations
model <- km(formula=~., design = design,
response = response,covtype="matern3_2")

## Not run:
###we need to compute some additional arguments:
#integration points, and current kriging means and variances at these points
integcontrol <- list(n.points=50,distrib="sobol")
obj <- integration_design(integcontrol=integcontrol,
lower=c(0,0),upper=c(1,1),model=model,T=T)

integration.points <- obj$integration.points

pred <- predict_nobias_km(object=model,newdata=integration.points,
type="UK",se.compute=TRUE)
pn <- pnorm((pred$mean-T)/pred$sd)

vorob_threshold(pn)

## End(Not run)
```

Index