# Package 'KrigInv'

May 7, 2026

**Type** Package

**Title** Kriging-Based Inversion for Deterministic and Noisy Computer Experiments

**Version** 1.4.2

**Date** 2022-09-06

**Maintainer** Dario Azzimonti <dario.azzimonti@gmail.com>

**Depends** DiceKriging

**Imports** randtoolbox, rgenoud, pbivnorm, anMC, mvtnorm

**Suggests** testthat

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

**URL** https://doi.org/10.1016/j.csda.2013.03.008

**License** GPL-3

**LazyLoad** yes

**Repository** CRAN

**RoxygenNote** 6.1.0

**NeedsCompilation** no

**Author** Clement Chevalier [aut],
Dario Azzimonti [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5080-3061>),
David Ginsbourger [aut] (ORCID:
<https://orcid.org/0000-0003-2724-2678>),
Victor Picheny [aut] (ORCID: <https://orcid.org/0000-0002-4948-5542>),
Yann Richet [ctb] (ORCID: <https://orcid.org/0000-0002-5677-8458>)

**Date/Publication** 2022-09-09 16:52:55 UTC

## Contents

---

`KrigInv-package` *Kriging-Based Inversion for Deterministic and Noisy Computer Experiments*

---

## Description

Kriging-based sequential algorithms, meant to identify the excursion set of a real valued function. The algorithms can also identify one or several level sets.

## Details

| | |
|---|---|
| Package: | KrigInv |
| Type: | Package |
| Version: | 1.4.1 |
| Date: | 2018-09-04 |
| License: | GPL version 3 |
| LazyLoad: | yes |

## Note

Important functions are `EGI` and `EGIparallel`. The last 1.4 version allows to handle multiple thresholds T, stored in an array and implements conservative excursion set strategies.

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!

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

Victor Picheny (INRA, Toulouse, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

with contributions from Yann Richet (IRSN, France)

Maintainer: Clement Chevalier (clement.chevalier@unine.ch)

## References

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets*. Under revision. Preprint at hal-01379642

Chevalier C., Picheny V., Ginsbourger D. (2014), *Kriginv: An efficient and user-friendly implementation of batch sequential inversion strategies based on kriging* (2014) Computational Statistics & Data Analysis, vol. 71, pp 1021-1034

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

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

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

Picheny V. (2009) *Improving accuracy and compensating for uncertainty in surrogate modeling*, Ph.D. thesis, University of Florida and Ecole Nationale Superieure des Mines de Saint-Etienne

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

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

---

`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 = 1)
```

## 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

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)

Victor Picheny (INRA, Toulouse, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

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

## See Also

EGI, max_infill_criterion

## Examples

```
#bichon_optim

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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 # resolution. You may use a larger value.
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)")
```

---

# computeQuickKrigcov *Quick computation of kriging covariances*

---

## Description

Computes kriging covariances between some new points and many integration points, using pre-computed 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 fixed integration points in the X space. |
| X.new | q*d matrix of new points. The calculated covariances are the covariances between these new point and 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.

## Value

Matrix of size p*q containing kriging covariances

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

precomputeUpdateData, predict_nobias_km

## Examples

```
#computeQuickKrigcov

set.seed(9)
N <- 20 #number of observations
testfun <- branin

#a 20 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 integration.points are the points where we want to
#compute predictions/covariances if a point new.x is added
#to the DOE
x.grid <- seq(0,1,length=20)
integration.points <- expand.grid(x.grid,x.grid)
integration.points <- as.matrix(integration.points)

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

#now we can compute quickly kriging covariances
#between these data and any other points.
#example if 5 new points are added:
X.new <- matrix(runif(10),ncol=2)
pred <- predict_nobias_km(object=model,
                          newdata=X.new,type="UK",se.compute=TRUE)

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

---

`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. 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 | Target threshold. |

## Details

Note that, even if the "jn" criterion can be used with more than one threshold, the computation of this constant is implemented only when the number of threshold is equal to 1.

## Value

a scalar

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

## See Also

precomputeUpdateData, predict_nobias_km

## Examples

```
#computeRealVolumeConstant

set.seed(9)
N <- 20 #number of observations
testfun <- branin
T <- 80

#a 20 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=500,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

## 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 iteration. The criterias aim at identifying an excursion set or one/many level sets. At each iteration batchsize = 1 new locations are evaluated. Different criteria are available for selecting experiments. The pointwise criteria are `"bichon"`, `"ranjan"`, `"tmse"`, `"tsee"` and are fast to compute. In addition, integral criteria require numerical integration and can potentially deliver more than one new location per iteration. Available integral criteria are `"imse"`, `"timse"`, `"sur"`, `"jn"`, `"vorob"`, `"vorobCons"`, `"vorobVol"`. The use of the integral criteria is implemented in the `EGIparallel` function.

## 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` | Array containing one or several thresholds. The criteria which can be used with multiple thresholds are `"tmse"`, `"timse"`, `"sur"`, `"jn"`. |
| `model` | A Kriging model of `km` class. |
| `method` | Criterion used for choosing observations. |
| `method.param` | Optional method parameters. For methods |

- "ranjan", "bichon", "tmse" and "timse": the tolerance value (scalar). If not provided, default value is used (1 for ranjan and bichon, 0 for tmse and timse).
- "vorobev": a list containing penalization (scalar, default=1), type I penalization, and typeEx,(character, default=">") either ">" or "<" denoting the type of excursion.
- "vorobCons" and "vorobVol": a list containing penalization (scalar, default =1), typeEx (character, default = ">"), consLevel (scalar, default=0.95), n_discrete_design (scalar, default=500*model@d), design (data.frame, default=as.data.frame(sobol (n = method.param$n_discrete_design, dim = model@d)) ), pred (list resulting from predict.km). See also the arguments alpha, pred, design, type from the function conservativeEstimate, package anMC, for more details.

| | |
|---|---|
| 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. |

If `method="vorobCons"` or `method="vorobVol"` the list also has components:

| | |
|---|---|
| `current.CE` | Conservative estimate computed on `lastmodel`. |
| `allCE_lvs` | The conservative estimate levels computed at each iteration. |

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

Victor Picheny (INRA, Toulouse, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

## References

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets*. Under revision. Preprint at hal-01379642

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

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

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

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

## See Also

EGIparallel,max_infill_criterion

## Examples

```
#EGI

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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(9)
N <- 20;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)


###############
# Conservative estimates with non-noisy initial observations
## Not run:
  testfun <- branin

  ## Minimize Type II error sampling

  # The list method.param contains all parameters for the
  # conservative estimate and the conservative sequential
  # strategy. Below are parameters for a type II strategy
  # with conservative estimates at 0.95
  method.param = list(penalization=0, # Type II strategy
                      typeEx=">", consLevel = 0.95,
                      n_discrete_design=500*model@d)
  # If the CE for the initial model is already computed
  # it is possible to pass the conservative Vorob'ev quantile
  # level with method.param$consVorbLevel

  obj_T2 <- EGI(T=T,model=model,method="vorobCons",
                fun=testfun,iter=iter,lower=lower,upper=upper,
                optimcontrol=optimcontrol,
                integcontrol=integcontrol,method.param=method.param)

  par(mfrow=c(1,2))
  print_uncertainty_2d(model=model,T=T,main="probability of excursion",
                 type="pn",new.points=0,cex.points=2,consQuantile = obj_T2$allCE_lvs[1])

  print_uncertainty_2d(model=obj_T2$lastmodel,T=T,
                       main="probability of excursion, parallel Type II sampling",
                       type="pn",new.points=iter,col.points.end="red",
                       cex.points=2,consQuantile = obj_T2$allCE_lvs[2])

  ## Maximize conservative estimate's volume
  # Set up method.param
  # Here we pass the conservative level computed
  # in the previous step for the initial model
  method.param = list(typeEx=">", consLevel = 0.95,
                      n_discrete_design=500*model@d,
                      consVorbLevel=obj_T2$allCE_lvs[1]
  )

  obj_consVol <- EGI(T=T,model=model,method="vorobVol",
                     fun=testfun,iter=iter,lower=lower,upper=upper,
                     optimcontrol=optimcontrol,
```

```
                    integcontrol=integcontrol,method.param=method.param)

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

  print_uncertainty_2d(model=obj_consVol$lastmodel,T=T,
                       main="probability of excursion, parallel consVol sampling",
                       type="pn",new.points=iter,col.points.end="red",
                       cex.points=2,consQuantile = obj_consVol$allCE_lvs[2])


## End(Not run)
```

---

`EGIparallel` *Efficient Global Inversion: parallel version to get batchsize locations at each iteration*

---

## Description

Sequential sampling based on the optimization of a kriging-based criterion, with model update after each iteration. The criterias aim at identifying an excursion set or one/many level sets. At each iteration batchsize new locations are evaluated. Different criteria are available for selecting experiments. The pointwise criteria are `"bichon"`, `"ranjan"`, `"tmse"`, `"tsee"` and are fast to compute. These criteria can be used only with batchsize = 1. In addition, integral criteria require numerical integration and can potentially deliver more than one new location per iteration. Available integral criteria are `"imse"`, `"timse"`, `"sur"`, `"jn"`, `"vorob"`, `"vorobCons"`, `"vorobVol"`.

## 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` | Array containing one or several thresholds. The criteria which can be used with multiple thresholds are `"tmse"`, `"timse"`, `"sur"`, `"jn"`. |
| `model` | A Kriging model of `km` class. |
| `method` | Criterion used for choosing observations. |
| `method.param` | Optional method parameters. For methods<br>• `"ranjan"`, `"bichon"`, `"tmse"` and `"timse"`: the tolerance value (scalar). If not provided, default value is used (1 for ranjan and bichon, 0 for tmse and timse). |

- "vorob": a list containing penalization (scalar, default=1), type I penalization, and typeEx,(character, default=">") either ">" or "<" denoting the type of excursion.
- "vorobCons" and "vorobVol": a list containing penalization (scalar, default =1), typeEx (character, default = ">"), consLevel (scalar, default=0.95), n_discrete_design (scalar, default=500*model@d), design (data.frame, default=as.data.frame(sobol (n = method.param$n_discrete_design, dim = model@d)) ), pred (result of predict.km on model at design) and consVorbLevel, the conservative estimate Vorob'ev quantile computed from pred. See also the arguments alpha, pred, design, type from the function conservativeEstimate, package anMC, for more details.

batchsize — Number of points to sample simultaneously. The sampling criterion will return batchsize points at each iteration. Some criteria can be used only with batchsize = 1 (see description).

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. |

If method="vorobCons" or method="vorobVol" the list also has components:

| | |
|---|---|
| current.CE | Conservative estimate computed on lastmodel. |
| allCE_lvs | The conservative estimate levels computed at each iteration. |

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

Victor Picheny (INRA, Toulouse, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

## References

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets*. Under revision. Preprint at hal-01379642

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

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

## See Also

EGI, max_sur_parallel

## Examples

```
#EGIparallel

set.seed(9)
N <- 20 #number of observations
T <- c(20,60) #thresholds
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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(9)
N <- 20;T <- c(20,60)
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=10*10)


  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)


##############
# Conservative estimates with non-noisy initial observations
## Not run:
  testfun <- branin
  # The conservative sampling strategies
  # only work with 1 threshold
  T <- 20
  ## Minimize Type II error sampling
```

```
# The list method.param contains all parameters for the
# conservative estimate and the conservative sequential
# strategy. Below are parameters for a type II strategy
# with conservative estimates at 0.95
method.param = list(penalization=0, # Type II strategy
                    typeEx=">", consLevel = 0.95,
                    n_discrete_design=500*model@d)
# If the CE for the initial model is already computed
# it is possible to pass the conservative Vorob'ev quantile
# level with method.param$consVorbLevel

obj_T2 <- EGIparallel(T=T,model=model,method="vorobCons",batchsize=batchsize,
                      fun=testfun,iter=iter,lower=lower,upper=upper,
                      optimcontrol=optimcontrol,
                      integcontrol=integcontrol,method.param=method.param)

par(mfrow=c(1,2))
print_uncertainty_2d(model=model,T=T,main="probability of excursion",
               type="pn",new.points=0,cex.points=2,consQuantile = obj_T2$allCE_lvs[1])

print_uncertainty_2d(model=obj_T2$lastmodel,T=T,
                     main="probability of excursion, parallel Type II sampling",
                     type="pn",new.points=iter*batchsize,col.points.end="red",
                     cex.points=2,consQuantile = obj_T2$allCE_lvs[2])

## Maximize conservative estimate's volume
# Set up method.param
# Here we pass the conservative level computed
# in the previous step for the initial model
method.param = list(typeEx=">", consLevel = 0.95,
                    n_discrete_design=500*model@d,
                    consVorbLevel=obj_T2$allCE_lvs[1]
)

obj_consVol <- EGIparallel(T=T,model=model,method="vorobVol",batchsize=batchsize,
                           fun=testfun,iter=iter,lower=lower,upper=upper,
                           optimcontrol=optimcontrol,
                           integcontrol=integcontrol,method.param=method.param)

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

print_uncertainty_2d(model=obj_consVol$lastmodel,T=T,
                     main="probability of excursion, parallel consVol sampling",
                     type="pn",new.points=iter*batchsize,col.points.end="red",
                     cex.points=2,consQuantile = obj_consVol$allCE_lvs[2])


## End(Not run)
```

`excursion_probability` *Excursion probability with one or many thresholds*

## Description

Probability that Gaussian random variables with some mean and variance are over a threshold `T`, or in an union of intervals. If `T` is a vector of size p, T1,T2,...,Tp then the considered union of interval is (T1,T2) U ... U (Tp, +infty) if p is odd, and (T1,T2) U ... U (Tp-1, Tp) if p is even.

## Usage

```
excursion_probability(mn,sn,T)
```

## Arguments

| | |
|---|---|
| `mn` | Array of size k containing the expectations of the Gaussian random variables. |
| `sn` | Array of size k containing the standard deviations of the Gaussian random variables. |
| `T` | Array containing one or several thresholds. |

## Value

Array of size k containing the k excursion probabilities.

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

## See Also

`predict_nobias_km`

## Examples

```
#excursion_probability

set.seed(9)
N <- 20 #number of observations
testfun <- branin

#a 20 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")

some_points <- matrix(runif(20),ncol=2)
pred <- predict_nobias_km(object = model,newdata = some_points,
                type = "UK",se.compute = TRUE)

T <- c(60,80,100)
excursion_probability(mn = pred$mean,sn = pred$sd,T=T)
# probability to be in the interval [60,80] U [100, infty]
```

---

`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"`, `"vorob"` 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"`, `"vorob"` and `"jn"` (default: `"sobol"`). C.1) The choice `"sobol"` corresponds to integration points chosen with the Sobol sequence in dimension d (equal weights). C.2) The choice `"MC"` corresponds to points chosen randomly, uniformly on the domain (equal weights). C.3) The choices `"timse"`, `"imse"`, `"sur"`, `"vorob"` and `"jn"` correspond to importance sampling distributions (unequal weights). It is 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 `"sobol"` or `"MC"` (uniform random points) or an user defined distribution `"spec"`. If the `"spec"` value is chosen the user must fill manually the field `init.distrib.spec` with 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` Array containing one or several thresholds.

`min.prob` This argument applies only when importance sampling distributions are chosen. For numerical reasons we give a minimum probability for a point to belong to the importance sample. This avoids potential 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 computation of criteria which involve numerical integration, like `"timse"`, `"sur"`, etc.

## Value

A list with components:

`integration.points`
p * d matrix of p points used for the numerical calculation of integrals

`integration.weights`
Vector of size p corresponding to the weights of each points. 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 (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

max_timse_parallel, max_sur_parallel

## Examples

```
#integration_design


#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)


#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
set.seed(9)
N <- 16;testfun <- branin
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="sur",n.points=200,n.candidates=5000,
  init.distrib="MC")

T <- c(60,100)
#we are interested in the set of points where the response is in [60,100]

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="sur",
col.points.init="red",cex.points=2,
main="sur uncertainty and one sample of integration points")
points(integ.param$integration.points,pch=17,cex=1)
```

---

jn_optim_parallel *Parallel jn criterion*

---

## Description

Evaluation of the parallel jn 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 expected "jn" uncertainty, which is the posterior expected variance of the excursion set's volume.

## Usage

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

## Arguments

| | |
|---|---|
| x | Vector of size batchsize*d at which one wants to evaluate the criterion. This argument is NOT a matrix. |
| integration.points | Matrix of points for numerical integration. Two cases are handled. If the "jn" importance sampling distribution has been used, this is a (2p)*d matrix containing p couples of integration points in D x D, where D is the integration domain. If instead a p*d matrix is given, then the function will perform the integration by using all p^2 couples. |
| integration.weights | Vector of size p corresponding to the weights of these integration points. |
| intpoints.oldmean | Vector of size p, or 2p, 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, or 2p, 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 | Array containing one or several thresholds. |
| 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). |
| ai_precalc | This is a matrix with ith row equal to intpoints.oldmean-T[i]. The argument DOES ALWAYS need to be filled, even if there is only one threshold T. |

## 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).

## Value

Parallel jn value

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

EGIparallel, max_sur_parallel

## Examples

```
#jn_optim_parallel

set.seed(9)
N <- 20 #number of observations
T <- c(80,100) #thresholds
testfun <- branin

#a 20 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
n.points <- 200
integcontrol <- list(n.points=n.points,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$integration.points # (2n.points)*d matrix
integration.weights <- obj$integration.weights #vector of size n.points
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)
nT <- 2 # number of thresholds
ai_precalc <- matrix(rep(intpoints.oldmean,times=nT),
    nrow=nT,ncol=length(intpoints.oldmean),byrow=TRUE)
ai_precalc <- ai_precalc - T  # substracts Ti to the ith row of ai_precalc

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)
jn_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=0,ai_precalc=ai_precalc)

# the criterion takes a negative value, which is normal.
# See the Technometrics paper in the references

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

---

`jn_optim_parallel2` *Parallel jn criterion*

---

## Description

Evaluation of the parallel jn 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 expected `"jn"` uncertainty, which is the posterior expected variance of the excursion set's volume.

## Usage

```
jn_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,ai_precalc)
```

## 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 | Matrix of points for numerical integration. Two cases are handled. If the "jn" importance sampling distribution has been used, this is a (2p)*d matrix containing p couples of integration points in D x D, where D is the integration domain. If instead a p*d matrix is given, then the function will perform the integration by using all p^2 couples. |
| integration.weights | Vector of size p corresponding to the weights of these integration points. |
| intpoints.oldmean | Vector of size p, or 2p, corresponding to the kriging mean at the integration points before adding x to the design of experiments. |
| intpoints.oldsd | Vector of size p, or 2p, 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 | Array containing one or several thresholds. |
| 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) |
| ai_precalc | When multiple thresholds are used (i.e. when T is a vector), this is an nT*p matrix with ith row equal to intpoints.oldmean-T[i]. The argument DOES ALWAYS need to be filled, even if there is only one threshold T. |

## 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.

## Value

Parallel jn value

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

EGIparallel, max_sur_parallel

## Examples

```
#jn_optim_parallel2

set.seed(9)
N <- 20 #number of observations
T <- c(80,100) #threshold or thresholds
testfun <- branin

#a 20 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=200,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)
nT <- 2 # number of thresholds
ai_precalc <- matrix(rep(intpoints.oldmean,times=nT),
    nrow=nT,ncol=length(intpoints.oldmean),byrow=TRUE)
ai_precalc <- ai_precalc - T  # substracts Ti to the ith row of ai_precalc

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 jn_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)
jn_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,ai_precalc=ai_precalc)

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_parallel.grid <- apply(X=x,FUN=jn_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,ai_precalc=ai_precalc)
z.grid <- matrix(jn_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(jn_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 jn_parallel criterion (black) and of f(x)=T (blue)")
```

---

`max_futureVol_parallel` *Maximize parallel volume criterion*

---

## Description

Maximizes the criterion `vorobVol_optim_parallel`.

## Usage

```
max_futureVol_parallel(lower, upper, optimcontrol = NULL, batchsize,
  integration.param, T, model, new.noise.var = 0, typeEx = ">")
```

## Arguments

| | |
|---|---|
| lower | lower bounds of the domain |
| upper | upper bounds of the domain |
| optimcontrol | optional list of control parameters for optimization aspects, see max_vorob_parallel for details |
| batchsize | size of the batch of new points |
| 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 | threshold |
| model | a km Model |
| new.noise.var | Optional scalar with the noise variance at the new observation |
| typeEx | a character (">" or "<") identifying the type of excursion |

## Value

A list containing par, the best set of parameters found, value the value of the criterion and alpha, the Vorob'ev quantile corresponding to the conservative estimate.

## Author(s)

Dario Azzimonti (IDSIA, Switzerland)

## References

Azzimonti, D. and Ginsbourger, D. (2018). *Estimating orthant probabilities of high dimensional Gaussian vectors with an application to set estimation*. Journal of Computational and Graphical Statistics, 27(2), 255-267.

Azzimonti, D. (2016). *Contributions to Bayesian set estimation relying on random field priors*. PhD thesis, University of Bern.

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets*. Under revision. Preprint at hal-01379642

Chevalier, C., Bect, J., Ginsbourger, D., Vazquez, E., Picheny, V., and Richet, Y. (2014). *Fast kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*. Technometrics, 56(4):455-465.

## See Also

EGIparallel,max_vorob_parallel

## Examples

```
#max_futureVol_parallel

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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_futureVol_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(object=model,newX=obj$par,newy=apply(obj$par,1,testfun),
                      cov.reestim=TRUE)

  consLevel = 0.95; n_discrete_design=500*new.model@d
  CE_design=as.matrix (randtoolbox::sobol (n = n_discrete_design,
                               dim = new.model@d))
  colnames(CE_design) <- colnames(new.model@X)
  current.pred = predict.km(object = new.model,
                            newdata = CE_design,
                            type = "UK",cov.compute = TRUE)
  current.pred$cov <- current.pred$cov +1e-7*diag(nrow = nrow(current.pred$cov),
                                                  ncol = ncol(current.pred$cov))

  current.CE = anMC::conservativeEstimate(alpha = consLevel, pred=current.pred,
                                    design=CE_design, threshold=T, pn = NULL,
                                    type = ">", verb = 1,
                                    lightReturn = TRUE, algo = "GANMC")
```

```
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",consQuantile=obj$alpha)

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",consQuantile=current.CE$lvs)


## End(Not run)
```

---

`max_infill_criterion` *Optimizer for the infill criteria*

---

## Description

Optimization, 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 optimization using the genoud algorithm, or `"discrete"` for an optimization 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"`, `"tsee"`, or `"tmse"`. |
| `T` | Array containing one or several thresholds. The `"tmse"` criterion can be used with multiple thresholds. The `"ranjan"`, `"bichon"`, `"tsee"` criteria can be used with only one threshold. |
| `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 (INRA, Toulouse, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2012), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing vol. 22(3), pp 773-793

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

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

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

## See Also

EGI,ranjan_optim,tmse_optim,bichon_optim,tsee_optim

## Examples

```
#max_infill_criterion

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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)

## Not run:
obj <- max_infill_criterion(lower=lower,upper=upper,optimcontrol=optimcontrol,
                            method="bichon",T=T,model=model)

obj$par;obj$value
new.model <- update(object=model,newX=obj$par,newy=testfun(obj$par),cov.reestim=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"` *or* `"jn"` *criterion*

---

## Description

Minimization, based on the package rgenoud (or on exhaustive search on a discrete set), of the `"sur"` or `"jn"` 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,real.volume.variance=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), batch-size 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.

`real.volume.variance`
Optional argument to use the `"jn"` criterion instead of `"sur"`. Default: FALSE

## 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 (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

`EGIparallel`,`sur_optim_parallel`,`jn_optim_parallel`

## Examples

```
#max_sur_parallel


set.seed(9)
N <- 20 #number of observations
T <- c(40,80) #thresholds
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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 (hard) optim in dimension 5*2 !

obj$par;obj$value #optimum in 5 new points
new.model <- update(object=model,newX=obj$par,newy=apply(obj$par,1,testfun),
                      cov.reestim=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_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 | Array containing one or several thresholds. |
| 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 (INRA, Toulouse, France)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

Picheny V. (2009) *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. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

## See Also

EGIparallel,max_sur_parallel

## Examples

```
#max_timse_parallel


set.seed(9)
N <- 20 #number of observations
T <- c(40,80) #thresholds
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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(object=model,newX=obj$par,newy=apply(obj$par,1,testfun),
                        cov.reestim=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,
penalisation = NULL, typeEx = ">")
```

## 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), batch-size 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). The criterion CANNOT be used with multiple thresholds.

model — A Kriging model of km class.

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

penalisation — Optional penalization constant for type I errors. If equal to zero, computes the Type II criterion.

typeEx — A character (">" or "<") identifying the type of excursion

## 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 (University of Neuchatel, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

## References

Chevalier C., Ginsbouger D., Bect J., Molchanov I. (2013) *Estimating and quantifying uncertainties on level sets using the Vorob'ev expectation and deviation with gaussian process models* mODa 10, Advances in Model-Oriented Design and Analysis, Contributions to Statistics, pp 35-43

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

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets.* Under revision. Preprint at hal-01379642

## See Also

EGIparallel,max_sur_parallel

## Examples

```
#max_vorob_parallel

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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(object=model,newX=obj$par,newy=apply(obj$par,1,testfun),
                      cov.reestim=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 precomputations 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 (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

computeQuickKrigcov

## Examples

```
#precomputeUpdateData

set.seed(9)
N <- 20 #number of observations
testfun <- branin

#a 20 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)
integration.points <- expand.grid(x.grid,y.grid)
integration.points <- as.matrix(integration.points)
precalc.data <- precomputeUpdateData(model=model,integration.points=integration.points)

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

kn <- computeQuickKrigcov(model=model,integration.points=integration.points,X.new=newdata,
                    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 covariances, DOE points and new point
#these covariances have been computed quickly with computeQuickKrigcov
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(newdata, 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*

---

## Description

This function is similar to the predict.km function from the DiceKriging package. The only change is the additionnal F.newdata output.

## 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. |

## 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 (Ecole des Mines de St-Etienne, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

## 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, https://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(9)
N <- 20 #number of observations
testfun <- branin

#a 20 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)

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 kriging mean and stdev
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_parallel` *Quick update of kriging means and variances when one or many new points are added to the DOE.*

---

## Description

The functions uses the kriging update formulas 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 (University of Neuchatel, Switzerland)

## References

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

## See Also

computeQuickKrigcov, precomputeUpdateData

## Examples

```
#predict_update_km_parallel

set.seed(9)
N <- 20 #number of observations
testfun <- branin

#a 20 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)
newdata <- as.matrix(newdata)
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))

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 the value of a given measure of uncertainty. The function can be used to print relevant outputs after having used the function `EGI` or `EGIparallel`.

## Usage

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

## Arguments

| | |
|---|---|
| `model` | Kriging model of `km` class. |
| `T` | Array containing one or several thresholds. |
| `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 functions `print_uncertainty_1d`, `2d` or `nd`. |

## Value

the integrated uncertainty

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2012), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing vol. 22(3), pp 773-793

## See Also

print_uncertainty_1d,print_uncertainty_2d,print_uncertainty_nd

## Examples

```
#print_uncertainty

set.seed(9)
N <- 20 #number of observations
T <- c(80,100) #threshold
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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). The function can be used to print relevant outputs after having used the function EGI or EGIparallel.

## Usage

```
print_uncertainty_1d(model, T, type = "pn",
lower = 0, upper = 1, resolution = 500, new.points = 0,
xscale = c(0, 1), show.points = TRUE, 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,
vorobmean=FALSE,krigmeanplot=FALSE,Tplot=FALSE,consQuantile=NULL,...)
```

## Arguments

| | |
|---|---|
| model | Kriging model of km class. |
| T | Array containing one or several thresholds. |
| 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). |
| 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.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. |
| 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). |
| krigmeanplot | When set to TRUE a kriging mean is plotted on a second y axis. |
| Tplot | When set to TRUE, and if krigmeanplot is also set to TRUE, draws horizontal lines corresponding to the different thresholds T. |
| consQuantile | Optional value for plotting conservative quantiles. In order to plot<br>• Conservative estimates: consQuantile is a list containing at least consLevel (scalar), with the option typeEx (character, default = ">").<br>• Generic Vorob'ev quantiles: consQuantile is a scalar corresponding to the Vorob'ev quantile level. |
| ... | Additional arguments to the plot function. |

## Value

The integrated uncertainty. If the conservative estimate is computed, it also returns the conservative quantile level.

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

## References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2012), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing vol. 22(3), pp 773-793

## See Also

print_uncertainty_2d,print_uncertainty_nd

## Examples

```
#print_uncertainty_1d

set.seed(9)
N <- 9 #number of observations
T <- c(-0.2,0.2) #thresholds
testfun <- sin
lower <- c(0)
```

```
upper <- c(6)

#a 20 points initial design
design <- data.frame( lower+(upper-lower)*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(model=model,T=T,lower=lower,upper=upper,
  main="probability of excursion",xlab="x",ylab="pn",
  cex.points=1.5,col.points.init="red",
  krigmeanplot=TRUE,Tplot=TRUE)

## Not run:
uq1d <- print_uncertainty_1d(model=model,T=T,lower=lower,upper=upper,
  main="probability of excursion",xlab="x",ylab="pn",
  cex.points=1.5,col.points.init="red",
  krigmeanplot=TRUE,Tplot=TRUE,consQuantile =list(consLevel=0.95))

  print_uncertainty_1d(model=model,T=T,lower=lower,upper=upper,
  main="probability of excursion",xlab="x",ylab="pn",
  cex.points=1.5,col.points.init="red",
  krigmeanplot=TRUE,Tplot=TRUE,consQuantile =uq1d[2])

## End(Not run)
```

---

`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). The function can be used to print relevant outputs after having used the function `EGI` or `EGIparallel`.

## Usage

```
print_uncertainty_2d(model, T, type = "pn",
lower = c(0, 0), upper = c(1, 1), resolution = 200,
new.points = 0,
xscale = c(0, 1), yscale = c(0, 1), show.points = TRUE,
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,
krigmeanplot=FALSE,vorobmean=FALSE,consQuantile=NULL,...)
```

## Arguments

| | |
|---|---|
| model | Kriging model of km class. |
| T | Array containing one or several thresholds. |
| 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). |
| 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.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. |
| 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. |
| consQuantile | Optional value for plotting conservative quantiles. In order to plot<br>• Conservative estimates: consQuantile is a list containing at least consLevel (scalar), with the option typeEx (character, default = ">"). |

- Generic Vorob'ev quantiles: `consQuantile` is a scalar corresponding to the Vorob'ev quantile level.

`...` Additional arguments to the `image` function.

## Value

The integrated uncertainty. If the conservative estimate is computed, it also returns the conservative quantile level.

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

## References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2012), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing vol. 22(3), pp 773-793

## See Also

print_uncertainty_1d,print_uncertainty_nd

## Examples

```
#print_uncertainty_2d

set.seed(9)
N <- 20 #number of observations
T <- c(20,40) #thresholds
testfun <- branin
lower <- c(0,0)
upper <- c(1,1)

#a 20 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:
print_uncertainty_2d(model=model,T=T,main="probability of excursion",
                     type="pn",krigmeanplot=TRUE,vorobmean=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 and conservative estimate at level 0.95
# uq2d<- print_uncertainty_2d(model=model,T=T,main="probability of excursion",
#                     type="pn",krigmeanplot=TRUE,vorobmean=FALSE,
#                     consQuantile=list(consLevel=0.95))
# print_uncertainty_2d(model=model,T=T,main="probability of excursion",
#                     type="pn",krigmeanplot=TRUE,vorobmean=FALSE,
#                     consQuantile=uq2d[2])

## End(Not run)
```

---

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

---

## Description

This function draws projections on various plans of a given measure of uncertainty. The function can be used to print relevant outputs after having used the function `EGI` or `EGIparallel`.

## Usage

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

## Arguments

| | |
|---|---|
| `model` | Kriging model of `km` class. |
| `T` | Array containing one or several thresholds. |
| `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. The `"vorob"` option is not available due to the difficulty of computing a Vorob'ev threshold. |
| `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 do to computation time, except when the argument `"pairs"` is not set to its default value. |

| | |
|---|---|
| nintegpoints | to do |
| 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". |
| pairs | Optional argument. When set to codeNULL (default) the function performs the projections on plans spanned by each pair (i,j) of dimension. Otherwise, the argument is an array of size 2 corresponding to the dimensions spanning the (only) plan on which the projection is performed. |
| ... | Additional arguments to the image function. |

## Value

The integrated uncertainty

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2012), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing vol. 22(3), pp 773-793

## See Also

print_uncertainty_1d,print_uncertainty_2d

## Examples

```
#print_uncertainty_nd

set.seed(9)
N <- 30 #number of observations
T <- -1 #threshold
testfun <- hartman3
#The hartman3 function is defined over the domain [0,1]^3.

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

```
#a 30 points initial design
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 = 1)
```

## 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)

Victor Picheny (INRA, Toulouse, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

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

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, https://arxiv.org/abs/1009.5177

## See Also

EGI, max_infill_criterion

## Examples

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

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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_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 expected future sur uncertainty when the candidate points are added.

## 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, ai_precalc = NULL)
```

## Arguments

| | |
|---|---|
| `x` | 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` | Array containing one or several thresholds. |
| `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) |
| ai_precalc | When multiple thresholds are used (i.e. when T is a vector), this is an nT*p matrix with ith row equal to intpoints.oldmean-T[i]. The argument does not need to be filled if only one threshold is used. |

## 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).

## Value

Parallel sur value

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

EGIparallel, max_sur_parallel

## Examples

```
#sur_optim_parallel

set.seed(9)
N <- 20 #number of observations
T <- c(80,100) #thresholds
testfun <- branin

#a 20 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)
nT <- 2 # number of thresholds
ai_precalc <- matrix(rep(intpoints.oldmean,times=nT),
    nrow=nT,ncol=length(intpoints.oldmean),byrow=TRUE)
ai_precalc <- ai_precalc - T  # substracts Ti to the ith row of ai_precalc

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,ai_precalc=ai_precalc)


#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 expected future sur uncertainty when the candidate points are added.

## 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,ai_precalc = NULL)
```

## 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 | Array containing one or several thresholds. |
| 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) |
| ai_precalc | When multiple thresholds are used (i.e. when T is a vector), this is an nT*p matrix with ith row equal to intpoints.oldmean-T[i]. The argument does not need to be filled if only one threshold is used. |

## 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.

## Value

Parallel sur value

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

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

## See Also

EGIparallel, max_sur_parallel

## Examples

```
#sur_optim_parallel2

set.seed(9)
N <- 20 #number of observations
T <- c(80,100) #thresholds
testfun <- branin

#a 20 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)
nT <- 2 # number of thresholds
ai_precalc <- matrix(rep(intpoints.oldmean,times=nT),
    nrow=nT,ncol=length(intpoints.oldmean),byrow=TRUE)
ai_precalc <- ai_precalc - T  # substracts Ti to the ith row of ai_precalc
```

```
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,ai_precalc=ai_precalc)

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,ai_precalc=ai_precalc)
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_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 | Array containing one or several thresholds. |
| 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 (INRA, Toulouse, France)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

Picheny V. (2009) *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. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

## See Also

EGIparallel, max_timse_parallel

## Examples

```
#timse_optim_parallel

set.seed(9)
N <- 20 #number of observations
T <- c(80,100) #thresholds
testfun <- branin

#a 20 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
weight0 <- 1/sqrt( 2*pi*(intpoints.oldsd^2) )
weight <- 0
for(i in 1:length(T)){
  Ti <- T[i]
 weight <- weight + weight0 * exp(-0.5*((intpoints.oldmean-Ti)/sqrt(intpoints.oldsd^2))^2)
}

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 | Array containing one or several thresholds. |
| 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.

## Value

Parallel timse value

## Author(s)

Victor Picheny (INRA, Toulouse, France)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

Picheny V. (2009) *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. (2014), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, Technometrics, vol. 56(4), pp 455-465

## See Also

EGIparallel, max_timse_parallel

## Examples

```
#timse_optim_parallel2

set.seed(9)
N <- 20 #number of observations
```

```
T <- c(80,100) #thresholds
testfun <- branin

#a 20 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
weight0 <- 1/sqrt( 2*pi*(intpoints.oldsd^2) )
weight <- 0
for(i in 1:length(T)){
  Ti <- T[i]
 weight <- weight + weight0 * exp(-0.5*((intpoints.oldmean-Ti)/sqrt(intpoints.oldsd^2))^2)
}

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` | Array containing one or several thresholds. |
| `method.param` | Scalar tolerance around the targets 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)

Victor Picheny (INRA, Toulouse, France)

David Ginsbourger (IDIAP Martigny and University of Bern, Switzerland)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Picheny V., Ginsbourger D., Roustant O., Haftka R.T., (2010) *Adaptive designs of experiments for accurate approximation of a target region*, J. Mech. Des. vol. 132(7)

Picheny V. (2009) *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(9)
N <- 20 #number of observations
T <- c(40,80) #thresholds
testfun <- branin

#a 20 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 the 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 (University of Neuchatel, Switzerland)

Yann Richet (IRSN, France)

## See Also

`EGI`, `max_infill_criterion`

## Examples

```
#tsee_optim

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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)")
```

---

# vorobVol_optim_parallel

*Compute volume criterion*

---

## Description

Compute the volume criterion

## Usage

```
vorobVol_optim_parallel(x, integration.points, integration.weights = NULL,
  intpoints.oldmean, intpoints.oldsd, precalc.data, model, T,
  new.noise.var = NULL, batchsize, alpha, current.crit, typeEx = ">")
```

## Arguments

| | |
|---|---|
| x | vector of size batchsize*d describing the point(s) where to evaluate the criterion |
| integration.points | $p * d$ matrix with the integration points for evaluating numerically the integral of the criterion |
| integration.weights | vector of size $p$ containting the integration weights for the integral numerical evaluation |
| 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 | a km Model |
| T | threshold |
| new.noise.var | Optional scalar with the noise variance at the new observation |
| batchsize | size of the batch of new points |
| alpha | threshold on pn obtained with conservative estimates |
| current.crit | starting value for criterion |
| typeEx | a character (">" or "<") identifying the type of excursion |

## Value

The value of the criterion at x.

## Author(s)

Dario Azzimonti (IDSIA, Switzerland)

## References

Azzimonti, D. and Ginsbourger, D. (2018). *Estimating orthant probabilities of high dimensional Gaussian vectors with an application to set estimation*. Journal of Computational and Graphical Statistics, 27(2), 255-267.

Azzimonti, D. (2016). *Contributions to Bayesian set estimation relying on random field priors*. PhD thesis, University of Bern.

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets*. Under revision. Preprint at hal-01379642

Chevalier, C., Bect, J., Ginsbourger, D., Vazquez, E., Picheny, V., and Richet, Y. (2014). *Fast kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*. Technometrics, 56(4):455-465.

## See Also

EGIparallel, max_futureVol_parallel

## Examples

```
#vorobVol_optim_parallel

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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, the pn threshold should be computed with conservativeEstimate
# Here it is fixed at 0.992364
alpha <- 0.992364

## Not run:
  # You can compute it with the following code
  CE_design=as.matrix (randtoolbox::sobol (n = 500*model@d,
                                           dim = model@d))
  colnames(CE_design) <- colnames(model@X)

  CE_pred = predict.km(object = model, newdata = CE_design,
                       type = "UK",cov.compute = TRUE)
 CE_pred$cov <- CE_pred$cov +1e-7*diag(nrow = nrow(CE_pred$cov),ncol = ncol(CE_pred$cov))
```

```
  Cestimate <- anMC::conservativeEstimate(alpha = 0.95, pred=CE_pred,
                                          design=CE_design, threshold=T, pn = NULL,
                                          type = ">", verb = 1,
                                          lightReturn = TRUE, algo = "GANMC")
  alpha <- Cestimate$lvs

## End(Not run)

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)
vorobVol_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)


#the function max_futureVol_parallel will help to find the optimum:
#ie: the batch of 4 maximizing the expectation of the future
# uncertainty (future volume of the Vorob'ev quantile)
```

---

`vorobVol_optim_parallel2` *Compute volume criterion*

---

## Description

Compute the volume criterion. Useful for optimization routines.

## Usage

```
vorobVol_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.crit, typeEx = ">")
```

## Arguments

| | |
|---|---|
| x | One point where to evaluate the criterion |
| other.points | remaining points of the batch |
| integration.points | $p * d$ matrix with the integration points for evaluating numerically the integral of the criterion |
| integration.weights | vector of size $p$ containting the integration weights for the integral numerical evaluation |
| 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 | a km Model |
| T | threshold |
| new.noise.var | Optional scalar with the noise variance at the new observation |
| batchsize | size of the batch of new points |
| alpha | threshold on pn obtained with conservative estimates |
| current.crit | starting value for criterion |
| typeEx | a character (">" or "<") identifying the type of excursion |

## Value

The value of the criterion at x.

## Author(s)

Dario Azzimonti (IDSIA, Switzerland)

## References

Azzimonti, D. and Ginsbourger, D. (2018). *Estimating orthant probabilities of high dimensional Gaussian vectors with an application to set estimation*. Journal of Computational and Graphical Statistics, 27(2), 255-267.

Azzimonti, D. (2016). *Contributions to Bayesian set estimation relying on random field priors*. PhD thesis, University of Bern.

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets*. Under revision. Preprint at hal-01379642

Chevalier, C., Bect, J., Ginsbourger, D., Vazquez, E., Picheny, V., and Richet, Y. (2014). *Fast kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*. Technometrics, 56(4):455-465.

## See Also

EGIparallel, max_futureVol_parallel

## Examples

```
#vorobVol_optim_parallel2

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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, the pn threshold should be computed with conservativeEstimate
# Here it is fixed at 0.992364
alpha <- 0.992364

## Not run:
# You can compute it with the following code
CE_design=as.matrix (randtoolbox::sobol (n = 500*model@d,
                                         dim = model@d))
colnames(CE_design) <- colnames(model@X)

CE_pred = predict.km(object = model, newdata = CE_design,
                         type = "UK",cov.compute = TRUE)
CE_pred$cov <- CE_pred$cov +1e-7*diag(nrow = nrow(CE_pred$cov),ncol = ncol(CE_pred$cov))

Cestimate <- anMC::conservativeEstimate(alpha = 0.95, pred=CE_pred,
                          design=CE_design, threshold=T, pn = NULL,
                          type = ">", verb = 1,
                          lightReturn = TRUE, algo = "GANMC")
alpha <- Cestimate$lvs

## End(Not run)
```

```
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 vorobVol_optim_parallel criterion2
#we calculate the expectation of the future volume vorobev 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)
vorobVol_optim_parallel2(x=x,other.points=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)


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)
vorobVol_parallel.grid <- apply(X=x,FUN=vorobVol_optim_parallel2,MARGIN=1,other.points=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)
z.grid <- matrix(vorobVol_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)

# Note that we want to maximize this criterion.
i.best <- which.max(vorobVol_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 vorobVol_parallel criterion (black) and of f(x)=T (blue)")
```

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,
penalisation=NULL,typeEx=">")
```

## 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). The criterion CANNOT be used with multiple thresholds. |
| 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) |
| penalisation | Optional penalization constant for type I errors. If equal to zero, computes the Type II criterion. |
| typeEx | A character (">" or "<") identifying the type of excursion |

## 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).

## Value

Parallel vorob value

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

## References

Chevalier C., Ginsbouger D., Bect J., Molchanov I. (2013) *Estimating and quantifying uncertainties on level sets using the Vorob'ev expectation and deviation with gaussian process models* mODa 10, Advances in Model-Oriented Design and Analysis, Contributions to Statistics, pp 35-43

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

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2018). *Adaptive design of experiments for conservative estimation of excursion sets*. Under revision. Preprint at hal-01379642

## See Also

EGIparallel, max_vorob_parallel

## Examples

```
#vorob_optim_parallel

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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,
penalisation = NULL, typeEx = ">")
```

## 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). The criterion CANNOT be used with multiple thresholds. |
| 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) |
| penalisation | Optional penalization constant for type I errors. If equal to zero, computes the Type II criterion. |
| typeEx | A character (">" or "<") identifying the type of excursion |

## 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.

## Value

Parallel Vorob'ev value

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

Dario Azzimonti (IDSIA, Switzerland)

## References

Chevalier C., Ginsbouger D., Bect J., Molchanov I. (2013) *Estimating and quantifying uncertainties on level sets using the Vorob'ev expectation and deviation with gaussian process models* mODa 10, Advances in Model-Oriented Design and Analysis, Contributions to Statistics, pp 35-43

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

## See Also

EGIparallel, max_vorob_parallel

## Examples

```
#vorob_optim_parallel2

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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 to be equaly weighted.

## Value

a scalar: the Vorob'ev thresold

## Author(s)

Clement Chevalier (University of Neuchatel, Switzerland)

## References

Chevalier C., Ginsbouger D., Bect J., Molchanov I. (2013) *Estimating and quantifying uncertainties on level sets using the Vorob'ev expectation and deviation with gaussian process models* mODa 10, Advances in Model-Oriented Design and Analysis, Contributions to Statistics, pp 35-43

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

## See Also

max_vorob_parallel, vorob_optim_parallel

## Examples

```
#vorob_threshold

set.seed(9)
N <- 20 #number of observations
T <- 80 #threshold
testfun <- branin

#a 20 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