# Package 'profExtrema'

March 21, 2020

**Type** Package

**Title** Compute and Visualize Profile Extrema Functions

**Version** 0.2.1

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

**Date** 2020-03-20

**Description** Computes profile extrema functions for arbitrary functions. If the function is expensive-to-evaluate it computes profile extrema by emulating the function with a Gaussian process (using package 'DiceKriging'). In this case uncertainty quantification on the profile extrema can also be computed. The different plotting functions for profile extrema give the user a tool to better locate excursion sets.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 2.10), DiceKriging, KrigInv, pGPx

**Imports** microbenchmark, quantreg, lhs, splines, methods, RColorBrewer, utils, MASS, rcdd

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Author** Dario Azzimonti [aut, cre, cph] (<https://orcid.org/0000-0001-5080-3061>)

**Repository** CRAN

**Date/Publication** 2020-03-21 17:10:02 UTC

## R topics documented:

---

`approxMaxMin` *Approximate coordinate profile functions*

---

## Description

Evaluate profile extrema over other variables with approximations at few values

## Usage

```
approxMaxMin(f, fprime = NULL, d, opts = NULL)
```

## Arguments

| | |
|---|---|
| f | the function to be evaluated |
| fprime | derivative of the function |
| d | dimension of the input domain |
| opts | a list containing the options for this function and the subfunctions getMax, getMin or getMaxMinMC, see documentation of getMax, getMin, getMaxMinMC for details. The options only for approxMaxMin are |

- limits: an optional list with the upper and lower limits of each dimension, if NULL then for each dimension limits are 0,1
- smoother: Select which smoother to use:a string that selects which smoother to use:
  - "1order": first order interpolation with gradient
  - "splineSmooth": smoothing spline with default degrees of freedom (DEFAULT OPTION)
  - "quantSpline": profile inf and profile sup approximated with quantile spline regression at levels 0.1 and 0.9 respectively
- heavyReturn: If TRUE returns also all minimizers, default is FALSE.
- initDesign: The design of few points where the expensive sup is evaluated.
- fullDesignSize: The full design where the function is approximated.
- multistart: number of multistarts for optim procedure.
- MonteCarlo: if TRUE, computes sup with Monte Carlo procedure.
- numMCsamples: number of MC samples for the sup.
- plts: If TRUE, plots the max/min functions at each coordinate, default is FALSE.
- verb: If TRUE, outputs intermediate results, default is FALSE.

## Value

a list of two data frames (min, max) of the evaluations of $f_sup(x_i) = sup_{x_j \neq i} f(x_1, \ldots, x_d)$ and $f_inf(x_i) = inf_{x_j \neq i} f(x_1, \ldots, x_d)$ for each i at the design Design. By default Design is a 100 equally spaced points for each dimension. It can be changed by defining it in options$Design

## Author(s)

Dario Azzimonti

## Examples

```
if (!requireNamespace("DiceKriging", quietly = TRUE)) {
stop("DiceKriging needed for this example to work. Please install it.",
     call. = FALSE)
}
# Compute the coordinate profile extrema with full optimization on 2d example

# Define the function
```

```
g=function(x){
  return(-branin(x))
}
# Define the gradient
gprime = function(x){
  x1 = x[1]*15-5
  x2 = x[2]*15
  f1prime = (15*25)/(4*pi^4)*x1^3 - (15*75)/(2*pi^3)*x1^2 +
  (80*15)/(pi^2)*x1 - (5*15)/(pi^2)*x2*x1 +
  10*15/pi*x2 - 60*15/pi-10*15* (1 - 1/(8*pi))*sin(x1)
  f2prime = 2*15*(x2-5/(4*pi^2)*x1^2 +5/pi*x1-6)
  return(matrix(c(-f1prime,-f2prime),nrow=1))
}

# generic approximation options
init_des<-lhs::maximinLHS(15,2)
options_approx<- list(multistart=4,heavyReturn=TRUE,initDesign=init_des,fullDesignSize=100)

# 1order approximation
options_approx$smoother<-"1order"
coordProf_approx_1order<-approxMaxMin(f = g,fprime = gprime,d=2,opts = options_approx)

# quantile regression
options_approx$smoother<-"quantSpline"
coordProf_approx_quantReg<-approxMaxMin(f = g,fprime = gprime,d=2,opts = options_approx)



# Consider threshold=-10
threshold<- -10
# obtain the points where the profiles take the threshold value
pp_change<-getChangePoints(threshold = threshold,allRes = coordProf_approx_quantReg)
# evaluate g at a grid and plot the image
x<-seq(0,1,,100)
grid<-expand.grid(x,x)
g_evals<- apply(X = grid,MARGIN = 1,FUN = g)
image(x = x,y = x,z = matrix(g_evals,nrow = 100),col = grey.colors(20))
contour(x=x,y=x,z=matrix(g_evals,nrow = 100), add=TRUE, nlevels = 20)
contour(x=x,y=x,z=matrix(g_evals,nrow = 100), add=TRUE, levels = threshold,col=2)
abline(h = pp_change$neverEx$`-10`[[2]],col="darkgreen",lwd=2)
abline(v = pp_change$neverEx$`-10`[[1]],col="darkgreen",lwd=2)
# Plot the coordinate profiles and a threshold
plotMaxMin(allRes = coordProf_approx_1order,threshold = threshold,changes = TRUE)
plotMaxMin(allRes = coordProf_approx_quantReg,threshold = threshold,changes = TRUE)
```

---

`approxProfileExtrema` *Approximate profile extrema functions*

---

## Description

Evaluate profile extrema for a set of Psi with approximations at few values

## Usage

```
approxProfileExtrema(f, fprime = NULL, d, allPsi, opts = NULL)
```

## Arguments

| | |
|---|---|
| `f` | the function to be evaluated |
| `fprime` | derivative of the function |
| `d` | dimension of the input domain |
| `allPsi` | a list containing the matrices Psi (dim $pxd$) for which to compute the profile extrema |
| `opts` | a list containing the options for this function and the subfunctions getProfileSup_optim, getProfileInf_optim or getProfileExtrema. The options only for approxProfileExtrema are |

- `limits`: an optional list with the upper and lower limits of input space dimension, if NULL then `limits=list(upper=rep(1,d),lower=rep(0,d))`
- `smoother`: Select which smoother to use:a string that selects which smoother to use:
  - `"1order"`: first order interpolation with gradient
  - `"splineSmooth"`: smoothing spline with default degrees of freedom (DEFAULT OPTION)
  - `"quantSpline"`: profile inf and profile sup approximated with quantile spline regression at levels 0.1 and 0.9 respectively
- `heavyReturn`: If TRUE returns also all minimizers, default is FALSE.
- `initDesign`: A list of the same length as allPsi containing the designs of few points where the expensive sup is evaluated. If Null it is automatically initialized
- `fullDesignSize`: The full design where the function is approximated.
- `multistart`: number of multistarts for optim procedure.
- `numMCsamples`: number of MC samples for the sup.
- `plts`: If TRUE, plots the max/min functions at each coordinate, default is FALSE.
- `verb`: If TRUE, outputs intermediate results, default is FALSE.

## Value

a list of two data frames (min, max) of the evaluations of $f_sup(x_i) = sup_{x_j \neq i} f(x_1, \ldots, x_d)$ and $f_inf(x_i) = inf_{x_j \neq i} f(x_1, \ldots, x_d)$ for each i at the design Design. By default Design is a 100 equally spaced points for each dimension. It can be changed by defining it in options$Design

## Author(s)

Dario Azzimonti

## Examples

```
# Compute the oblique profile extrema with approximate optimization on 2d example

# Define the function
testF <- function(x,params,v1=c(1,0),v2=c(0,1)){
return(sin(crossprod(v1,x)*params[1]+params[2])+cos(crossprod(v2,x)*params[3]+params[4])-1.5)
}

testFprime <- function(x,params,v1=c(1,0),v2=c(0,1)){
  return(matrix(c(params[1]*v1[1]*cos(crossprod(v1,x)*params[1]+params[2])-
                  params[3]*v2[1]*sin(crossprod(v2,x)*params[3]+params[4]),
                 params[1]*v1[2]*cos(crossprod(v1,x)*params[1]+params[2])-
                  params[3]*v2[2]*sin(crossprod(v2,x)*params[3]+params[4])),ncol=1))
}


# Define the main directions of the function
theta=pi/6
pparams<-c(1,0,10,0)
vv1<-c(cos(theta),sin(theta))
vv2<-c(cos(theta+pi/2),sin(theta+pi/2))

# Define optimizer friendly function
f <-function(x){
return(testF(x,pparams,vv1,vv2))
}
fprime <- function(x){
 return(testFprime(x,pparams,vv1,vv2))
}

# Define list of directions where to evaluate the profile extrema
all_Psi <- list(Psi1=vv1,Psi2=vv2)

# Evaluate profile extrema along directions of all_Psi
allOblique<-approxProfileExtrema(f=f,fprime = fprime,d = 2,allPsi = all_Psi,
                                 opts = list(plts=FALSE,heavyReturn=TRUE))


# Consider threshold=0
threshold <- 0

# Plot oblique profile extrema functions
plotMaxMin(allOblique,allOblique$Design,threshold = threshold)

## Since the example is two dimensional we can visualize the regions excluded by the profile extrema
# evaluate the function at a grid for plots
inDes<-seq(0,1,,100)
inputs<-expand.grid(inDes,inDes)
outs<-apply(X = inputs,MARGIN = 1,function(x){return(testF(x,pparams,v1=vv1,v2=vv2))})

# obtain the points where the profiles take the threshold value
cccObl<-getChangePoints(threshold = threshold,allRes = allOblique,Design = allOblique$Design)
```

```
# visualize the functions and the regions excluded

image(inDes,inDes,matrix(outs,ncol=100),col=grey.colors(20),main="Example and oblique profiles")
contour(inDes,inDes,matrix(outs,ncol=100),add=TRUE,nlevels = 20)
contour(inDes,inDes,matrix(outs,ncol=100),add=TRUE,levels = c(threshold),col=4,lwd=1.5)
plotOblique(cccObl$alwaysEx$`0`[[1]],all_Psi[[1]],col=3)
plotOblique(cccObl$alwaysEx$`0`[[2]],all_Psi[[2]],col=3)
plotOblique(cccObl$neverEx$`0`[[1]],all_Psi[[1]],col=2)
plotOblique(cccObl$neverEx$`0`[[2]],all_Psi[[2]],col=2)
```

---

`bound_profiles` *Bound for profile extrema quantiles*

---

## Description

The function bound_profiles computes the upper and lower bounds for the profile extrema quantiles of a Gaussian process model.

## Usage

```
bound_profiles(objectUQ, mean_var_delta = NULL, beta = 0.0124,
  alpha = 0.025, allPsi = NULL, options_approx = NULL,
  options_full_sims = NULL)
```

## Arguments

| | |
|---|---|
| `objectUQ` | an object returned by coordProf_UQ or the object saved in `obj$res_UQ`, if `obj` is the object returned by coordinateProfiles |
| `mean_var_delta` | the profile extrema functions at `options_approx$design` for the mean and variance function of the difference process $Z^{\Delta} = Z_x - \widetilde{Z}_x$. Object returned by prof_mean_var_Delta. |
| `beta` | the level of confidence for the approximate simulations |
| `alpha` | the level of confidence for the bound |
| `allPsi` | optional list of matrices (dim $pxd$) for which to compute the profile extrema. If NULL coordinate profiles are computed. |
| `options_approx` | an optional list of options for approxMaxMin (or approxProfileExtrema if `allPsi` not NULL). |
| `options_full_sims` | an optional list of options for getAllMaxMin (or getProfileExtrema if `allPsi` not NULL). If NULL the full computations are not excuted. NOTE: this computations might be very expensive! |

### Value

a list containing

- `bound`: a list containing the upper/lower bound for profile sup and inf
- `approx`: a list containing the upper/lower approximate quantiles for profile sup and inf

### Author(s)

Dario Azzimonti

---

## `cleanProfileResults` *Clean a profile extrema object*

---

### Description

The function cleanProfileResults cleans a profile extrema object to partially redo some computations.

### Usage

```
cleanProfileResults(object, level = 1)
```

### Arguments

| | |
|---|---|
| `object` | a list containing profile extrema results. |
| `level` | an integer 1-4 denoting how much it should be removed from `object`. See Value for details. |

### Value

returns `object` with the deleted parts as selected by `level`. In particular

- `1`: keep only `profMean_full`.
- `2`: keep `profMean_full` and `profMean_approx`. Remove all UQ results.
- `3`: keep `profMean_full` and `profMean_approx` and the pilot points. Remove all UQ simulations.
- `4`: Remove only the bound computations.

### Author(s)

Dario Azzimonti

## Examples

```
if (!requireNamespace("DiceKriging", quietly = TRUE)) {
stop("DiceKriging needed for this example to work. Please install it.",
     call. = FALSE)
}
# Compute a kriging model from 50 evaluations of the Branin function
# Define the function
g=function(x){
  return(-branin(x))
}
gp_des<-lhs::maximinLHS(20,2)
reals<-apply(gp_des,1,g)
kmModel<-km(design = gp_des,response = reals,covtype = "matern3_2")

threshold=-10

# Compute coordinate profiles on the posterior mean
# Increase multistart and size of designs for more precise results
options_full<-list(multistart=2,heavyReturn=TRUE, Design = replicate(2,seq(0,1,,50)))
init_des<-lhs::maximinLHS(12,2)
options_approx<- list(multistart=2,heavyReturn=TRUE,initDesign=init_des,fullDesignSize=50)
cProfilesMean<-coordinateProfiles(object=kmModel,threshold=threshold,options_full=options_full,
                                  options_approx=options_approx,uq_computations=FALSE,
                            plot_level=3,plot_options=NULL,CI_const=NULL,return_level=2)

# If we want to run again the computation of approximate coordinate profiles
# we delete that result and run again the coordinate profiles function
cProfiles_full <- cleanProfileResults(cProfilesMean,level=1)
## Not run:
# Coordinate profiles with UQ with approximate profiles
plot_options<-list(save=FALSE, titleProf = "Coordinate profiles",
                   title2d = "Posterior mean",qq_fill=TRUE)
cProfilesUQ<-coordinateProfiles(object=cProfilesMean,threshold=threshold,options_full=options_full,
                                  options_approx=options_approx,uq_computations=TRUE,
                            plot_level=3,plot_options=NULL,CI_const=NULL,return_level=2)
# If we would like to remove all UQ results
cProf_noUQ <- cleanProfileResults(cProfilesUQ,level=2)

# If we would like to remove the simulations but keep the pilot points
cProf_noSims <- cleanProfileResults(cProfilesUQ,level=3)
# the line above is useful, for example, if we need a more accurate UQ. In that case
# we obtain more simulations with the same pilot points and then combine the results.


## End(Not run)
```

---

`coastal_flooding` *Coastal flooding as function of offshore forcing conditions.*

---

## Description

A dataset containing the results of a numerical simulation conducted with the MARS model (Lazure and Dumas, 2008) for coastal flooding. The numerical model was adapted to the Boucholeurs area (French Atlantic coast), close to La Rochelle, and validated with data from the 2010 Xynthia storm event. See Azzimonti et al. (2017+) and Rohmer et al. (2018) for more details.

## Usage

```
coastal_flooding
```

## Format

A data frame with 200 rows and 6 variables:

**Tide** High tide level in meters;

**Surge** Surge peak amplitude in meters;

**phi** Phase difference between high tide and surge peak;

**t-** Duration of the increasing part of the surge temporal signal (assumed to be triangular);

**t+** Duration of the decreasing part of the surge temporal signal (assumed to be triangular);

**Area** Flooded area in m^2.

## Details

The data frame contains 5 input variables: `Tide`, `Surge`, `phi`, `t-`, `t+` detailing the offshore forcing conditions for the model. All input variables are normalized in $[0, 1]$. The response is `Area`, the area flooded in m^2.

## References

Azzimonti, D., Ginsbourger, D., Rohmer, J. and Idier, D. (2017+). *Profile extrema for visualizing and quantifying uncertainties on excursion regions. Application to coastal flooding.* arXiv:1710.00688.

Rohmer, J., Idier, D., Paris, F., Pedreros, R., and Louisor, J. (2018). *Casting light on forcing and breaching scenarios that lead to marine inundation: Combining numerical simulations with a random-forest classification approach.* Environmental Modelling & Software, 104:64-80.

Lazure, P. and Dumas, F. (2008). *An external-internal mode coupling for a 3D hydrodynamical model for applications at regional scale (MARS).* Advances in Water Resources, 31:233-250.

## Examples

```
# Define inputs
inputs<-data.frame(coastal_flooding[,-6])
colnames(inputs)<-colnames(coastal_flooding[,-6])
colnames(inputs)[4:5]<-c("tPlus","tMinus")

# put response in areaFlooded variable
areaFlooded<-data.frame(coastal_flooding[,6])
colnames(areaFlooded)<-colnames(coastal_flooding)[6]
response = sqrt(areaFlooded)
```

```
model <- km(formula=~Tide+Surge+I(phi^2)+tMinus+tPlus,
            design = inputs,response = response,covtype="matern3_2")
# Fix threshold
threshold<-sqrt(c(1.2e6,1.9e6,3.1e6,6.5e6))

# use the coordinateProfile function
## set up plot options
options_plots <- list(save=FALSE, folderPlots = "./" ,
                      titleProf = "Coordinate profiles",
                      title2d = "Posterior mean",qq_fill=TRUE)
# set up full profiles options
options_full<-list(multistart=15,heavyReturn=TRUE)
# set up approximation options
d <- model@d
init_des<-lhs::maximinLHS(5*d , d )
options_approx<- list(multistart=2,heavyReturn=TRUE, initDesign=init_des,
                      fullDesignSize=100, smoother="quantSpline")

# run the coordinate profile extrema on the mean
CF_CoordProf_mean<- coordinateProfiles(object = model, threshold = threshold,
                               uq_computations = FALSE, options_approx = options_approx,
                              plot_level=3, plot_options= options_plots, return_level=3,
                                      options_full=options_full)

## Not run:
## UQ computations might require a long time
# set up simulation options
## reduce nsims and batchsize for faster/less accurate UQ
nsims=200
opts_sims<-list(algorithm="B", lower=rep(0,d ),
                upper=rep(1,d ), batchsize=150,
                optimcontrol=list(method="genoud", pop.size=100,print.level=0),
                integcontrol = list(distrib="sobol",n.points=2000),nsim=nsims)

opts_sims$integration.param <- integration_design(opts_sims$integcontrol,
                                                  d , opts_sims$lower,
                                                  opts_sims$upper,
                                                  model,threshold)
opts_sims$integration.param$alpha <- 0.5

# run UQ computations
CF_CoordProf_UQ<- coordinateProfiles(object = CF_CoordProf_mean, threshold = threshold,
                                 uq_computations = TRUE, options_approx = options_approx,
                                plot_level=3, plot_options= options_plots, return_level=3,
                                      options_sims=opts_sims,options_full=options_full,
                                      options_bound = list(beta=0.024,alpha=0.05))

## End(Not run)
```

`coordinateProfiles` *Coordinate profiles starting from a kriging model*

## Description

The function coordinateProfiles computes the profile extrema functions for the posterior mean of a Gaussian process and its confidence bounds

## Usage

```
coordinateProfiles(object, threshold, options_full = NULL,
  options_approx = NULL, uq_computations = FALSE, plot_level = 0,
  plot_options = NULL, CI_const = NULL, return_level = 1, ...)
```

## Arguments

| | |
|---|---|
| `object` | either a km model or a list containing partial results. If `object` is a km model then all computations are carried out. If `object` is a list, then the function carries out all computations to complete the list results. |
| `threshold` | the threshold of interest |
| `options_full` | an optional list of options for getAllMaxMin, see getAllMaxMin for details. |
| `options_approx` | an optional list of options for approxMaxMin, see approxMaxMin for details. |
| `uq_computations` | boolean, if TRUE the uq computations for the profile mean are computed. |
| `plot_level` | an integer to select the plots to return (0=no plots, 1=basic plots, 2= all plots) |
| `plot_options` | an optional list of parameters for plots. See setPlotOptions for currently available options. |
| `CI_const` | an optional vector containing the constants for the CI. If not NULL, then profiles extrema for $m_n(x) \pm CI_const[i] * s_n(x,x)$ are computed. |
| `return_level` | an integer to select the amount of details returned |
| `...` | additional parameters to be passed to coordProf_UQ. |

## Value

If return_level=1 a list containing

- `profMean_full`: the results of `getAllMaxMin` for the posterior mean
- `profMean_approx`: the results of `approxMaxMin` for the posterior mean
- `res_UQ`: the results of `coordProf_UQ` for the posterior mean

if return_level=2 the same list as above but also including

- `abs_err`: the vector of maximum absolute approximation errors for the profile inf /sup on posterior mean for the chosen approximation
- `times`: a list containing
  - `full`: computational time for the full computation of profile extrema
  - `approx`: computational time for the approximate computation of profile extrema

## Author(s)

Dario Azzimonti

## Examples

```
if (!requireNamespace("DiceKriging", quietly = TRUE)) {
stop("DiceKriging needed for this example to work. Please install it.",
     call. = FALSE)
}
# Compute a kriging model from 50 evaluations of the Branin function
# Define the function
g=function(x){
  return(-branin(x))
}
gp_des<-lhs::maximinLHS(20,2)
reals<-apply(gp_des,1,g)
kmModel<-km(design = gp_des,response = reals,covtype = "matern3_2")

threshold=-10

# Compute coordinate profiles on the posterior mean
# Increase multistart and size of designs for more precise results
options_full<-list(multistart=2,heavyReturn=TRUE, Design = replicate(2,seq(0,1,,50)))
init_des<-lhs::maximinLHS(12,2)
options_approx<- list(multistart=2,heavyReturn=TRUE,initDesign=init_des,fullDesignSize=50)
cProfilesMean<-coordinateProfiles(object=kmModel,threshold=threshold,options_full=options_full,
                                  options_approx=options_approx,uq_computations=FALSE,
                            plot_level=3,plot_options=NULL,CI_const=NULL,return_level=2)
## Not run:
# Coordinate profiles with UQ with approximate profiles
plot_options<-list(save=FALSE, titleProf = "Coordinate profiles",
                   title2d = "Posterior mean",qq_fill=TRUE)
cProfilesUQ<-coordinateProfiles(object=cProfilesMean,threshold=threshold,options_full=options_full,
                                  options_approx=options_approx,uq_computations=TRUE,
                            plot_level=3,plot_options=NULL,CI_const=NULL,return_level=2)

# Coordinate profiles with UQ with fully optim profiles
options_full_sims<-list(multistart=4,heavyReturn=TRUE, Design = replicate(2,seq(0,1,,60)))
cProfilesUQ<-coordinateProfiles(object=cProfilesMean,threshold=threshold,options_full=options_full,
                                  options_approx=options_approx,uq_computations=TRUE,
                            plot_level=3,plot_options=NULL,CI_const=NULL,return_level=2,
                                  options_full_sims=options_full_sims)

## End(Not run)
```

---

coordProf_UQ *Coordinate profiles UQ from a kriging model*

---

## Description

The function coordProf_UQ computes the profile extrema functions for posterior realizations of a Gaussian process and its confidence bounds

## Usage

```
coordProf_UQ(object, threshold, allResMean = NULL,
  quantiles_uq = c(0.05, 0.95), options_approx = NULL,
  options_full_sims = NULL, options_sims = NULL,
  options_bound = NULL, plot_level = 0, plot_options = NULL,
  return_level = 1)
```

## Arguments

| | |
|---|---|
| `object` | either a km model or a list containing partial results. If `object` is a km model then all computations are carried out. If `object` is a list, then the function carries out all computations to complete the results list. |
| `threshold` | the threshold of interest |
| `allResMean` | a list resulting from `getAllMaxMin` or `approxMaxMin` for the profile extrema on the mean. If NULL the median from the observations is plotted |
| `quantiles_uq` | a vector containing the quantiles to be computed |
| `options_approx` | an optional list of options for approxMaxMin, see approxMaxMin for details. |
| `options_full_sims` | an optional list of options for getAllMaxMin, see getAllMaxMin for details. If NULL the full computations are not excuted. NOTE: this computations might be very expensive! |
| `options_sims` | an optional list of options for the posterior simulations.<br>• `algorithm`: string choice of the algorithm to select the pilot points ("A" or "B", default "B");<br>• `lower`: $d$ dimensional vector with lower bounds for pilot points, default `rep(0,d)`;<br>• `upper`: $d$ dimensional vector with upper bounds for pilot points, default `rep(1,d)`;<br>• `batchsize`: number of pilot points, default `120`;<br>• `optimcontrol`: list containing the options for optimization, see optim_dist_measure;<br>• `integcontrol`: list containing the options for numerical integration of the criterion, see optim_dist_measure;<br>• `integration.param`: list containing the integration design, obtained with the function integration_design;<br>• `nsim`: number of approximate GP simulations, default `300`. |
| `options_bound` | an optional list containing `beta` the confidence level for the approximation and `alpha` the confidence level for the bound. Note that `alpha > 2*beta`. If `NULL`, the bound is not computed. |
| `plot_level` | an integer to select the plots to return (0=no plots, 1=basic plots, 2= all plots) |

| | |
|---|---|
| plot_options | an optional list of parameters for plots. See setPlotOptions for currently available options. |
| return_level | an integer to select the amount of details returned |

## Value

If return_level=1 a list containing

- profSups: an array dxfullDesignSizexnsims containing the profile sup for each coordinate for each realization.
- profInfs: an array dxfullDesignSizexnsims containing the profile inf for each coordinate for each realization.
- prof_quantiles_approx: a list containing the quantiles (levels set by quantiles_uq) of the profile extrema functions.

if return_level=2 the same list as above but also including more: a list containing

- times: a list containing
  - tSpts: computational time for selecting pilot points.
  - tApprox1ord: vector containing the computational time required for profile extrema computation for each realization
- simuls: a matrix containing the value of the field simulated at the pilot points
- sPts: the pilot points

## Author(s)

Dario Azzimonti

## Examples

```
if (!requireNamespace("DiceKriging", quietly = TRUE)) {
stop("DiceKriging needed for this example to work. Please install it.",
     call. = FALSE)
}
# Compute a kriging model from 50 evaluations of the Branin function
# Define the function
g<-function(x){
  return(-branin(x))
}
gp_des<-lhs::maximinLHS(20,2)
reals<-apply(gp_des,1,g)
kmModel<-km(design = gp_des,response = reals,covtype = "matern3_2")

threshold=-10
d<-2

# Compute coordinate profiles UQ starting from GP model
# define simulation options
options_sims<-list(algorithm="B", lower=rep(0,d), upper=rep(1,d),
             batchsize=80, optimcontrol = list(method="genoud",pop.size=100,print.level=0),
```

```
                    integcontrol = list(distrib="sobol",n.points=1000), nsim=150)
# define 1 order approximation options
init_des<-lhs::maximinLHS(15,d)
options_approx<- list(multistart=4,heavyReturn=TRUE,
                      initDesign=init_des,fullDesignSize=100,
                      smoother="1order")
# define plot options
options_plots<-list(save=FALSE, titleProf = "Coordinate profiles",
                    title2d = "Posterior mean",qq_fill=TRUE)
## Not run:
# profile UQ on approximate coordinate profiles
cProfiles_UQ<-coordProf_UQ(object = kmModel,threshold = threshold,allResMean = NULL,
                           quantiles_uq = c(0.05,0.95),options_approx = options_approx,
                           options_full_sims = NULL,options_sims = options_sims,
                           options_bound = NULL,plot_level = 3,
                           plot_options = options_plots,return_level = 3)
# profile UQ on full optim coordinate profiles
options_full_sims<-list(multistart=4,heavyReturn=TRUE)
cProfiles_UQ_full<-coordProf_UQ(object = cProfiles_UQ,threshold = threshold,allResMean = NULL,
                           quantiles_uq = c(0.05,0.95),options_approx = options_approx,
                         options_full_sims = options_full_sims,options_sims = options_sims,
                           options_bound = NULL,plot_level = 3,
                           plot_options = options_plots,return_level = 3)

# profile UQ on full optim coordinate profiles with bound
cProfiles_UQ_full_bound<-coordProf_UQ(object = cProfiles_UQ_full,threshold = threshold,
                                      allResMean = NULL, quantiles_uq = c(0.05,0.95),
                                      options_approx = options_approx,
                                      options_full_sims = options_full_sims,
                                      options_sims = options_sims,
                                      options_bound = list(beta=0.024,alpha=0.05),
                                      plot_level = 3, plot_options = options_plots,
                                      return_level = 3)

## End(Not run)
```

---

`getAllMaxMin` *Coordinate profile extrema with BFGS optimization*

---

## Description

Evaluate coordinate profile extrema with full optimization.

## Usage

```
getAllMaxMin(f, fprime = NULL, d, options = NULL)
```

## Arguments

| | |
|---|---|
| f | the function to be evaluated |
| fprime | derivative of the function |
| d | dimension of the input domain |
| options | a list containing the options for this function and the subfunctions getMax, getMin see documentation of getMax, getMin for details. The options only for getAllMaxMin are<br>• Design: an optional design matrix with the discretization of each dimension, if NULL then for each dimension Design[,coord] = seq(0,1,length.out=100)<br>• heavyReturn: If TRUE returns also all minimizers, default is FALSE.<br>• plts: If TRUE, plots the max/min functions at each coordinate, default is FALSE.<br>• verb: If TRUE, outputs intermediate results, default is FALSE.<br>• MonteCarlo: If TRUE, use the MC optimizer otherwise use standard optim. |

## Value

a list of two data frames (min, max) of the evaluations of $f_sup(x_i) = sup_{x_j \neq i} f(x_1, \ldots, x_d)$ and $f_inf(x_i) = inf_{x_j \neq i} f(x_1, \ldots, x_d)$ for each i at the design Design. By default Design is a 100 equally spaced points for each dimension. It can be changed by defining it in options$Design

## Author(s)

Dario Azzimonti

## Examples

```
if (!requireNamespace("DiceKriging", quietly = TRUE)) {
stop("DiceKriging needed for this example to work. Please install it.",
     call. = FALSE)
}
# Compute the coordinate profile extrema with full optimization on 2d example

# Define the function
g=function(x){
  return(-branin(x))
}
# Define the gradient
gprime = function(x){
  x1 = x[1]*15-5
  x2 = x[2]*15
  f1prime = (15*25)/(4*pi^4)*x1^3 - (15*75)/(2*pi^3)*x1^2 +
  (80*15)/(pi^2)*x1 - (5*15)/(pi^2)*x2*x1 +
  10*15/pi*x2 - 60*15/pi-10*15* (1 - 1/(8*pi))*sin(x1)
  f2prime = 2*15*(x2-5/(4*pi^2)*x1^2 +5/pi*x1-6)
  return(c(-f1prime,-f2prime))
}
# set up dimension
coordProf<-getAllMaxMin(f = g,fprime = gprime,d=2,options = list(multistart=4,heavyReturn=TRUE))
```

```
# Consider threshold=-10
threshold<- -10
# obtain the points where the profiles take the threshold value
pp_change<-getChangePoints(threshold = threshold,allRes = coordProf)
# evaluate g at a grid and plot the image
x<-seq(0,1,,100)
grid<-expand.grid(x,x)
g_evals<- apply(X = grid,MARGIN = 1,FUN = g)
image(x = x,y = x,z = matrix(g_evals,nrow = 100),col = grey.colors(20))
contour(x=x,y=x,z=matrix(g_evals,nrow = 100), add=TRUE, nlevels = 20)
contour(x=x,y=x,z=matrix(g_evals,nrow = 100), add=TRUE, levels = threshold,col=2)
abline(h = pp_change$neverEx$`-10`[[2]],col="darkgreen",lwd=2)
abline(v = pp_change$neverEx$`-10`[[1]],col="darkgreen",lwd=2)
# Plot the coordinate profiles and a threshold
plotMaxMin(allRes = coordProf,threshold = threshold,changes = TRUE)
```

---

`getChangePoints` *Coordinate profiles crossing points*

---

## Description

Obtain the points where the coordinate profile extrema functions cross the threshold

## Usage

```
getChangePoints(threshold, Design = NULL, allRes)
```

## Arguments

| | |
|---|---|
| `threshold` | if not null plots the level |
| `Design` | a d dimensional design corresponding to the points |
| `allRes` | list containing the list `res` which contains the computed minima and maxima. The object returned by the function `getAllMaxMin`. |

## Value

returns a list containing two lists with d components where

- alwaysEx: each component is a numerical vector indicating the points $x_i$ where $inf_{x^{-i}} f(x) >$ `threshold`;
- neverEx: each component is a numerical vector indicating the points $x_i$ where $sup_{x^{-i}} f(x) <$ `threshold`.

## Author(s)

Dario Azzimonti

## getClosePoints *Find close points*

### Description

Obtain points close in one specific dimension

### Usage

```
getClosePoints(x, allPts, whichDim)
```

### Arguments

| | |
|---|---|
| x | one dimensional point |
| allPts | dataframe containing a list of d dimensional points |
| whichDim | integer defining the dimension of x |

### Value

the index in allPts (row number) of the closest point in allPts to x along the whichDim dimension

### Author(s)

Dario Azzimonti

## getMax *Coordinate profile sup function*

### Description

Compute coordinate profile sup functions

### Usage

```
getMax(x, f, fprime, coord, d, options = NULL)
```

### Arguments

| | |
|---|---|
| x | one dimensional point where the function is to be evaluated |
| f | function to be optimized (takes a vector y of dimension d and returns a real number) |
| fprime | derivative of f (same format) |
| coord | integer selecting the dimension that is fixed, the other ones are optimized over |
| d | dimension of the input for f |

options — a list containing the options to be passed to optim:

- par: contains the starting point (a point in dimension d-1)
- method: is the string denoting the chosen method for the optimization (see optim for details)
- lower: the lower bounds for the optimization domain (see optim for details)
- upper: the upper bounds for the optimization domain (see optim for details)

## Value

a real value corresponding to $max_{x_1,\ldots,x_{coord-1},x_{coord+1},\ldots,x_d} f(x_1,\ldots,x_d)$

## Author(s)

Dario Azzimonti

---

# getMaxMinMC *Coordinate profile extrema with MC*

---

## Description

Compute coordinate profile extrema with MC

## Usage

```
getMaxMinMC(x, f, fprime, coord, d, options = NULL)
```

## Arguments

x — one dimensional point where the function is to be evaluated

f — function to be optimized (takes a vector y of dimension d and returns a real number)

fprime — derivative of f (same format)

coord — integer selecting the dimension that is fixed, the other ones are optimized over

d — dimension of the input for f

options — a list containing the options to be passed to the MC optimizer:

- par: contains the starting point (a point in dimension d-1)
- numMCsamples: number of MC samples
- randstring that chooses the type of randomness in MC: "unif" (uniform in [lower,upper]), "norm" (independent normal with mean 0 and variance 1)
- lower: the lower bounds for the optimization domain (see optim for details)
- upper: the upper bounds for the optimization domain (see optim for details)

## Value

a real value corresponding to $max_{x_1,\ldots,x_{coord-1},x_{coord+1},\ldots,x_d} f(x_1,\ldots,x_d)$

## Author(s)

Dario Azzimonti

---

`getMin` *Coordinate profile inf function*

---

## Description

Compute coordinate profile inf functions

## Usage

```
getMin(x, f, fprime, coord, d, options = NULL)
```

## Arguments

| | |
|---|---|
| `x` | one dimensional point where the function is to be evaluated |
| `f` | function to be optimized (takes a vector y of dimension d and returns a real number) |
| `fprime` | derivative of f (same format) |
| `coord` | integer selecting the dimension that is fixed, the other ones are optimized over |
| `d` | dimension of the input for f |
| `options` | a list containing the options to be passed to optim:<br>• `par`: contains the starting point (a point in dimension d-1)<br>• `method`: is the string denoting the chosen method for the optimization (see optim for details)<br>• `lower`: the lower bounds for the optimization domain (see optim for details)<br>• `upper`: the upper bounds for the optimization domain (see optim for details) |

## Value

a real value corresponding to $min_{x_1,\ldots,x_{coord-1},x_{coord+1},\ldots,x_d} f(x_1,\ldots,x_d)$

## Author(s)

Dario Azzimonti

## getPointProportion *Obtain proportion of true observations in excursion set*

### Description

Computes the proportion of observations in the excursion set from true function evaluations, binned by the grid determined with `xBins`, `yBins`.

### Usage

```
getPointProportion(pp, xBins, yBins, whichAbove, plt = FALSE)
```

### Arguments

| | |
|---|---|
| `pp` | a matrix of dimension nPts x 2 with the true points locations in 2 dimensions. |
| `xBins` | numerical vector with the ordered breaks of the grid along the x axis |
| `yBins` | numerical vector with the ordered breaks of the grid along the y axis |
| `whichAbove` | boolean vector of dimension nPts, selects the points above |
| `plt` | if not `TRUE` plots the grid, the points and the counts for each cell. |

### Value

a list containing `above`, the counts of points in excursion, `full` the counts per cell of all points, `freq`, the relative frequence.

### Author(s)

Dario Azzimonti

## getProfileExtrema *Profile extrema with BFGS optimization*

### Description

Evaluate profile extrema for a set of matrices allPsi with full optimization.

### Usage

```
getProfileExtrema(f, fprime = NULL, d, allPsi, opts = NULL)
```

## Arguments

| | |
|---|---|
| f | the function to be evaluated |
| fprime | derivative of the function |
| d | dimension of the input domain |
| allPsi | a list containing the matrices Psi (dim $pxd$) for which to compute the profile extrema |
| opts | a list containing the options for this function and the subfunctions getProfileSup_optim, getProfileInf_optim. The options only for getProfileExtrema are |

- `limits`: an optional list containing `lower` and `upper`, two vectors with the limits of the input space. If NULL then `limits=list(upper=rep(1,d),lower=rep(0,d))`
- `discretization`: an optional integer representing the discretization size for the profile computation for each dimension of eta. Pay attention that this leads to a grid of size `discretization^p`.
- `heavyReturn`: If TRUE returns also all minimizers, default is FALSE.
- `plts`: If TRUE and p==1 for all Psi in allPsi, plots the profile functions at each Psi, default is FALSE.
- `verb`: If TRUE, outputs intermediate results, default is FALSE.

## Value

a list of two data frames (min, max) of the evaluations of $P^sup_Psif(eta) = sup_{Psix=\eta}f(x)$ and $P^inf_Psif(eta) = inf_{Psix=\eta}f(x)$ discretized over 50 equally spaced points for each dimension for each Psi in `allPsi`. This number can be changed by defining it in options$discretization.

## Author(s)

Dario Azzimonti

## Examples

```
# Compute the oblique profile extrema with full optimization on 2d example

# Define the function
testF <- function(x,params,v1=c(1,0),v2=c(0,1)){
return(sin(crossprod(v1,x)*params[1]+params[2])+cos(crossprod(v2,x)*params[3]+params[4])-1.5)
}

testFprime <- function(x,params,v1=c(1,0),v2=c(0,1)){
  return(matrix(c(params[1]*v1[1]*cos(crossprod(v1,x)*params[1]+params[2])-
                  params[3]*v2[1]*sin(crossprod(v2,x)*params[3]+params[4]),
                 params[1]*v1[2]*cos(crossprod(v1,x)*params[1]+params[2])-
                  params[3]*v2[2]*sin(crossprod(v2,x)*params[3]+params[4])),ncol=1))
}


# Define the main directions of the function
theta=pi/6
pparams<-c(1,0,10,0)
```

```
vv1<-c(cos(theta),sin(theta))
vv2<-c(cos(theta+pi/2),sin(theta+pi/2))

# Define optimizer friendly function
f <-function(x){
return(testF(x,pparams,vv1,vv2))
}
fprime <- function(x){
 return(testFprime(x,pparams,vv1,vv2))
}

# Define list of directions where to evaluate the profile extrema
all_Psi <- list(Psi1=vv1,Psi2=vv2)



# Evaluate profile extrema along directions of all_Psi
allOblique<-getProfileExtrema(f=f,fprime = fprime,d = 2,allPsi = all_Psi,
                              opts = list(plts=FALSE,discretization=100,multistart=8))


# Consider threshold=0
threshold <- 0

# Plot oblique profile extrema functions
plotMaxMin(allOblique,allOblique$Design,threshold = threshold)

## Since the example is two dimensional we can visualize the regions excluded by the profile extrema
# evaluate the function at a grid for plots
inDes<-seq(0,1,,100)
inputs<-expand.grid(inDes,inDes)
outs<-apply(X = inputs,MARGIN = 1,function(x){return(testF(x,pparams,v1=vv1,v2=vv2))})

# obtain the points where the profiles take the threshold value
cccObl<-getChangePoints(threshold = threshold,allRes = allOblique,Design = allOblique$Design)

# visualize the functions and the regions excluded

image(inDes,inDes,matrix(outs,ncol=100),col=grey.colors(20),main="Example and oblique profiles")
contour(inDes,inDes,matrix(outs,ncol=100),add=TRUE,nlevels = 20)
contour(inDes,inDes,matrix(outs,ncol=100),add=TRUE,levels = c(threshold),col=4,lwd=1.5)
plotOblique(cccObl$alwaysEx$`0`[[1]],all_Psi[[1]],col=3)
plotOblique(cccObl$alwaysEx$`0`[[2]],all_Psi[[2]],col=3)
plotOblique(cccObl$neverEx$`0`[[1]],all_Psi[[1]],col=2)
plotOblique(cccObl$neverEx$`0`[[2]],all_Psi[[2]],col=2)
```

## getProfileInf_optim *Generic profile inf function computation with optim*

## Description

Compute profile inf function for an arbitrary matrix `Psi` with with the L-BFGS-B algorithm of optim. Here the linear equality constraint is eliminated by using the Null space of `Psi`.

## Usage

```
getProfileInf_optim(eta, Psi, f, fprime, d, options = NULL)
```

## Arguments

| | |
|---|---|
| `eta` | $p$ dimensional point where the function is to be evaluated |
| `Psi` | projection matrix of dimension `pxd` |
| `f` | function to be optimized (takes a vector `y` of dimension `d` and returns a real number) |
| `fprime` | derivative of f (same format, returning a $d$ dimensional vector) |
| `d` | dimension of the input for f |
| `options` | a list containing the options to be passed to optim:<br>• `par:` contains the starting point (a point in dimension d)<br>• `lower:` the lower bounds for the optimization domain (see optim for details)<br>• `upper:` the upper bounds for the optimization domain (see optim for details) |

## Value

a real value corresponding to $min_{x \in D_Psi} f(x)$

## Author(s)

Dario Azzimonti

## See Also

getProfileSup_optim, plotMaxMin

---

`getProfileSup_optim` *Generic profile sup function computation with optim*

---

## Description

Compute profile sup function for an arbitrary matrix `Psi` with the L-BFGS-B algorithm of optim.

## Usage

```
getProfileSup_optim(eta, Psi, f, fprime, d, options = NULL)
```

### Arguments

| | |
|---|---|
| `eta` | $p$ dimensional point where the function is to be evaluated |
| `Psi` | projection matrix of dimensions `p x d` |
| `f` | function to be optimized (takes a vector `y` of dimension `d` and returns a real number) |
| `fprime` | derivative of f (same format, returning a $d$ dimensional vector) |
| `d` | dimension of the input for f |
| `options` | a list containing the options to be passed to optim:<br>• `par:` contains the starting point (a point in dimension d-1)<br>• `lower:` the lower bounds for the optimization domain (see optim for details)<br>• `upper:` the upper bounds for the optimization domain (see optim for details) |

### Value

a real value corresponding to $max_{x \in D_{Psi}} f(x)$

### Author(s)

Dario Azzimonti

### See Also

getProfileInf_optim, plotMaxMin

---

`getSegments` *getSegments*

---

### Description

Auxiliary function for `getChangePoints`

### Usage

```
getSegments(y)
```

### Arguments

| | |
|---|---|
| `y` | a vector |

### Value

plots the sup and inf of the function for each dimension. If threshold is not NULL

### Author(s)

Dario Azzimonti

gradKm_dnewdata *Gradient of posterior mean and variance*

## Description

Computes the gradient of the posterior mean and variance of the kriging model in `object` at the points `newdata`.

## Usage

```
gradKm_dnewdata(object, newdata, type, se.compute = TRUE,
  light.return = FALSE, bias.correct = FALSE)
```

## Arguments

| | |
|---|---|
| `object` | a km object |
| `newdata` | a vector, matrix or data frame containing the points where to perform predictions. |
| `type` | a character corresponding to the type of kriging family (`"SK"` or `"UK"`). |
| `se.compute` | an optional boolean indicating whether to compute the posterior variance or not. Default is TRUE. |
| `light.return` | an optional boolean indicating whether to return additional variables. Default is FALSE. |
| `bias.correct` | an optional boolean to correct bias in the UK variance. Default is FALSE. |

## Value

Returns a list containing

- `mean`: the gradient of the posterior mean at `newdata`.
- `trend`: the gradient of the trend at `newdata`.
- `s2`: the gradient of the posterior variance at `newdata`.

## Author(s)

Dario Azzimonti

## grad_mean_Delta_T *Gradient of the mean function of difference process*

### Description

The function grad_mean_Delta_T computes the gradient for the mean function of the difference process $Z_x - \widetilde{Z}_x$ at x.

### Usage

```
grad_mean_Delta_T(x, kmModel, simupoints, T.mat, F.mat)
```

### Arguments

| | |
|---|---|
| x | a matrix $rxd$ containing the $r$ points where the function is to be computed. |
| kmModel | the km model of the Gaussian process $Z$. |
| simupoints | the matrix $lxd$ containing the pilot points $G$. |
| T.mat | the upper triangular factor of the Choleski decomposition of the covariance matrix of rbind(kmModel@X,simupoints) |
| F.mat | the evaluation of the trend function at rbind(kmModel@X,simupoints), see model.matrix. |

### Value

the value of the gradient for the mean function at x for the difference process $Z^{\Delta} = Z_x - \widetilde{Z}_x$.

### Author(s)

Dario Azzimonti

## grad_var_Delta_T *Gradient of the variance function of difference process*

### Description

The function grad_var_Delta_T computes the gradient for the variance function of the difference process $Z_x - \widetilde{Z}_x$ at x.

### Usage

```
grad_var_Delta_T(x, kmModel, simupoints, T.mat, F.mat)
```

### Arguments

| | |
|---|---|
| x | a matrix $rxd$ containing the $r$ points where the function is to be computed. |
| kmModel | the km model of the Gaussian process $Z$. |
| simupoints | the matrix $lxd$ containing the pilot points $G$. |
| T.mat | the upper triangular factor of the Choleski decomposition of the covariance matrix of rbind(kmModel@X,simupoints) |
| F.mat | the evaluation of the trend function at rbind(kmModel@X,simupoints), see model.matrix. |

### Value

the value of the gradient for the variance function at x for the difference process $Z^{\Delta} = Z_x - \widetilde{Z}_x$.

### Author(s)

Dario Azzimonti

---

## kGradSmooth *First order approximation*

---

### Description

Compute first order approximation of function from evaluations and gradient

### Usage

```
kGradSmooth(newPoints, profPoints, profEvals, profGradient)
```

### Arguments

| | |
|---|---|
| newPoints | vector of points where to approximate the function |
| profPoints | locations where the function was evaluated |
| profEvals | value of the evaluation at profPoints |
| profGradient | value of the gradient at profPoints |

### Value

approximated values of the function at newPoints

### Author(s)

Dario Azzimonti

## mean_Delta_T *mean function of difference process*

### Description

The function mean_Delta_T computes the mean function of the difference process $Z_x - \widetilde{Z}_x$ at x.

### Usage

```
mean_Delta_T(x, kmModel, simupoints, T.mat, F.mat)
```

### Arguments

| | |
|---|---|
| x | a matrix $rxd$ containing the $r$ points where the function is to be computed. |
| kmModel | the km model of the Gaussian process $Z$. |
| simupoints | the matrix $lxd$ containing the pilot points $G$. |
| T.mat | the upper triangular factor of the Choleski decomposition of the covariance matrix of rbind(kmModel@X,simupoints) |
| F.mat | the evaluation of the trend function at rbind(kmModel@X,simupoints), see model.matrix. |

### Value

the value of the mean function at x for the difference process $Z^{\Delta} = Z_x - \widetilde{Z}_x$.

### Author(s)

Dario Azzimonti

## obliqueProfiles *Oblique coordinate profiles starting from a kriging model*

### Description

The function obliqueProfiles computes the (oblique) profile extrema functions for the posterior mean of a Gaussian process and its confidence bounds

### Usage

```
obliqueProfiles(object, allPsi, threshold, options_full = NULL,
  options_approx = NULL, uq_computations = FALSE, plot_level = 0,
  plot_options = NULL, CI_const = NULL, return_level = 1, ...)
```

## Arguments

| | |
|---|---|
| `object` | either a km model or a list containing partial results. If `object` is a km model then all computations are carried out. If `object` is a list, then the function carries out all computations to complete the list results. |
| `allPsi` | a list containing the matrices Psi (dim $pxd$) for which to compute the profile extrema |
| `threshold` | the threshold of interest |
| `options_full` | an optional list of options for getProfileExtrema, see getProfileExtrema for details. |
| `options_approx` | an optional list of options for approxProfileExtrema, see approxProfileExtrema for details. |
| `uq_computations` | boolean, if TRUE the uq computations for the profile mean are computed. |
| `plot_level` | an integer to select the plots to return (0=no plots, 1=basic plots, 2= all plots) |
| `plot_options` | an optional list of parameters for plots. See setPlotOptions for currently available options. |
| `CI_const` | an optional vector containing the constants for the CI. If not NULL, then profiles extrema for $m_n(x) \pm CI_const[i] * s_n(x, x)$ are computed. |
| `return_level` | an integer to select the amount of details returned |
| `...` | additional parameters to be passed to obliqueProf_UQ. |

## Value

If return_level=1 a list containing

- `profMean_full`: the results of `getProfileExtrema` for the posterior mean
- `profMean_approx`: the results of `approxProfileExtrema` for the posterior mean
- `res_UQ`: the results of `obliqueProf_UQ` for the posterior mean

if return_level=2 the same list as above but also including

- `abs_err`: the vector of maximum absolute approximation errors for the profile inf /sup on posterior mean for the chosen approximation
- `times`: a list containing
  - `full`: computational time for the full computation of profile extrema
  - `approx`: computational time for the approximate computation of profile extrema

## Author(s)

Dario Azzimonti

## Examples

```
if (!requireNamespace("DiceKriging", quietly = TRUE)) {
stop("DiceKriging needed for this example to work. Please install it.",
     call. = FALSE)
}
# Compute a kriging model from 50 evaluations of the Branin function
# Define the function
g=function(x){
  return(-branin(x))
}
gp_des<-lhs::maximinLHS(20,2)
reals<-apply(gp_des,1,g)
kmModel<-km(design = gp_des,response = reals,covtype = "matern3_2")

threshold=-10

# Compute oblique profiles on the posterior mean
# (for theta=0 it is equal to coordinateProfiles)
options_full<-list(multistart=4,heavyReturn=TRUE,discretization=100)
options_approx<- list(multistart=4,heavyReturn=TRUE,initDesign=NULL,fullDesignSize=100)
theta=pi/4
allPsi = list(Psi1=matrix(c(cos(theta),sin(theta)),ncol=2),
Psi2=matrix(c(cos(theta+pi/2),sin(theta+pi/2)),ncol=2))

## Not run:
profMeans<-obliqueProfiles(object = kmModel,allPsi = allPsi,threshold = threshold,
                           options_full = options_full,options_approx = options_approx,
                           uq_computations = FALSE,plot_level = 3,plot_options = NULL,
                           CI_const = NULL,return_level = 2)


# Approximate oblique profiles with UQ
plot_options<-list(save=FALSE, titleProf = "Coordinate profiles",
                   title2d = "Posterior mean",qq_fill=TRUE)
options_sims<-list(nsim=150)
obProfUQ<-obliqueProfiles(object=profMeans,threshold=threshold,allPsi = allPsi,
                           options_full=options_full, options_approx=options_approx,
                           uq_computations=TRUE, plot_level=3,plot_options=NULL,
                           CI_const=NULL,return_level=2,options_sims=options_sims)

## End(Not run)
```

---

`obliqueProf_UQ` *Oblique profiles UQ from a kriging model*

---

## Description

The function obliqueProf_UQ computes the profile extrema functions for posterior realizations of a Gaussian process and its confidence bounds

## Usage

```
obliqueProf_UQ(object, allPsi, threshold, allResMean = NULL,
  quantiles_uq = c(0.05, 0.95), options_approx = NULL,
  options_full_sims = NULL, options_sims = NULL,
  options_bound = NULL, plot_level = 0, plot_options = NULL,
  return_level = 1)
```

## Arguments

| | |
|---|---|
| `object` | either a km model or a list containing partial results. If `object` is a km model then all computations are carried out. If `object` is a list, then the function carries out all computations to complete the results list. |
| `allPsi` | a list containing the matrices Psi (dim $pxd$) for which to compute the profile extrema |
| `threshold` | the threshold of interest |
| `allResMean` | a list resulting from `getProfileExtrema` or `approxProfileExtrema` for the profile extrema on the mean. If NULL the median from the observations is plotted |
| `quantiles_uq` | a vector containing the quantiles to be computed |
| `options_approx` | an optional list of options for approxProfileExtrema, see approxProfileExtrema for details. |
| `options_full_sims` | an optional list of options for getProfileExtrema, see getProfileExtrema for details. If NULL the full computations are not executed. NOTE: this computations might be very expensive! |
| `options_sims` | an optional list of options for the posterior simulations.<br>• `algorithm`: string choice of the algorithm to select the pilot points ("A" or "B", default "B");<br>• `lower`: $d$ dimensional vector with lower bounds for pilot points, default `rep(0,d)`;<br>• `upper`: $d$ dimensional vector with upper bounds for pilot points, default `rep(1,d)`;<br>• `batchsize`: number of pilot points, default `120`;<br>• `optimcontrol`: list containing the options for optimization, see optim_dist_measure;<br>• `integcontrol`: list containing the options for numerical integration of the criterion, see optim_dist_measure;<br>• `integration.param`: list containing the integration design, obtained with the function integration_design;<br>• `nsim`: number of approximate GP simulations, default `300`. |
| `options_bound` | an optional list containing `beta` the confidence level for the approximation and `alpha` the confidence level for the bound. Note that `alpha` > `2*beta`. If `NULL`, the bound is not computed. |
| `plot_level` | an integer to select the plots to return (0=no plots, 1=basic plots, 2= all plots) |
| `plot_options` | an optional list of parameters for plots. See setPlotOptions for currently available options. |
| `return_level` | an integer to select the amount of details returned |

## Value

If return_level=1 a list containing

- profSups: an array dxfullDesignSizexnsims containing the profile sup for each coordinate for each realization.
- profInfs: an array dxfullDesignSizexnsims containing the profile inf for each coordinate for each realization.
- prof_quantiles_approx: a list containing the quantiles (levels set by quantiles_uq) of the profile extrema functions.

if return_level=2 the same list as above but also including more: a list containing

- times: a list containing
  - tSpts: computational time for selecting pilot points.
  - tApprox1ord: vector containing the computational time required for profile extrema computation for each realization
- simuls: a matrix containing the value of the field simulated at the pilot points
- sPts: the pilot points

## Author(s)

Dario Azzimonti

## Examples

```
if (!requireNamespace("DiceKriging", quietly = TRUE)) {
stop("DiceKriging needed for this example to work. Please install it.",
     call. = FALSE)
}
# Compute a kriging model from 50 evaluations of the Branin function
# Define the function
g<-function(x){
  return(-branin(x))
}
gp_des<-lhs::maximinLHS(20,2)
reals<-apply(gp_des,1,g)
kmModel<-km(design = gp_des,response = reals,covtype = "matern3_2")

threshold=-10
d<-2

# Compute oblique profiles UQ starting from GP model
# define simulation options
options_sims<-list(algorithm="B", lower=rep(0,d), upper=rep(1,d),
             batchsize=80, optimcontrol = list(method="genoud",pop.size=100,print.level=0),
                   integcontrol = list(distrib="sobol",n.points=1000), nsim=150)
# define approximation options
options_approx<- list(multistart=4,heavyReturn=TRUE,
                      initDesign=NULL,fullDesignSize=100,
                      smoother=NULL)
```

```
# define plot options
options_plots<-list(save=FALSE, titleProf = "Coordinate profiles",
                    title2d = "Posterior mean",qq_fill=TRUE)

# Define the oblique directions
# (for theta=0 it is equal to coordinateProfiles)
theta=pi/4
allPsi = list(Psi1=matrix(c(cos(theta),sin(theta)),ncol=2),
              Psi2=matrix(c(cos(theta+pi/2),sin(theta+pi/2)),ncol=2))
## Not run:
# here we reduce the number of simulations to speed up the example
# a higher number should be used
options_sims$nsim <- 50

# profile UQ on approximate oblique profiles
oProfiles_UQ<-obliqueProf_UQ(object = kmModel,threshold = threshold,allPsi=allPsi,
                             allResMean = NULL,quantiles_uq = c(0.05,0.95),
                             options_approx = options_approx, options_full_sims = NULL,
                             options_sims = options_sims,options_bound = NULL,
                           plot_level = 3, plot_options = options_plots,return_level = 3)
# profile UQ on full optim oblique profiles

options_full_sims<-list(multistart=4,heavyReturn=TRUE)
oProfiles_UQ_full<- obliqueProf_UQ(object = oProfiles_UQ,threshold = threshold,allPsi=allPsi,
                             allResMean = NULL,quantiles_uq = c(0.05,0.95),
                   options_approx = options_approx, options_full_sims = options_full_sims,
                             options_sims = options_sims,options_bound = NULL,
                           plot_level = 3, plot_options = options_plots,return_level = 3)



# profile UQ on full optim oblique profiles with bound
oProfiles_UQ_full_bound<-obliqueProf_UQ(object = oProfiles_UQ_full,threshold = threshold,
                                        allPsi=allPsi, allResMean = NULL,
                                        quantiles_uq = c(0.05,0.95),
                                        options_approx = options_approx,
                                        options_full_sims = options_full_sims,
                                      options_sims = options_sims,
                                      options_bound = list(beta=0.024,alpha=0.05),
                                      plot_level = 3, plot_options = options_plots,
                                      return_level = 3)

## End(Not run)
```

---

`plotBivariateProfiles` *Plot bivariate profiles*

---

## Description

Plot bivariate profiles, for dimension up to 6.

### Usage

```
plotBivariateProfiles(bivProf, allPsi, Design = NULL, threshold = NULL,
  whichIQR = NULL, plot_options = NULL, ...)
```

### Arguments

| | |
|---|---|
| bivProf | list returned by obliqueProfiles. |
| allPsi | a list containing the matrices Psi (dim $2xd$) for which to compute the profile extrema |
| Design | a matrix of dimension $(2d)xnumPsi$ encoding the first (Design[1:d,]) and the second ((Design[(d+1):(2*d),])) axis values. |
| threshold | if not NULL plots the level as a contour. |
| whichIQR | which quantiles to use for the inter-quantile range plot. If NULL, automatically selects the first and the last element of bivProfres_UQ$prof_quantiles_approx |
| plot_options | list as returned by setPlotOptions. |
| ... | additional parameters to be passed to the plot function |

### Value

plots the 2d maps of the profile sup and inf of the function for each Psi in allPsi. If threshold is not NULL also contours the threshold level.

### Author(s)

Dario Azzimonti

---

## plotMaxMin *Plot coordinate profiles*

---

### Description

Plot coordinate profiles, for dimension up to 6.

### Usage

```
plotMaxMin(allRes, Design = NULL, threshold = NULL, changes = FALSE,
  trueEvals = NULL, ...)
```

### Arguments

| | |
|---|---|
| allRes | list containing the list res which contains the computed minima and maxima. The object returned by the function getAllMaxMin. |
| Design | a d dimensional design corresponding to the points |
| threshold | if not NULL plots the level |

| | |
|---|---|
| changes | boolean, if not FALSE plots the points where profile extrema take values near the threshold. |
| trueEvals | if not NULL adds to each plot the data points and the observed value |
| ... | additional parameters to be passed to the plot function |

## Value

plots the sup and inf of the function for each dimension. If threshold is not NULL

## Author(s)

Dario Azzimonti

---

plotOblique *plotOblique*

---

## Description

Auxiliary function for 2d plotting of excluded regions

## Usage

```
plotOblique(changePoints, direction, ...)
```

## Arguments

| | |
|---|---|
| changePoints | Numerical vector with the change points (usually if cp=getChangePoints(...), then this is cc$alwaysEx[[1]][[1]] for example) |
| direction | The Psi vector used for the direction |
| ... | parameters to be passed to abline |

## Value

adds to the current plot the lines $x$ s.t. direction^T $x$ = changePoints[i] for all i

## Author(s)

Dario Azzimonti

`plotOneBivProfile` *Plot bivariate profiles*

## Description

Plots the bivariate profiles stored in `allRes` for each Psi in `allPsi`.

## Usage

```
plotOneBivProfile(allRes, allPsi, Design = NULL, threshold = NULL,
  trueEvals = NULL, main_addendum = "", ...)
```

## Arguments

| | |
|---|---|
| `allRes` | list containing the list `res` which contains the computed minima and maxima. The object returned by the function `getProfileExtrema`. |
| `allPsi` | a list containing the matrices Psi (dim $2xd$) for which to compute the profile extrema |
| `Design` | a matrix of dimension $(2d)xnumPsi$ encoding the first (`Design[1:d,]`) and the second ((`Design[(d+1):(2*d),]`)) axis values. |
| `threshold` | if not `NULL` plots the level as a contour. |
| `trueEvals` | if not `NULL` adds to each plot the data points and the observed value |
| `main_addendum` | additional string to add to image title. Default is empty string. |
| `...` | additional parameters to be passed to the plot function |

## Value

plots the 2d maps of the profile sup and inf in `allRes` for each Psi in `allPsi`. If threshold is not NULL also contours the threshold level.

## Author(s)

Dario Azzimonti

## See Also

plotBivariateProfiles

plot_univariate_profiles_UQ *Univariate profile extrema with UQ*

## Description

Function to plot the univariate profile extrema functions with UQ

## Usage

```
plot_univariate_profiles_UQ(objectUQ, plot_options, nsims, threshold,
  nameFile = "prof_UQ", quantiles_uq = c(0.05, 0.95),
  profMean = NULL, typeProf = "approx")
```

## Arguments

| | |
|---|---|
| objectUQ | an object returned by coordProf_UQ or the object saved in obj$res_UQ, if obj is the object returned by coordinateProfiles. |
| plot_options | a list containing the same elements as the one passed to coordinateProfiles |
| nsims | number of simulations |
| threshold | threshold of interest |
| nameFile | the central name of the plot file |
| quantiles_uq | a vector containing the quantiles to be computed |
| profMean | the profile coordinate extrema functions for the mean. It is saved in obj$profMean_full or obj$profMean_approx if obj is an object returned by coordinateProfiles. |
| typeProf | a string to choose with type of profile extrema for simulations to plot<br>• "approx"plots only the approximate profile extrema for simulations<br>• "full"plots only the full profile extrema for simulations<br>• "both"plots both the approximate and full profile extrema for simulations |

## Value

Plots either to the default graphical device or to pdf (according to the options passed in plot_options)

profExtrema *profExtrema package*

## Description

Computation and plots of profile extrema functions. The package main functions are:

**Computation:**
- `coordinateProfiles`: Given a `km` objects computes the coordinate profile extrema function for the posterior mean and its quantiles.
- `coordProf_UQ`: UQ part of `coordinateProfiles`.
- `obliqueProfiles`: Given a `km` objects computes the profile extrema functions for a generic list of matrices Psi for the posterior mean and its quantiles.
- `obliqueProf_UQ`: The UQ part of `obliqueProfiles`.
- `getAllMaxMin`: computes coordinate profile extrema with full optimization for a deterministic function.
- `approxMaxMin`: approximates coordinate profile extrema for a deterministic function.
- `getProfileExtrema`: computes profile extrema given a list of matrices Psi for a deterministic function.
- `approxProfileExtrema`: approximates profile extrema given a list of matrices Psi for a deterministic function.

**Plotting:**
- `plot_univariate_profiles_UQ`: plots for the results of `coordProf_UQ` or `obliqueProf_UQ`. Note that this function only works for univariate profiles.
- `plotBivariateProfiles`: plots the bivariate maps results of a call to `obliqueProfiles` with a two dimensional projection matrix Psi.
- `plotMaxMin`: simple plotting function for univariate profile extrema.
- `plotOneBivProfile`: simple plotting function for bivariate profile extrema.

## Details

Package: profExtrema
Type: Package
Version: 0.2.1
Date: 2020-03-20

## Note

This work was supported in part the Hasler Foundation, grant number 16065 and by the Swiss National Science Foundation, grant number 167199. The author warmly thanks David Ginsbourger, Jérémy Rohmer and Déborah Idier for fruitful discussions and accurate, thought provoking suggestions.

## Author(s)

Dario Azzimonti (dario.azzimonti@gmail.com) .

## References

Azzimonti, D., Bect, J., Chevalier, C., and Ginsbourger, D. (2016). Quantifying uncertainties on excursion sets under a Gaussian random field prior. SIAM/ASA Journal on Uncertainty Quantification, 4(1):850–874.

Azzimonti, D., Ginsbourger, D., Rohmer, J. and Idier, D. (2017+). Profile extrema for visualizing and quantifying uncertainties on excursion regions. Application to coastal flooding. arXiv:1710.00688.

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

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

Johnson, S. G. The NLopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt

Koenker, R. (2017). quantreg: Quantile Regression. R package version 5.33.

Nocedal, J. and Wright, S. J. (2006). Numerical Optimization, second edition. Springer- Verlag, New York.

Neuwirth, E. (2014). RColorBrewer: ColorBrewer Palettes. R package version 1.1-2.

Roustant, O., Ginsbourger, D., Deville, Y. (2012). DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization. Journal of Statistical Software, 51(1): 1-55.

---

`prof_mean_var_Delta` *Profile extrema for the mean and variance functions of difference process*

---

## Description

The function prof_mean_var_Delta computes the profile extrema functions for the mean and variance functions of the difference process $Z_x - \widetilde{Z}_x$ at `x`.

## Usage

```
prof_mean_var_Delta(kmModel, simupoints, allPsi = NULL,
  options_full_sims = NULL, options_approx = NULL, F.mat = NULL,
  T.mat = NULL)
```

## Arguments

| | |
|---|---|
| `kmModel` | the km model of the Gaussian process $Z$. |
| `simupoints` | the matrix $lxd$ containing the pilot points $G$. |
| `allPsi` | optional list of matrices (dim $pxd$) for which to compute the profile extrema. If NULL coordinate profiles are computed. |

options_full_sims
: an optional list of options for getAllMaxMin(or approxProfileExtrema if allPsi not NULL). If NULL the full computations are not excuted. NOTE: this computations might be very expensive!

options_approx
: an optional list of options for approxMaxMin (or approxProfileExtrema if allPsi not NULL).

F.mat
: the evaluation of the trend function at rbind(kmModel@X,simupoints), see model.matrix, if NULL it is computed.

T.mat
: the upper triangular factor of the Choleski decomposition of the covariance matrix of rbind(kmModel@X,simupoints), if NULL it is computed.

## Value

the profile extrema functions at options_approx$design for the mean and variance function of the difference process $Z^{\Delta} = Z_x - \widetilde{Z}_x$.

## Author(s)

Dario Azzimonti

---

# setPlotOptions *Set-up the plot options when NULL*

---

## Description

Function to set-up plot options for plot_univariate_profiles_UQ, plotBivariateProfiles, coordinateProfiles, coordProf_UQ, obliqueProfiles and obliqueProf_UQ.

## Usage

```
setPlotOptions(plot_options = NULL, d, num_T, kmModel = NULL)
```

## Arguments

plot_options
: the list of plot options to set-up

d
: number of coordinates

num_T
: number of thresholds of interest

kmModel
: a km model, used to obtain the coordinates names.

## Value

the properly set-up list containing the following fields

- save:boolean, if TRUE saves the plots in folderPlots
- folderPlots:a string containing the destination folder for plots, if save==TRUE default is ./

- ylim: a matrix coordx2 containing the ylim for each coordinate, if NULL in plot_options this is left NULL and automatically set at the plot time.
- titleProf: a string containing the title for the coordinate profile plots, default is "Coordinate profiles"
- title2d: a string containing the title for the 2d plots (if the input is 2d), default is "Posterior mean"
- design: a $dxr$ matrix where $d$ is the input dimension and $r$ is the size of the discretization for plots at each dimension
- coord_names: a $d$-vector of characters naming the dimensions. If NULL and kmModel not NULL then it is the names of kmModel@X otherwise x_1,...,x_d
- id_save: a string to be added to the plot file names, useful for serial computations on HPC, left as in plot_options.
- qq_fill: if TRUE it fills the region between the first 2 quantiles in quantiles_uq and between the upper and lower bound in objectUQ$bound$bound, if NULL, it is set as FALSE.
- bound_cols: a vector of two strings containing the names of the colors for upper and lower bound plots.
- qq_fill_colors: a list containing the colors for qq_fill: approx for 2 quantiles, bound_min for bounds on the profile inf, bound_max for profile sup. Initialized only if qq_fill==TRUE.
- col_CCPthresh_nev: Color palette of dimension num_T for the colors of the vertical lines delimiting the intersections between the profiles sup and the thresholds
- col_CCPthresh_alw: Color palette of dimension num_T for the colors of the vertical lines delimiting the intersections between the profiles inf and the thresholds
- col_thresh: Color palette of dimension num_T for the colors of the thresholds
- fun_evals: integer denoting the level of plot for the true evaluations.
  - 0: default, no plots for true evaluations;
  - 1: plot the true evaluations as points in 2d plots, no true evaluation plots in 1d;
  - 2: plot true evaluations, in 2d with different color for values above threshold;
  - 3: plot true evaluations, in 2d plots in color, with background of the image colored as proportion of points inside excursion;

if all the fields are already filled then returns plot_options

## Author(s)

Dario Azzimonti

`var_Delta_T` *Variance function of difference process*

## Description

The function var_Delta_T computes the gradient for the variance function of the difference process $Z_x - \widetilde{Z}_x$ at x.

## Usage

```
var_Delta_T(x, kmModel, simupoints, T.mat, F.mat)
```

## Arguments

| | |
|---|---|
| x | a matrix $rxd$ containing the $r$ points where the function is to be computed. |
| kmModel | the km model of the Gaussian process $Z$. |
| simupoints | the matrix $lxd$ containing the pilot points $G$. |
| T.mat | the upper triangular factor of the Choleski decomposition of the covariance matrix of `rbind(kmModel@X,simupoints)` |
| F.mat | the evaluation of the trend function at `rbind(kmModel@X,simupoints)`, see model.matrix. |

## Value

the value of the variance function at x for the difference process $Z^{\Delta} = Z_x - \widetilde{Z}_x$.

## Author(s)

Dario Azzimonti

# Index