Package ‘GPareto’

December 10, 2018

Type Package
Title Gaussian Processes for Pareto Front Estimation and Optimization
Version 1.1.2
Date 2018-12-07
Author Mickael Binois, Victor Picheny
Maintainer Mickael Binois <mbinois@mcs.anl.gov>
Description Gaussian process regression models, a.k.a. Kriging models, are applied to global multi-objective optimization of black-box functions. Multi-objective Expected Improvement and Step-wise Uncertainty Reduction sequential infill criteria are available. A quantification of uncertainty on Pareto fronts is provided using conditional simulations.
License GPL-3
Depends DiceKriging, emoa
Imports Rcpp (>= 0.12.15), methods, rgenoud, pbivnorm, pso, randtoolbox, KrigInv, MASS, DiceDesign, ks, rgl
Suggests knitr
VignetteBuilder knitr
LinkingTo Rcpp
Repository CRAN
URL http://github.com/mbinois/GPareto
BugReports http://github.com/mbinois/GPareto/issues
RoxygenNote 6.1.1
NeedsCompilation yes
Date/Publication 2018-12-10 10:10:03 UTC

R topics documented:

GPareto-package ........................................ 2
checkPredict ........................................... 6
Description

Multi-objective optimization and quantification of uncertainty on Pareto fronts, using Gaussian process models.

Details

- Important functions:
  - GParetoopt
  - easyGParetopt
  - crit_optimizer
  - plotGPareto
  - CPF

Note

Part of this work has been conducted within the frame of the ReDice Consortium, gathering industrial (CEA, EDF, IFPEN, IRSN, Renault) and academic (Ecole des Mines de Saint-Etienne, INRIA, and the University of Bern) partners around advanced methods for Computer Experiments. (http://www.redice-project.org/).
The authors would like to thank Yves Deville for his precious advices in R programming and packaging, as well as Olivier Roustant and David Ginsbourger for testing and suggestions of improvements for this package. We would also like to thank Tobias Wagner for providing his Matlab codes for the SMS-EGO strategy.

Author(s)
Mickael Binois, Victor Picheny

References


See Also
*DiceKriging, DiceOptim*

Examples
```r
## Not run:
#---------------------------------------------------------------
# Example 1 : Surrogate-based multi-objective Optimization with postprocessing
#---------------------------------------------------------------
set.seed(25468)

d <- 2
fname <- P2
```
plotParetoGrid(P2) # For comparison

# Optimization
budget <- 25
lower <- rep(0, d)
upper <- rep(1, d)

omEGO <- easyGParetooptim(fn = fname, budget = budget, lower = lower, upper = upper)

# Postprocessing
plotGPareto(omEGO, add= FALSE, UQ_PF = TRUE, UQ_PS = TRUE, UQ_dens = TRUE)

## End(Not run)
#---------------------------------------------------------------
# Example 2 : Surrogate-based multi-objective Optimization including a cheap function
#---------------------------------------------------------------
set.seed(42)
library(DiceDesign)
d <- 2
fname <- P1
n.grid <- 19
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
nappr <- 15
design.grid <- maximinESE_LHS(lhsDesign(nappr, d, seed = 42)$design)$design
response.grid <- t(apply(design.grid, 1, fname))

mf1 <- km(~., design = design.grid, response = response.grid[,1])
mf2 <- km(~., design = design.grid, response = response.grid[,2])
model <- list(mf1, mf2)
nsteps <- 1
lower <- rep(0, d)
upper <- rep(1, d)

# Optimization with fastfun: hypervolume with discrete search
optimcontrol <- list(method = "discrete", candidate.points = test.grid)
omEGO2 <- GParetooptim(model = model, fn = fname, cheapfn = branin, crit = "SMS",
nsteps = nsteps, lower = lower, upper = upper, optimcontrol = optimcontrol)
print(omEGO2$par)
print(omEGO2$values)

## Not run:
plotGPareto(omEGO2)

#---------------------------------------------------------------
# Example 3 : Surrogate-based multi-objective Optimization (4 objectives)
#---------------------------------------------------------------
```
set.seed(42)
library(DiceDesign)

d <- 5

fname <- DTLZ3
nappr <- 25
design.grid <- maximinESE_LHS(lhsDesign(nappr, d, seed = 42)$design)$design
response.grid <- t(apply(design.grid, 1, fname, nobj = 4))

# Optimization
nsteps <- 5
lower <- rep(0, d)
upper <- rep(1, d)

omEGO3 <- GParetoptim(model = list(mf1, mf2, mf3, mf4), fn = fname, crit = "EMI",
                      nsteps = nsteps, lower = lower, upper = upper, nobj = 4)

print(omEGO3$par)
print(omEGO3$values)
plotGPareto(omEGO3)

# Example 4 : quantification of uncertainty on Pareto front
library(DiceDesign)
set.seed(42)

nvar <- 2

# Test function P1
fname <- "P1"

# Initial design
nappr <- 10
design.grid <- maximinESE_LHS(lhsDesign(nappr, nvar, seed = 42)$design)$design
response.grid <- t(apply(design.grid, 1, fname))

PF <- t(nondominated_points(t(response.grid)))

# Kriging models : matern5_2 covariance structure, linear trend, no nugget effect
mf1 <- km(~., design = design.grid, response = response.grid[,1])
mf2 <- km(~., design = design.grid, response = response.grid[,2])
mf3 <- km(~., design = design.grid, response = response.grid[,3])
mf4 <- km(~., design = design.grid, response = response.grid[,4])

# Conditional simulations generation with random sampling points
nsim <- 100 # increase for better results
npointssim <- 1000 # increase for better results
simu_f1 <- matrix(0, nrow = nsim, ncol = npointssim)
simu_f2 <- matrix(0, nrow = nsim, ncol = npointssim)
design.sim <- array(0, dim = c(npointssim, nvar, nsim))
```

---

The code demonstrates the use of the GPareto package for optimization and quantification of uncertainty on Pareto fronts. It sets a seed, loads the DiceDesign library, and sets up a design grid for a function with two variables. The optimization process is carried out using different kriging models, and conditional simulations are generated with random sampling points for further analysis.

---

**GPareto-package**
for(i in 1:nsim){
  design.sim[,,i] <- matrix(runif(nvar*npointssim), nrow = npointssim, ncol = nvar)
  Simu_f1[i,] <- simulate(mf1, nsim = 1, newdata = design.sim[,,i], cond = TRUE,
    checkNames = FALSE, nugget.sim = 10^-8)
  Simu_f2[i,] <- simulate(mf2, nsim = 1, newdata = design.sim[,,i], cond = TRUE,
    checkNames = FALSE, nugget.sim = 10^-8)
}

# Computation of the attainment function and Vorob'ev Expectation
CPF1 <- CPF(Simu_f1, Simu_f2, response.grid, paretoFront = PF)

summary(CPF1)

plot(CPF1)

# Display of the symmetric deviation function
plotSymDevFun(CPF1)

## End(Not run)

---

**checkPredict**

*Prevention of numerical instability for a new observation*

**Description**

Check that the new point is not too close to already known observations to avoid numerical issues. Closeness can be estimated with several distances.

**Usage**

```r
checkPredict(x, model, threshold = 1e-04, distance = "euclidean",
             type = "UK")
```

**Arguments**

- `x`: a vector representing the input to check, alternatively a matrix with one point per row,
- `model`: list of objects of class `km`, one for each objective functions,
- `threshold`: optional value for the minimal distance to an existing observation, default to `1e-4`,
- `distance`: selection of the distance between new observations, between "euclidean" (default), "none", "covdist" and "covratio", see details,
- `type`: "SK" or "UK" (default), depending whether uncertainty related to trend estimation has to be taken into account.
Details

If the distance between \( x \) and the closest observations in model is below threshold, \( x \) should not be evaluated to avoid numerical instabilities. The distance can simply be the Euclidean distance or the canonical distance associated with the kriging predictive covariance \( k \):

\[
d(x, y) = \sqrt{k(x, x) - 2k(x, y) + k(y, y)}.
\]

The last solution is the ratio between the prediction variance at \( x \) and the variance of the process. None can be used, e.g., if points have been selected already.

Value

TRUE if the point should not be tested.

---

**CPF**

*Conditional Pareto Front simulations*

Description

Compute (on a regular grid) the empirical attainment function from conditional simulations of Gaussian processes corresponding to two objectives. This is used to estimate the Vorob’ev expectation of the attained set and the Vorob’ev deviation.

Usage

\[
\text{CPF}(\text{fun1sims}, \text{fun2sims}, \text{response}, \text{paretoFront} = \text{NULL}, \text{f1lim} = \text{NULL}, \\
\text{f2lim} = \text{NULL}, \text{refPoint} = \text{NULL}, n.\text{grid} = 100, \\
\text{compute.\text{VorobExp}} = \text{TRUE}, \text{compute.\text{VorobDev}} = \text{TRUE})
\]

Arguments

- **fun1sims**: numeric matrix containing the conditional simulations of the first output (one sample in each row),
- **fun2sims**: numeric matrix containing the conditional simulations of the second output (one sample in each row),
- **response**: a matrix containing the value of the two objective functions, one output per row,
- **paretoFront**: optional matrix corresponding to the Pareto front of the observations. It is estimated from response if not provided,
- **f1lim**: optional vector (see details),
- **f2lim**: optional vector (see details),
- **refPoint**: optional vector (see details),
- **n.grid**: integer determining the grid resolution,
- **compute.VorobExp**: optional boolean indicating whether the Vorob’ev Expectation should be computed. Default is TRUE,
- **compute.VorobDev**: optional boolean indicating whether the Vorob’ev deviation should be computed. Default is TRUE.
Details

Works with two objectives. The user can provide locations of grid lines for computation of the attainment function with vectors $f_{lim}$ and $f_{Rlim}$, in the form of regularly spaced points. It is possible to provide only refPoint as a reference for hypervolume computations. When missing, values are determined from the axis-wise extrema of the simulations.

Value

A list which is given the S3 class "CPF".

- **x**, **y**: locations of grid lines at which the values of the attainment are computed,
- **values**: numeric matrix containing the values of the attainment on the grid,
- **PF**: matrix corresponding to the Pareto front of the observations,
- **responses**: matrix containing the value of the two objective functions, one objective per column,
- **fun1sims**, **fun2sims**: conditional simulations of the first/second output,
- **VE**: Vorob’ev expectation, computed if compute.VorobExp = TRUE (default),
- **beta_star**: Vorob’ev threshold, computed if compute.VorobExp = TRUE (default),
- **VD**: Vorov’ev deviation, computed if compute.VorobDev = TRUE (default),

References


See Also

Methods coef, summary and plot can be used to get the coefficients from a CPF object, to obtain a summary or to display the attainment function (with the Vorob’ev expectation if compute.VorobExp is TRUE).

Examples

```r
library(DiceDesign)
set.seed(42)

nvar <- 2

fname <- "P1" # Test function

# Initial design
nappr <- 10
```
crit_EHI

```r
design.grid <- maximinESE_LHS(lhsDesign(nappr, nvar, seed = 42)$design)$design
response.grid <- t(apply(design.grid, 1, fname))

# kriging models: matern5_2 covariance structure, linear trend, no nugget effect
mf1 <- km(~., design = design.grid, response = response.grid[,1])
mf2 <- km(~., design = design.grid, response = response.grid[,2])

# Conditional simulations generation with random sampling points
nsim <- 40
npointssim <- 150 # increase for better results
Simu_f1 <- matrix(0, nrow = nsim, ncol = npointssim)
Simu_f2 <- matrix(0, nrow = nsim, ncol = npointssim)
design.sim <- array(0, dim = c(npointssim, nvar, nsim))
for(i in 1:nsim){
  design.sim[,i] <- matrix(runif(nvar*npointssim), nrow = npointssim, ncol = nvar)
  Simu_f1[i,] <- simulate(mf1, nsim = 1, newdata = design.sim[,i], cond = TRUE,
                          checkNames = FALSE, nugget.sim = 10^-8)
  Simu_f2[i,] <- simulate(mf2, nsim = 1, newdata = design.sim[,i], cond = TRUE,
                          checkNames = FALSE, nugget.sim = 10^-8)
}

# Attainment and Vorob'ev expectation + deviation estimation
CPF1 <- CPF(Simu_f1, Simu_f2, response.grid)

# Details about the Vorob'ev threshold and Vorob'ev deviation
summary(CPF1)

# Graphics
plot(CPF1)
```

crit_EHI | Expected Hypervolume Improvement with m objectives

**Description**

Multi-objective Expected Hypervolume Improvement with respect to the current Pareto front. With two objectives the analytical formula is used, while Sample Average Approximation (SAA) is used with more objectives. To avoid numerical instabilities, the new point is penalized if it is too close to an existing observation.

**Usage**

```r
crit_EHI(x, model, paretoFront = NULL, critcontrol = list(nb.samp = 50,
  seed = 42), type = "UK")
```
Arguments

- **x**: a vector representing the input for which one wishes to calculate EHI, alternatively a matrix with one point per row.
- **model**: list of objects of class `km`, one for each objective function.
- **paretoFront** *(optional)*: matrix corresponding to the Pareto front of size `[n.pareto x n.obj]`, or any reference set of observations.
- **critcontrol** *(optional)*: list with arguments:
  - `nb_samp`: number of random samples from the posterior distribution (with more than two objectives), default to 50, increasing gives more reliable results at the cost of longer computation time;
  - `seed`: seed used for the random samples (with more than two objectives);
  - `refPoint`: reference point for Hypervolume Expected Improvement;
  - `extendper`: if no reference point `refPoint` is provided, for each objective it is fixed to the maximum over the Pareto front plus `extendper` times the range, Default value to 0.2, corresponding to 1.1 for a scaled objective with a Pareto front in `[0,1]^n.obj`.

Options for the `checkPredict` function: `threshold(1e-4)` and `distance(covdist)` are used to avoid numerical issues occurring when adding points too close to the existing ones.

- **type**: "SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account.

Details

The computation of the analytical formula with two objectives is adapted from the Matlab source code by Michael Emmerich and Andre Deutz, LIACS, Leiden University, 2010 available here: [http://liacs.leidenuniv.nl/~csmoda/code/HV_based_expected_improvement.zip](http://liacs.leidenuniv.nl/~csmoda/code/HV_based_expected_improvement.zip).

Value

The Expected Hypervolume Improvement at `x`.

References


See Also

- `EI` from package DiceOptim, `crit_EMI`, `crit_SUR`, `crit_SMS`. 
crit_EMI

Examples

# Expected Hypervolume Improvement surface associated with the "P1" problem at a 15 points design
#--------------------------------------------------------
set.seed(25468)
library(DiceDesign)

n_var <- 2
f_name <- "P1"
n.grid <- 26
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
n_appr <- 15
design.grid <- round(maximinESELHS(lhsDesign(n_appr, n_var, seed = 42)$design)$design, 1)
response.grid <- t(apply(design.grid, 1, f_name))
Front_Pareto <- t(nondominated_points(t(response.grid)))

mf1 <- km(~., design = design.grid, response = response.grid[,1])
mf2 <- km(~., design = design.grid, response = response.grid[,2])

EHI_grid <- crit_EHI(x = as.matrix(test.grid), model = list(mf1, mf2),
critcontrol = list(refPoint = c(300,0)))

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
matrix(EHI_grid, n.grid), main = "Expected Hypervolume Improvement",
xlab = expression(x[1]), ylab = expression(x[2]), color = terrain.colors,
plot.axes = (axis(1); axis(2));
points(design.grid[,1], design.grid[,2], pch = 21, bg = "white")
)

Description

Expected Maximin Improvement with respect to the current Pareto front with Sample Average Approximation. The semi-analytical formula is used in the bi-objective scale if the Pareto front is in [-2,2]^2, for numerical stability reasons. To avoid numerical instabilities, the new point is penalized if it is too close to an existing observation.

Usage

crit_EMI(x, model, paretoFront = NULL, critcontrol = list(nb.samp = 50,
seed = 42), type = "UK")

Arguments

x a vector representing the input for which one wishes to calculate EMI,
model list of objects of class km, one for each objective functions,
**crit_EMI**

paretoFront (optional) matrix corresponding to the Pareto front of size \([n\text{.pareto} \times n\text{.obj}]\), or any reference set of observations,

critcontrol optional list with arguments (for more than 2 objectives only):

- nb.samp number of random samples from the posterior distribution, default to 50, increasing gives more reliable results at the cost of longer computation time;
- seed seed used for the random samples.

Options for the checkPredict function: threshold(1e-4) and distance(covdist) are used to avoid numerical issues occurring when adding points too close to the existing ones.

type "SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account.

**Details**

It is recommended to scale objectives, e.g. to \([0, 1]\). If the Pareto front does not belong to \([-2, 2]^2\), then SAA is used.

**Value**

The Expected Maximin Improvement at \(x\).

**References**


**See Also**

EI from package DiceOptim, crit_EHI, crit_SUR, crit_SMS.

**Examples**

```r
set.seed(25468)
library(DiceDesign)

n_var <- 2
f_name <- "P1"
n.grid <- 21
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
n.appr <- 15
design.grid <- round(maximinESE_LHS(1hsDesign(n_appr, n_var, seed = 42)$design)$design, 1)
response.grid <- t(apply(design.grid, 1, f_name))
```
crit_optimizer

Maximization of multiobjective infill criterion

Description

Given a list of objects of class km and a set of tuning parameters (lower, upper and critcontrol), crit_optimizer performs the maximization of an infill criterion and delivers the next point to be visited in a multi-objective EGO-like procedure.

The latter maximization relies either on a genetic algorithm using derivatives, genoud, particle swarm algorithm pso, exhaustive search at pre-specified points or on a user defined method. It is important to remark that the information needed about the objective function reduces here to the vector of response values embedded in the models (no call to the objective functions or simulators (except with cheapfn)).

Usage

crit_optimizer(crit = "SMS", model, lower, upper, cheapfn = NULL,
               type = "UK", paretoFront = NULL, critcontrol = NULL,
               optimcontrol = NULL)

Arguments

crit sampling criterion. Four choices are available: "SMS", "EHI", "EMI" and "SUR",
model list of objects of class km, one for each objective functions,
lower vector of lower bounds for the variables to be optimized over,
upper vector of upper bounds for the variables to be optimized over,
cheapfn optional additional fast-to-evaluate objective function (handled next with class fastfun), which does not need a kriging model,
type "SK" or "UK" (default), depending whether uncertainty related to trend estimation has to be taken into account.
paretoFront    (optional) matrix corresponding to the Pareto front of size \([n_{\text{pareto}} \times n_{\text{obj}}]\), or any reference set of observations,
critcontrol   optional list of control parameters for criterion crit, see details. Options for the checkPredict function: threshold (1e-4) and distance (covdist) are used to avoid numerical issues occurring when adding points too close to the existing ones.
optimcontrol   optional list of control parameters for optimization of the selected infill criterion. "method" set the optimization method; one can choose between "discrete", "pso" and "genoud" or a user defined method name (passed to match.fun). For each method, further parameters can be set. For "discrete", one has to provide the argument "candidate.points". For "pso", one can control the maximum number of iterations "maxit" (400) and the population size "s" (default: max(20, floor(10+2*sqrt(length(dim()))) (see psoptim). For "genoud", one can control, among others, "pop.size" (default: \([N = 3*2^*\text{dim for dim < 6 and N = 32*dim otherwise}]\), "max.generations" (12), "wait.generations" (2), "BFGSburnin" (2), BFGSmaxit (N) and solution.tolerance (1e-21) of function "genoud" (see genoud). Numbers into brackets are the default values. For a user defined method, it must have arguments like the default optim method, i.e. par, fn, lower, upper, ... and possibly control, and return a list with par and value. A trace trace argument is available, it can be set to 0 to suppress all messages, to 1 (default) for displaying the optimization progresses, and >1 for the highest level of details.

**Details**

Extension of the function max_EI for multi-objective optimization. Available infill criteria with crit are:

- Expected Hypervolume Improvement (EHI) **crit_EHI**,
- SMS criterion (SMS) **crit_SMS**,  
- Expected Maximin Improvement (EMI) **crit_EMI**,  
- Stepwise Uncertainty Reduction of the excursion volume (SUR) **crit_SUR**

Depending on the selected criterion, parameters such as a reference point for SMS and EHI or arguments for integration_design_optim with SUR can be given with critcontrol. Also options for checkPredict are available. More precisions are given in the corresponding help pages.

**Value**

A list with components:

- **par**: The best set of parameters found,
- **value**: The value of expected improvement at par.
**References**


**Examples**

```r
# Not run:
#---------------------------------------------------------------
# EHI surface associated with the "P1" problem at a 15 points design
#---------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

d <- 2
n.obj <- 2
fname <- "P1"
n.grid <- 51
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
nappr <- 15
design.grid <- round(maximinESE_LHS(lhsDesign(nappr, d, seed = 42)$design)$design, 1)
response.grid <- t(apply(design.grid, 1, fname))
paretoFront <- t(nondominated_points(t(response.grid)))

mf1 <- km(".", design = design.grid, response = response.grid[,1])
mf2 <- km(".", design = design.grid, response = response.grid[,2])

model <- list(mf1, mf2)

EHI_grid <- apply(test.grid, 1, crit_EHI, model = list(mf1, mf2),
                   critcontrol = list(refPoint = c(300, 0)))

lower <- rep(0, d)
upper <- rep(1, d)

omEGO <- crit_optimizer(crit = "EHI", model = model, lower = lower, upper = upper,
                         optimcontrol = list(method = "genoud", pop.size = 200, BFGSburnin = 2),
                         critcontrol = list(refPoint = c(300, 0)))

print(omEGO)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
              matrix(EHI_grid, nrow = n.grid), main = "Expected Hypervolume Improvement",
              xlab = expression(x[1]), ylab = expression(x[2]), color = terrain.colors,
              plot.axes = {axis(1); axis(2);}
              points(design.grid[, 1], design.grid[, 2], pch = 21, bg = "white"),
              points(omEGO$par, col = "red", pch = 4)
```
# SMS surface associated with the "P1" problem at a 15 points design

SMS_grid <- apply(test.grid, 1, crit_SMS, model = model, 
critcontrol = list(refPoint = c(300, 0)))

lower <- rep(0, d)
upper <- rep(1, d)

omEGO2 <- crit_optimizer(crit = "SMS", model = model, lower = lower, upper = upper, 
optimcontrol = list(method="genoud", pop.size = 200, BFGSburnin = 2), 
critcontrol = list(refPoint = c(300, 0)))

print(omEGO2)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50, 
matrix(pmax(0, SMS_grid), nrow = n.grid), main = "SMS Criterion (>0)", 
xlab = expression(x[1]), ylab = expression(x[2]), color = terrain.colors, 
plot.axes = (axis(1); axis(2));
points(design.grid[, 1], design.grid[, 2], pch = 21, bg = "white");
points(omEGO2$par, col = "red", pch = 4)

# Maximin Improvement surface associated with the "P1" problem at a 15 points design

#---------------------------------------------------------------

EMI_grid <- apply(test.grid, 1, crit_EMI, model = model, 
critcontrol = list(nb_samp = 20, type ="EMI"))

lower <- rep(0, d)
upper <- rep(1, d)

omEGO3 <- crit_optimizer(crit = "EMI", model = model, lower = lower, upper = upper, 
optimcontrol = list(method = "genoud", pop.size = 200, BFGSburnin = 2))

print(omEGO3)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50, 
matrix(EMI_grid, nrow = n.grid), main = "Expected Maximin Improvement", 
xlab = expression(x[1]), ylab = expression(x[2]), color = terrain.colors, 
plot.axes = (axis(1); axis(2));
points(design.grid[, 1], design.grid[, 2], pch = 21, bg = "white");
points(omEGO3$par, col = "red", pch = 4)

#---------------------------------------------------------------

# crit_SUR surface associated with the "P1" problem at a 15 points design

library(KrigInv)
crit_optimizer

integration.param <- integration_design_optim(lower = c(0, 0), upper = c(1, 1), model = model)
integration.points <- as.matrix(integration.param$integration.points)
integration.weights <- integration.param$integration.weights

precalc.data <- list()
mm.X <- sn.X <- matrix(0, n.obj, nrow(integration.points))

for (i in 1:n.obj){
  p.tst.all <- predict(model[[i]], newdata = integration.points, type = "UK",
                      checkNames = FALSE)
  mm.X[i,] <- p.tst.all$mean
  sn.X[i,] <- p.tst.all$sd
  precalc.data[[i]] <- precomputeUpdateData(model[[i]], integration.points)
}
critcontrol <- list(integration.points = integration.points,
                     integration.weights = integration.weights,
                     mm.X = mm.X, sn.X = sn.X, precalc.data = precalc.data)

EEV_grid <- apply(test.grid, 1, crit_SUR, model=model, paretoFront = paretoFront,
                   critcontrol = critcontrol)
lower <- rep(0, d)
upper <- rep(1, d)

omEGO4 <- crit_optimizer(crit = "SUR", model = model, lower = lower, upper = upper,
                          optimcontrol = list(method = "genoud", pop.size = 200, BFGSburnin = 2))
print(omEGO4)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
               matrix(pmax(0,EEV_grid), n.grid), main = "EEV criterion", nlevels = 50,
               xlab = expression(x[1]), ylab = expression(x[2]), color = terrain.colors,
               plot.axes = (axis(1); axis(2);
               points(test.grid[,1], test.grid[,2], pch = 21, bg = "white")
               points(omEGO4$par, col = "red", pch = 4)
)

# example using user defined optimizer, here L-BFGS-B from base optim
userOptim <- function(par, fn, lower, upper, control, ...){
  return(optim(par = par, fn = fn, method = "L-BFGS-B", lower = lower, upper = upper,
               control = control, ...))
}
omEGO4bis <- crit_optimizer(crit = "SUR", model = model, lower = lower, upper = upper,
                            optimcontrol = list(method = "userOptim"))
print(omEGO4bis)

#----------------------------------------------------------
# crit_SMS surface with problem "P1" with 15 design points, using cheapfn
#----------------------------------------------------------

# Optimization with fastfun: SMS with discrete search
# Separation of the problem P1 in two objectives:
# the first one to be kriged, the second one with fastobj
# Definition of the fastfun
f2 <- function(x){
    return(P1(x)[2])
}

SMS_grid_cheap <- apply(test.grid, 1, crit_SMS, model = list(mf1, fastfun(f2, design.grid)),
    paretoFront = paretoFront, critcontrol = list(refPoint = c(300, 0)))

optimcontrol <- list(method = "pso")
model2 <- list(mf1)
omEGO5 <- crit_optimizer(crit = "SMS", model = model2, lower = lower, upper = upper,
    cheapfn = f2, critcontrol = list(refPoint = c(300, 0)),
    optimcontrol = list(method = "genoud", pop.size = 200, BFGSburnin = 2))

print(omEGO5)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
    matrix(pmax(0, SMS_grid_cheap), nrow = n.grid), nlevels = 50,
    main = "SMS criterion with cheap 2nd objective (>0)", xlab = expression(x[1]),
    ylab = expression(x[2]), color = terrain.colors,
    plot.axes = (axis(1); axis(2);
    points(design.grid[,1], design.grid[,2], pch = 21, bg = "white")
    points(omEGO5$par, col = "red", pch = 4)
)

## End(Not run)

crit_SMS

Analytical expression of the SMS-EGO criterion with m>1 objectives

Description
Computes a slightly modified infill Criterion of the SMS-EGO. To avoid numerical instabilities, an additional penalty is added to the new point if it is too close to an existing observation.

Usage
crit_SMS(x, model, paretoFront = NULL, critcontrol = NULL,
    type = "UK")

Arguments

- **x**: a vector representing the input for which one wishes to calculate the criterion,
- **model**: a list of objects of class km (one for each objective),
- **paretoFront**: (optional) matrix corresponding to the Pareto front of size [n.pareto x n.obj], or any reference set of observations,
- **critcontrol**: list with arguments:
crit_SMS

- currentHV: current hypervolume;
- refPoint: reference point for hypervolume computations;
- extendper: if no reference point refPoint is provided, for each objective it is fixed to the maximum over the Pareto front plus extendper times the range. Default value to 0.2, corresponding to 1.1 for a scaled objective with a Pareto front in \([0,1]^n\,\text{obj}\);
- epsilon: optional value to use in additive epsilon dominance;
- gain: optional gain factor for sigma.

Options for the checkPredict function: threshold(1e-4) and distance(covdist) are used to avoid numerical issues occurring when adding points too close to the existing ones.

type

"SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account.

Value

Value of the criterion.

References


See Also

crit_EHI, crit_SUR, crit_EMI.

Examples

```r
# SMS-EGO surface associated with the "P1" problem at a 15 points design
set.seed(25468)
library(DiceDesign)

n_var <- 2
f_name <- "P1"
n.grid <- 26
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
n_appr <- 15
design.grid <- round(maximinESE_LHS(LhsDesign(n_appr, n_var, seed = 42)$design)$design, 1)
response.grid <- t(apply(design.grid, 1, f_name))
PF <- t(nondominated_points(t(response.grid)))
mfl <- km(-., design = design.grid, response = response.grid[,1])
```
mf2 <- km(-., response = response.grid[,2])

crit_sur <- function(x, model, paretoFront = NULL, critcontrol = NULL, type = "UK")
{
  # Code for crit_sur function
}

crit_sur

Description

Computes the SUR criterion (Expected Excursion Volume Reduction) at point x for 2 or 3 objectives. To avoid numerical instabilities, the new point is penalized if it is too close to an existing observation.

Usage

crit_sur(x, model, paretoFront = NULL, critcontrol = NULL, type = "UK")

Arguments

x a vector representing the input for which one wishes to calculate the criterion,
model a list of objects of class km (one for each objective),
paretoFront (optional) matrix corresponding to the Pareto front of size [n.pareto x n.obj], or any reference set of observations,
critcontrol list with two possible options.

A) One can use the four following arguments:
  • integration.points, matrix of integration points of size [n.integ.pts x d];
  • integration.weights, vector of integration weights of length n.integ.pts;
  • mn.X and sn.X, matrices of kriging means and sd, each of size [n.obj x n.integ.pts];
  • precalc.data, list of precalculated data (based on kriging models at integration points) for faster computation.
B) Alternatively, one can define arguments passed to `integration_design_optim`: `SURcontrol` (optional), `lower`, `upper`, `min.prob` (optional). This is slower since arguments of A), used in the function, are then recomputed each time (note that this is not the case when called from `GParetoptim` and `crit_optimizer`).

Options for the `checkPredict` function: `threshold` (1e-4) and `distance` (covidist) are used to avoid numerical issues occuring when adding points too close to the existing ones.

`type` "SK" or "UK" (default), depending whether uncertainty related to trend estimation has to be taken into account.

**Value**

Value of the criterion.

**References**


**See Also**

`crit_EHI, crit_SMS, crit_EMI`.

**Examples**

```r
#-----------------------------
# crit_SUR surface associated with the "P1" problem at a 15 points design
#-----------------------------
set.seed(25468)
library(DiceDesign)
library(KrigInv)

n_var <- 2
n_obj <- 2
f_name <- "P1"
n.grid <- 14
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
n_appr <- 15
design.grid <- round(minimESE_LHS(lhsDesign(n_appr, n_var, seed = 42)$design)$design, 1)
response.grid <- t(apply(design.grid, 1, f_name))
paretoFront <- t(nondominated_points(t(response.grid)))

mf1 <- km(~., design = design.grid, response = response.grid[,1])
mf2 <- km(~., design = design.grid, response = response.grid[,2])
model <- list(mf1, mf2)

integration.param <- integration_design_optim(lower = c(0, 0), upper = c(1, 1), model = model)
integration.points <- as.matrix(integration.param$integration.points)
integration.weights <- integration.param$integration.weights
```
easyGParetoptim <- list()

for (i in 1:n.obj){
  p.tst.all <- predict(model[[i]], newdata = integration.points, type = "UK", checkNames = FALSE)
  mn.X[i,] <- p.tst.all$mean
  sn.X[i,] <- p.tst.all$sd
  precalc.data[i] <- precomputeUpdatedData(model[[i]], integration.points)
}

critcontrol <- list(integration.points = integration.points,
                      integration.weights = integration.weights,
## Alternatively: critcontrol <- list(lower = rep(0, n_var), upper = rep(1,n_var))

EEV_grid <- apply(test.grid, 1, crit_SUR, model = model, paretoFront = paretoFront,
                   critcontrol = critcontrol)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
              matrix(pmax(0,EEV_grid), nrow = n.grid), main = "EEV criterion",
              xlab = expression(x[1]), ylab = expression(x[2]), color = terrain.colors,
              plot.axes = (axis(1); axis(2);
              points(design.grid[,1], design.grid[,2], pch = 21, bg = "white")
)

---

easyGParetoptim  

**EGO algorithm for multiobjective optimization**

---

**Description**

User-friendly wrapper of the function G Paretoptim. Generates initial DOEs and kriging models
(objects of class km), and executes nsteps iterations of multiobjective EGO methods.

**Usage**

easyGParetoptim(fn, cheapfn = NULL, budget, lower, upper, par = NULL,
                 value = NULL, noise.var = NULL, control = list(method = "SMS",
                 trace = 1, inneroptim = "pso", maxit = 100, seed = 42), ...)

**Arguments**

- **fn**  
  the multi-objective function to be minimized (vectorial output), found by a call
to match.fun, see details,

- **cheapfn**  
  optional additional fast-to-evaluate objective function (handled next with class
fastfun), which does not need a kriging model, handled by a call to match.fun,

- **budget**  
  total number of calls to the objective function,
lower vector of lower bounds for the variables to be optimized over,
upper vector of upper bounds for the variables to be optimized over,
par initial design of experiments. If not provided, par is taken as a maximin LHD with budget/3 points,
value initial set of objective observations \( fn(\text{par}) \). Computed if not provided. Not that value may NOT contain any cheapfn value,
optvar optional noise variance, for noisy objectives \( fn \). If not NULL, either a scalar (constant noise, identical for all objectives), a vector (constant noise, different for each objective) or a function (type closure) with vectorial output (variable noise, different for each objective). Alternatively, set optvar="given_by_fn", see details.
control an optional list of control parameters. See "Details",
... additional parameters to be given to the objective \( fn \).

Details

Does not require specific knowledge on kriging models (objects of class \( \text{km} \)).

The problem considered is of the form: \( \min f(x) = f_1(x), ..., f_p(x) \). The control argument is a list that can supply any of the following optional components:

- method: choice of multiobjective improvement function: "SMS", "EHI", "EMI" or "SUR" (see crit_SMS, crit_EHI, crit_EMI, crit_SUR),
- trace: if positive, tracing information on the progress of the optimization is produced (1 (default) for general progress, >1 for more details, e.g., warnings from \text{genoud}),
- inneroptim: choice of the inner optimization algorithm: "genoud", "pso" or "random" (see \text{genoud} and \text{psoptim}),
- maxit: maximum number of iterations of the inner loop,
- seed: to fix the random variable generator,
- refPoint: reference point for hypervolume computations (for "SMS" and "EHI" methods),
- extendper: if no reference point refPoint is provided, for each objective it is fixed to the maximum over the Pareto front plus extendper times the range. Default value to 0.2, corresponding to 1.1 for a scaled objective with a Pareto front in \( [0, 1]^n \).obj.

If optvar="given_by_fn", \( fn \) must return a list of two vectors, the first being the objective functions and the second the corresponding noise variances. See examples in \text{GParetoptim}.

For additional details or other possible arguments, see \text{GParetoptim}.

Display of results and various post-processings are available with \text{plotGPareto}.
Value

A list with components:

- `par`: all the non-dominated points found,
- `value`: the matrix of objective values at the points given in `par`,
- `history`: a list containing all the points visited by the algorithm (X) and their corresponding objectives (y).
- `model`: a list of objects of class `km`, corresponding to the last kriging models fitted.

Note that in the case of noisy problems, `value` and `history` are denoised values. The original observations are available in the slot `historyDy`.

Author(s)

Victor Picheny (INRA, Castanet-Tolosan, France)
Mickael Binois (Mines Saint-Etienne/Renault, France)

References


Examples

```r
# 2D objective function, 4 cases
## Not run:
set.seed(25468)
n_var <- 2
fname <- ZDT3
lower <- rep(0, n_var)
upper <- rep(1, n_var)

# Expected Hypervolume Improvement optimization, using pso
res <- easyGParetoptim(fn=fname, lower=lower, upper=upper, budget=15,
...```
```r
easyGParetooptm

control=list(method="EHI", inneroptim="pso", maxit=20))
par(mfrow=c(1,2))
plotGPareto(res)
title("Pareto Front")
plot(res$history$X, main="Pareto set", col = "red", pch = 20)
points(res$par, col="blue", pch = 17)

# 2- SMS Improvement optimization using random search, with initial DOE given
library(DiceDesign)
design.init <- maximinESE_LHS(lhsDesign(10, n_var, seed = 42)$design)$design
response.init <- t(apply(design.init, 1, fn))
res <- easyGParetooptm(fn=fn, par=design.init, value=response.init, lower=lower, upper=upper,
                      budget=15, control=list(method="SMS", inneroptim="random", maxit=100))
par(mfrow=c(1,2))
plotGPareto(res)
title("Pareto Front")
plot(res$history$X, main="Pareto set", col = "red", pch = 20)
points(res$par, col="blue", pch = 17)

# 3- Stepwise Uncertainty Reduction optimization, with one fast objective function
fname <- camelback
cheapfn <- function(x) {
  if (is.null(dim(x))) return(-sum(x))
  else return(-rowSums(x))
}
res <- easyGParetooptm(fn=fname, cheapfn=cheapfn, lower=lower, upper=upper, budget=15,
                      control=list(method="SUR", inneroptim="pso", maxit=20))
par(mfrow=c(1,2))
plotGPareto(res)
title("Pareto Front")
plot(res$history$X, main="Pareto set", col = "red", pch = 20)
points(res$par, col="blue", pch = 17)

# 4- Expected Hypervolume Improvement optimization, using pso, noisy fn
noise.var <- c(0.1, 0.2)
funnoise <- function(x) {ZDT3(x) + sqrt(noise.var)*rnorm(n=2)}
res <- easyGParetooptm(fn=funnoise, lower=lower, upper=upper, budget=30, noise.var=noise.var,
                       control=list(method="EHI", inneroptim="pso", maxit=20))
par(mfrow=c(1,2))
plotGPareto(res)
title("Pareto Front")
plot(res$history$X, main="Pareto set", col = "red", pch = 20)
points(res$par, col="blue", pch = 17)

# 5- Stepwise Uncertainty Reduction optimization, functional noise
```
funnoise <- function(x) {ZDT3(x) + sqrt(abs(0.1*x)) * rnorm(n=2)}
noise.var <- function(x) return(abs(0.1*x))

res <- easyGParetoptim(fn=funnoise, lower=lower, upper=upper, budget=30, noise.var=noise.var, 
control=list(method="SUR", inneroptim="pso", maxit=20))
par(mfrow=c(1,2))
plotGPareto(res)
title("Pareto Front")
plot(res$history$X, main="Pareto set", col = "red", pch = 20)
points(res$par, col="blue", pch = 17)

## End(Not run)

---

### fastfun

**Fast-to-evaluate function wrapper**

### Description

Modification of an R function to be used with methods `predict` and `update` (similar to a `km` object). It creates an S4 object which contains the values corresponding to evaluations of other costly observations. It is useful when an objective can be evaluated fast.

### Usage

```r
fastfun(fn, design, response = NULL)
```

### Arguments

- **fn**: the evaluator function, found by a call to `match.fun`.
- **design**: a data frame representing the design of experiments. The ith row contains the values of the d input variables corresponding to the ith evaluation.
- **response**: optional vector (or 1-column matrix or data frame) containing the values of the 1-dimensional output given by the objective function at the design points.

### Value

An object of class `fastfun-class`.

### Examples

```r
# Example with a fast to evaluate objective
set.seed(25468)
library(DiceDesign)
d <- 2
```
fname <- P1
n.grid <- 21
nappr <- 11
design.grid <- maximinESE.LHS(lhsDesign(nappr, d, seed = 42)$design)$design
response.grid <- t(apply(design.grid, 1, fname))
Front_Pareto <- t(nondominated_points(t(response.grid)))

mf1 <- km(., design = design.grid, response = response.grid[,1])
mf2 <- km(., design = design.grid, response = response.grid[,2])
model <- list(mf1, mf2)
nsteps <- 5
lower <- rep(0, d)
upper <- rep(1, d)

# Optimization reference: SMS with discrete search
optimcontrol <- list(method = "psd")
omEGO1 <- GPararetoPoptim(model = model, fn = fname, crit = "SMS", nsteps = nsteps,
                          lower = lower, upper = upper, optimcontrol = optimcontrol)
print(omEGO1$par)
print(omEGO1$values)
plot(response.grid, xlim = c(0, 300), ylim = c(-40, 0), pch = 17, col = "blue")
points(omEGO1$values, pch = 20, col = "green")

# Optimization with fastfun: SMS with discrete search
# Separation of the problem P1 in two objectives:
# the first one to be kriged, the second one with fastobj
f1 <- function(x){
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  b1 <- 15*x[,1] - 5
  b2 <- 15*x[,2]
  return( (b2 - 5.1*(b1/(2*pi))ˆ2 + 5/pi*b1 - 6)ˆ2 + 10*((1 - 1/(8*pi))*cos(b1) + 1) )
}
f2 <- function(x){
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  b1<-15*x[,1] - 5
  b2<-15*x[,2]
  return(-sqrt((10.5 - b1)*(b1 + 5.5)*(b2 + 0.5))
         - 1/30*(b2 - 5.1*(b1/(2*pi))ˆ2 - 6)ˆ2
         - 1/3*((1 - 1/(8*pi))*cos(b1) + 1))
}

optimcontrol <- list(method = "psd")
model2 <- list(mf1)
omEGO2 <- GPararetoPoptim(model = model2, fn = f1, cheapfn = f2, crit = "PSD", nsteps = nsteps,
                          lower = lower, upper = upper, optimcontrol = optimcontrol)
print(omEGO2$par)
print(omEGO2$values)
points(omEGO2$values, col = "red", pch = 15)

## End(Not run)
getDesign

Get design corresponding to an objective target

Description

Find the design that maximizes the probability of dominating a target given by the user.

Usage

getDesign(model, target, lower, upper, optimcontrol = NULL)

Arguments

model list of objects of class km, one for each objective functions,
target vector corresponding to the desired output in the objective space,
lower vector of lower bounds for the variables to be optimized over,
upper vector of upper bounds for the variables to be optimized over,
optimcontrol optional list of control parameters for optimization of the selected infill criterion.

"method" set the optimization method; one can choose between "discrete", "pso" and "genoud". For each method, further parameters can be set.
For "discrete", one has to provide the argument "candidate.points".
For "pso", one can control the maximum number of iterations "maxit" (400) and the population size "s" (default: max(20, floor(10+2*sqrt(length(dim)))) (see psoptim).
For "genoud", one can control, among others, "pop.size" (default: \(N = 3*2^\text{dim}\) for \text{dim} < 6 and \(N = 32*\text{dim}\) otherwise), "max.generations" (12), "wait.generations" (2), "BFGSburnin" (2), BFGSmaxit (N) and solution.tolerance (1e-21) of function "genoud" (see genoud). Numbers into brackets are the default values.

Value

A list with components:

- par: best design found,
- value: probability that the design dominates the target,
- mean: kriging mean of the objectives at the design,
- sd: prediction standard deviation at the design.

Examples

## Not run:

#---------------------------------------------------------------
# Example of interactive optimization
#---------------------------------------------------------------
set.seed(25468)
library(DiceDesign)

d <- 2
n.obj <- 2
fun <- "P1"
n.grid <- 51
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
nappr <- 20
design.grid <- round(maximinES_LHS(lhsDesign(nappr, d, seed = 42)$design)$design, 1)
response.grid <- t(apply(design.grid, 1, fun))
paretoFront <- t(nondominated_points(t(response.grid)))
mf1 <- km(~., design = design.grid, response = response.grid[,1])
mf2 <- km(~., design = design.grid, response = response.grid[,2])
model <- list(mf1, mf2)
lower <- rep(0, d); upper <- rep(1, d)

sol <- G Paretoptim(model, fun, crit = "SUR", nsteps = 5, lower = lower, upper = upper)

plotGPareto(sol)

target1 <- c(15, -25)
points(x = target1[1], y = target1[2], col = "black", pch = 13)

nDesign <- getDesign(sol$lastmodel, target = target1, lower = rep(0, d), upper = rep(1, d))
points(t(nDesign$mean), col = "green", pch = 20)

target2 <- c(48, -27)
points(x = target2[1], y = target2[2], col = "black", pch = 13)

nDesign2 <- getDesign(sol$lastmodel, target = target2, lower = rep(0, d), upper = rep(1, d))
points(t(nDesign2$mean), col = "darkgreen", pch = 20)

## End(Not run)

---

**GParetoptim**

*Sequential multi-objective Expected Improvement maximization and model re-estimation, with a number of iterations fixed in advance by the user*

**Description**

Executes `nsteps` iterations of multi-objective EGO methods to objects of class `km`. At each step, kriging models are re-estimated (including covariance parameters re-estimation) based on the initial design points plus the points visited during all previous iterations; then a new point is obtained by maximizing one of the four multi-objective Expected Improvement criteria available. Handles noiseless and noisy objective functions.

**Usage**

```r
GParetoptim(model, fn, cheapfn = NULL, crit = "SMS", nsteps, lower, upper,)
```
upper, type = "UK", cov.reestim = TRUE, critcontrol = NULL,
noise.var = NULL, reinterpolation = NULL,
optimcontrol = list(method = "genoud", trace = 1), ...

Arguments

- **model**: list of objects of class km, one for each objective functions,
- **fn**: the multi-objective function to be minimized (vectorial output), found by a call to match.fun,
- **cheapfn**: optional additional fast-to-evaluate objective function (handled next with class fastfun), which does not need a kriging model, handled by a call to match.fun,
- **crit**: choice of multi-objective improvement function: "SMS", "EHI", "EMI" or "SUR", see details below,
- **nsteps**: an integer representing the desired number of iterations,
- **lower**: vector of lower bounds for the variables to be optimized over,
- **upper**: vector of upper bounds for the variables to be optimized over,
- **type**: "SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account, see km
- **cov.reestim**: optional boolean specifying if the kriging hyperparameters should be re-estimated at each iteration,
- **critcontrol**: optional list of parameters for criterion crit, see details,
- **noise.var**: noise variance (of the objective functions). Either NULL (noiseless objectives), a scalar (constant noise, identical for all objectives), a vector (constant noise, different for each objective) or a function (type closure) with vectorial output (variable noise, different for each objective). Alternatively, set noise.var = "given_by_fn", see details. If not provided but km models are based on noisy observations, noise.var is taken as the average of model@noise.var,
- **reinterpolation**: Boolean: for noisy problems, indicates whether a reinterpolation model is used, see details,
- **optimcontrol**: an optional list of control parameters for optimization of the selected infill criterion: "method" can be set to "discrete", "pso", "genoud" or a user defined method name (passed to match.fun). For "discrete", a matrix candidate.points must be given. For "pso" and "genoud", specific parameters to the chosen method can also be specified (see genoud and psoptim). A user defined method must have arguments like the default optim method, i.e. par, fn, lower, upper, ... and eventually control.
- **trace** argument is available, it can be set to 0 to suppress all messages, to 1 (default) for displaying the optimization progresses, and >1 for the highest level of details.
- **...**: additional parameters to be given to the objective fn.
Details

Extension of the function `EG0.nsteps` for multi-objective optimization.

Available infill criteria with `crit` are:

- Expected Hypervolume Improvement (EHI) `crit_EHI`,
- SMS criterion (SMS) `crit_SMS`,
- Expected Maximin Improvement (EMI) `crit_EMI`,
- Stepwise Uncertainty Reduction of the excursion volume (SUR) `crit_SUR`.

Depending on the selected criterion, parameters such as reference point for SMS and EHI or arguments for `integration_design_optim` with SUR can be given with `critcontrol`. Also options for `checkPredict` are available. More precisions are given in the corresponding help pages.

The `reinterpolation=TRUE` setting can be used to handle noisy objective functions. It works with all criteria and is the recommended option. If `reinterpolation=FALSE` and `noise.var!=NULL`, the criteria are used based on a "denoised" Pareto front.

If `noise.var="given_by_fn"`, `fn` must return a list of two vectors, the first being the objective functions and the second the corresponding noise variances (see examples).

Display of results and various post-processings are available with `plotGPareto`.

Value

A list with components:

- `par`: a data frame representing the additional points visited during the algorithm,
- `values`: a data frame representing the response values at the points given in `par`,
- `nsteps`: an integer representing the desired number of iterations (given in argument).
- `lastmodel`: a list of objects of class `km` corresponding to the last kriging models fitted.
- `observations.denoised`: if `noise.var!=NULL`, a matrix representing the mean values of the `km` models at observation points. If a problem occurs during either model updates or criterion maximization, the last working model and corresponding values are returned.

References


Examples

```r
set.seed(25468)
library(DiceDesign)

# NOISELESS PROBLEMS

d <- 2
fname <- ZDT3
n.grid <- 21
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))
nappr <- 15
design.grid <- maximinESE_LHS(lhsDesign(nappr, d, seed = 42)$design)$design
response.grid <- t(apply(design.grid, 1, fname))
Front_Pareto <- t(nondominated_points(t(response.grid)))

mf1 <- km(-1, design = design.grid, response = response.grid[, 1], lower = c(0.1, 1))
mf2 <- km(-., design = design.grid, response = response.grid[, 2], lower = c(0.1, 1))
model <- list(mf1, mf2)

nsteps <- 2
lower <- rep(0, d)
upper <- rep(1, d)

# Optimization 1: EHI with pso
optimcontrol <- list(method = "pso", maxit = 20)
critcontrol <- list(refPoint = c(1, 10))
omoEGO1 <- GParetoptim(model = model, fn = fname, crit = "EHI", nsteps = nsteps,
lower = lower, upper = upper, critcontrol = critcontrol,
optimcontrol = optimcontrol)
print(omoEGO1$par)
print(omoEGO1$values)

## Not run:
nsteps <- 10
# Optimization 2: SMS with discrete search
optimcontrol <- list(method = "discrete", candidate.points = test.grid)
critcontrol <- list(refPoint = c(1, 10))
omoEGO2 <- GParetoptim(model = model, fn = fname, crit = "SMS", nsteps = nsteps,
lower = lower, upper = upper, critcontrol = critcontrol,
optimcontrol = optimcontrol)
print(omoEGO2$par)
print(omoEGO2$values)

# Optimization 3: SUR with genoud
optimcontrol <- list(method = "genoud", pop.size = 20, max.generations = 10)
```

critcontrol <- list(distrib = "SUR", n.points = 100)
omEGO3 <- GPararetoptim(model = model, fn = fname, crit = "SUR", nsteps = nsteps,
  lower = lower, upper = upper, critcontrol = critcontrol,
  optimcontrol = optimcontrol)

print(omEGO3$par)
print(omEGO3$values)

# Optimization 4: EMI with pso
optimcontrol <- list(method = "pso", maxit = 20)
omEGO4 <- GPararetoptim(model = model, fn = fname, crit = "EMI", nsteps = nsteps,
  lower = lower, upper = upper, optimcontrol = optimcontrol)

print(omEGO4$par)
print(omEGO4$values)

# graphics
sol.grid <- apply(expand.grid(seq(0, 1, length.out = 100),
  seq(0, 1, length.out = 100)), 1, fname)
plot(t(sol.grid), pch = 20, col = rgb(0, 0, 0.05), xlim = c(0, 1),
  ylim = c(-2, 10), xlab = expression(f[1]), ylab = expression(f[2]))
plotGPareto(res = omEGO1, add = TRUE,
  control = list(pch = 20, col = "blue", PF.pch = 17,
    PF.points.col = "blue", PF.line.col = "blue"))

plotGPareto(res = omEGO2, add = TRUE,
  control = list(pch = 20, col = "green", PF.pch = 17,
    PF.points.col = "green", PF.line.col = "green"))

plotGPareto(res = omEGO3, add = TRUE,
  control = list(pch = 20, col = "red", PF.pch = 17,
    PF.points.col = "red", PF.line.col = "red"))

plotGPareto(res = omEGO4, add = TRUE,
  control = list(pch = 20, col = "orange", PF.pch = 17,
    PF.points.col = "orange", PF.line.col = "orange"))

legend("topright", c("EMI", "SUR", "EMI"), col = c("blue", "green", "red", "orange"),
  pch = rep(17, 4))

# Post-processing
plotGPareto(res = omEGO1, UQ_PF = TRUE, UQ_PS = TRUE, UQ_dens = TRUE)

###################################################
# NOisy PROBLEMS
###################################################
set.seed(25468)
library(DiceDesign)
d <- 2
nsteps <- 3
lower <- rep(0, d)
upper <- rep(1, d)
optimcontrol <- list(method = "psdo", maxit = 20)
critcontrol <- list(refPoint = c(1, 10))

n.grid <- 21
test.grid <- expand.grid(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid))

n.init <- 30
design <- maximinESE.LHS(lhsDesign(n.init, d, seed = 42))

fit.models <- function(u) km(~., design = design, response = response[, u],
            noise.var = design.noise.var[, u])

# Test 1: EHI, constant noise.var
noise.var <- c(0.1, 0.2)
funnoise1 <- function(x) {ZDT3(x) + sqrt(noise.var)*rnorm(n=d)}
response <- t(apply(design, 1, funnoise1))
design.noise.var <- matrix(rep(noise.var, n.init), ncol=d, byrow=TRUE)
model <- lapply(1:d, fit.models)

omEGO1 <- GParetoptim(model = model, fn = funnoise1, crit = "EHI", nsteps = nsteps,
            lower = lower, upper = upper, critcontrol = critcontrol,
            reinterpolation=TRUE, noise.var=design.noise.var, optimcontrol = optimcontrol)
plotGPareto(omEGO1)

# Test 2: EMI, noise.var given by fn
funnoise2 <- function(x) {list(ZDT3(x) + sqrt(0.05 + abs(0.1*x)))*rnorm(n=d), 0.05 + abs(0.1*x))}
temp <- funnoise2(design)
response <- temp[[1]]
design.noise.var <- temp[[2]]
model <- lapply(1:d, fit.models)

omEGO2 <- GParetoptim(model = model, fn = funnoise2, crit = "EMI", nsteps = nsteps,
            lower = lower, upper = upper, critcontrol = critcontrol,
            reinterpolation=TRUE, noise.var="given_by_fn", optimcontrol = optimcontrol)
plotGPareto(omEGO2)

# Test 3: SMS, functional noise.var
funnoise3 <- function(x) {ZDT3(x) + sqrt(0.025 + abs(0.05*x))*rnorm(n=d)}
noise.var <- function(x) return(0.025 + abs(0.05*x))
response <- t(apply(design, 1, funnoise3))
design.noise.var <- t(apply(design, 1, noise.var))
model <- lapply(1:d, fit.models)

omEGO3 <- GParetoptim(model = model, fn = funnoise3, crit = "SMS", nsteps = nsteps,
            lower = lower, upper = upper, critcontrol = critcontrol,
            reinterpolation=TRUE, noise.var=design.noise.var, optimcontrol = optimcontrol)
plotGPareto(omEGO3)

# Test 4: SUR, fastfun, constant noise.var
noise.var <- 0.1
funnoise4 <- function(x) {ZDT3(x)[1] + sqrt(noise.var)*rnorm(1)}
cheapfn <- function(x) ZDT3(x)[2]
response <- apply(design, 1, funnoise4)
design.noise.var <- rep(noise.var, n.init)
model <- list(km(.-, design = design, response = response, noise.var = design.noise.var))

omEGO4 <- GPar所得税(model = model, fn = funnoise4, cheapfn = cheapfn, crit = "SUR",
                        nsteps = nsteps, lower = lower, upper = upper, critcontrol = critcontrol,
                        reinterpolation = TRUE, noise.var = noise.var, optimcontrol = optimcontrol)
plotGPareto(omEGO4)

# Test 5: EMI, fastfun, noise.var given by fn
funnoise5 <- function(x) {
  if (is.null(dim(x))) x <- matrix(x, nrow = 1)
  list(apply(x, 1, ZDT3)[1,] + sqrt(abs(0.05*x[,1]))*rnorm(nrow(x)), abs(0.05*x[,1]))
}
cheapfn <- function(x) {
  if (is.null(dim(x))) x <- matrix(x, nrow = 1)
  apply(x, 1, ZDT3)[2,]
}

temp <- funnoise5(design)
response <- temp[1,]
design.noise.var <- temp[2,]
model <- list(km(.-, design = design, response = response, noise.var = design.noise.var))

omEGO5 <- GPar所得税(model = model, fn = funnoise5, cheapfn = cheapfn, crit = "EMI",
                        nsteps = nsteps, lower = lower, upper = upper, critcontrol = critcontrol,
                        reinterpolation = TRUE, noise.var = "given_by_fn", optimcontrol = optimcontrol)
plotGPareto(omEGO5)

# Test 6: EHI, fastfun, functional noise.var
noise.var <- 0.1
funnoise6 <- function(x) (ZDT3(x)[1] + sqrt(abs(0.1*x[1]))*rnorm(1))
noise.var <- function(x) return(abs(0.1*x[1]))
cheapfn <- function(x) ZDT3(x)[2]
response <- apply(design, 1, funnoise6)
design.noise.var <- t(apply(design, 1, noise.var))
model <- list(km(.-, design = design, response = response, noise.var = design.noise.var))

omEGO6 <- GPar所得税(model = model, fn = funnoise6, cheapfn = cheapfn, crit = "EMI",
                        nsteps = nsteps, lower = lower, upper = upper, critcontrol = critcontrol,
                        reinterpolation = TRUE, noise.var = noise.var, optimcontrol = optimcontrol)
plotGPareto(omEGO6)

## End(Not run)
Description

Modification of the function `integration_design` from the package `KrigInv` to be usable for SUR-based optimization. Handles two or three objectives. Available important sampling schemes: none so far.

Usage

```
integration_design_optim(SURcontrol = NULL, d = NULL, lower, upper,
model = NULL, min.prob = 0.001)
```

Arguments

- **SURcontrol**: Optional list specifying the procedure to build the integration points and weights. Many options are possible; see 'Details'.
- **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 list of kriging models of `km` class.
- **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 probabilities equal to zero and importance sampling weights equal to infinity. In an importance sample of `M` points, the maximum weight becomes `1/min.prob * 1/M`.

Details

The `SURcontrol` argument is a list with possible entries `integration.points`, `integration.weights`, `n.points`, `n.candidates`, `distrib`, `init.distrib` and `init.distrib.spec`. It can be used in one of the following ways:

- A) If nothing is specified, `100 * d` points are chosen using the Sobol sequence;
- B) One can directly set the field `integration.points` (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" and "SUR" (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 choice "SUR" corresponds to importance sampling distributions (unequal weights). 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.

Value

A list with components:

- `integration.points` p x d matrix of p points used for the numerical calculation of integrals
- `integration.weights` a vector of size p corresponding to the weight of each point. If all the points are equally weighted, `integration.weights` is set to NULL

References


See Also

`GParetoptim crit_SUR integration_design`

nonDomSet

**Non-dominated points with respect to a reference**

**Description**

Determines which elements in a set are dominated by reference points

**Usage**

`nonDomSet(points, ref)`

**Arguments**

- `points` matrix (one point per row) that are compared to a reference `ref` (i.e., not between themselves)
- `ref` matrix (one point per row) of reference (faster if they are already Pareto optimal)
Examples

```r
## Not run:
d <- 6
n <- 1000
n2 <- 1000

test <- matrix(runif(d * n), n)
ref <- matrix(runif(d * n), n)
indPF <- nonDomInd(ref)

system.time(res <- nonDomSet(test, ref[indPF,,drop = F]))

res2 <- rep(NA, n2)
library(eemoa)
t0 <- Sys.time()
for(i in 1:n2){
  res2[i] <- !is_dominated(t(rbind(test[i,, drop = F], ref[indPF,,])))[1]
}
print(Sys.time() - t0)

all(res == res2)
## End(Not run)
```

---

**ParetoSetDensity**

**Estimation of Pareto set density**

### Description

Estimation of the density of Pareto optimal points in the variable space.

### Usage

```r
ParetoSetDensity(model, lower, upper, CPS = NULL, nsim = 50,
                  simpoints = 1000, ...)
```

### Arguments

- **model**: list of objects of class `km`, one for each objective functions,
- **lower**: vector of lower bounds for the variables,
- **upper**: vector of upper bounds for the variables,
- **CPS**: optional matrix containing points from Conditional Pareto Set Simulations (in the variable space), see details
- **nsim**: optional number of conditional simulations to perform if CPS is not provided,
- **simpoints**: (optional) If CPS is NULL, either a number of simulation points, or a matrix where conditional simulations are to be performed. In the first case, then simulation points are taken as a maximin LHS design using `lhsDesign`.  

```r
# Not run:
d <- 6
n <- 1000
n2 <- 1000

test <- matrix(runif(d * n), n)
ref <- matrix(runif(d * n), n)
indPF <- nonDomInd(ref)

system.time(res <- nonDomSet(test, ref[indPF,,drop = F]))

res2 <- rep(NA, n2)
library(eemoa)
t0 <- Sys.time()
for(i in 1:n2){
  res2[i] <- !is_dominated(t(rbind(test[i,, drop = F], ref[indPF,,])))[1]
}
print(Sys.time() - t0)

all(res == res2)
## End(Not run)
```
... further arguments to be passed to `kde`. In particular, if the input dimension is greater than three, a matrix `eval.points` can be given (else it is taken as the simulation points).

Details

This function estimates the density of Pareto optimal points in the variable space given by the surrogate models. Based on conditional simulations of the objectives at simulation points, Conditional Pareto Set (CPS) simulations are obtained, out of which a density is fitted.

This function relies on the `ks` package for the kernel density estimation.

Value

An object of class `kde` accounting for the estimated density of Pareto optimal points.

Examples

```r
## Not run:
#-----------------------------------------------------------------------------------
# Example of estimation of the density of Pareto optimal points
#-----------------------------------------------------------------------------------
set.seed(42)
n_var <- 2
fname <- P1
lower <- rep(0, n_var)
upper <- rep(1, n_var)

res1 <- easyG Paretooptim(fn = fname, lower = lower, upper = upper, budget = 15,
control=list(method = "EHI", inneroptim = "pso", maxit = 20))
estDens <- ParetoSetDensity(res1$model, lower = lower, upper = upper)

# graphics
par(mfrow = c(1,2))
plot(estDens, display = "persp", xlab = "X1", ylab = "X2")
plot(estDens, display = "filled.contour", main = "Estimated density of Pareto optimal point")
points(res1$model[,1], res1$model[,2], col="blue")
points(estDens$x[, 1], estDens$x[, 2], pch = 20, col = rgb(0, 0, 0, 0.15))
par(mfrow = c(1,1))
## End(Not run)
```

plotGPareto

---

**Plot multi-objective optimization results and post-processing**

Description

Display results of multi-objective optimization returned by either `GParetooptim` or `easyGParetoptim`, possibly completed with various post-processings of uncertainty quantification.
Usage

plotGPareto(res, add = FALSE, UQ_PF = FALSE, UQ_PS = FALSE,
            UQ_dens = FALSE, lower = NULL, upper = NULL, control = list(pch =
            20, col = "red", PF.line.col = "cyan", PF.pch = 17, PF.points.col =
            "blue", VE.line.col = "cyan", nsim = 100, npsim = 1500, gridtype =
            "runif", displaytype = "persp", printVD = TRUE, use.rgl = TRUE, bounds =
            NULL, meshsize3d = 50, theta = -25, phi = 10, add_denoised_PF = TRUE))

Arguments

res
  list returned by GParetoptim or easyGParetoptim,
add
  logical; if TRUE adds the first graphical output to an already existing plot; if
  FALSE, (default) starts a new plot,
UQ_PF
  logical; for 2 objectives, if TRUE perform a quantification of uncertainty on the
  Pareto front to display the symmetric deviation function with plotSymDevFun
  (cannot be added to existing graph),
UQ_PS
  logical; if TRUE call plot_uncertainty representing the probability of non-
  domination in the variable space,
UQ_dens
  logical; for 2D problems, if TRUE call ParetoSetDensity to estimate and dis-
  play the density of Pareto optimal points in the variable space,
lower
  optional vector of lower bounds for the variables. Necessary if UQ_PF and/or
  UQ_PS are TRUE (if not provided, variables are supposed to vary between 0 and
  1),
upper
  optional vector of upper bounds for the variables. Necessary if UQ_PF and/or
  UQ_PS are TRUE (if not provided, variables are supposed to vary between 0 and
  1),
control
  optional list, see details.

Details

By default, plotGPareto displays the Pareto front delimiting the non-dominated area with 2 ob-
jectives, by a perspective view with 3 objectives and using parallel coordinates with more objectives.

Setting one or several of UQ_PF, UQ_PS and UQ_dens allows to run and display post-processing
tools that assess the precision and confidence of the optimization run, either in the objective (UQ_PF)
or the variable spaces (UQ_PS, UQ_dens). Note that these options are computationally intensive.

Various parameters can be used for the display of results and/or passed to subsequent function:

- col, pch correspond the color and plotting character for observations,
- PF.line.col, PF.pch, PF.points.col define the color of the line denoting the current Pareto
  front, the plotting character and color of non-dominated observations, respectively,
- nsim, npsim and gridtype define the number of conditional simulations performed with
  simulate along with the number of simulation points (in case UQ_PF and/or UQ_dens are
  TRUE),
gridtype to define how simulation points are selected; alternatives are 'runif' (default) for uniformly sampled points, 'LHS' for a Latin Hypercube design using \texttt{lhsDesign} and 'grid2d' for a two dimensional grid,

- \texttt{f1lim}, \texttt{f2lim} can be passed to \texttt{CPF}.
- resolution, option, nintegpoints are to be passed to \texttt{plot_uncertainty}
- displaytype type of display for \texttt{UQ_dens}, see \texttt{plot.kde},
- printVD logical, if \texttt{TRUE} and \texttt{UQ_PF} is \texttt{TRUE} as well, print the value of the Vorob’ev deviation,
- use.\texttt{rgl} if \texttt{TRUE}, use \texttt{rgl} for 3D plots, else \texttt{persp} is used,
- bounds if use.\texttt{rgl} is \texttt{TRUE}, optional 2*nobj matrix of boundaries, see \texttt{plotParetoEmp}
- meshsize3d mesh size of the perspective view for 3-objective problems,
- theta, phi angles for perspective view of 3-objective problems,
- add_denoised_PF if \texttt{TRUE}, in the noisy case, add the Pareto front from the estimated mean of the observations.

References


A. Inselberg (2009), \textit{Parallel coordinates}, Springer.

Examples

```r
## Not run:
#---------------------------------------------------------------
# 2D objective function
#---------------------------------------------------------------
set.seed(25468)
n_var <- 2
fname <- P1
lower <- rep(0, n_var)
upper <- rep(1, n_var)
res <- easyGParetoptim(family=fname, lower=lower, upper=upper, budget=15,
control=list(method="EH", inneroptim="pso", maxit=20))

## Pareto front only
plotGPareto(res)

## With post-processing
plotGPareto(res, UQ_PF = TRUE, UQ_PS = TRUE, UQ_dens = TRUE)

## With noise
noise.var <- c(10, 2)
funnoise <- function(x) {P1(x) + sqrt(noise.var)*rnorm(n2)}
res2 <- easyGParetoptim(family=funnoise, lower=lower, upper=upper, budget=15, noise.var=noise.var,
control=list(method="EH", inneroptim="pso", maxit=20))
```
plotGPareto(res2, control=list(add_denoised_PF=FALSE))  # noisy observations only
plotGPareto(res2)

# 3D objective function
set.seed(1)
n_var <- 3
fname <- DTLZ1
lower <- rep(0, n_var)
upper <- rep(1, n_var)
res3 <- easyGParetoptim(fn=fname, lower=lower, upper=upper, budget=50,
control=list(method="EHI", inneroptim="pso", maxit=20))

## Pareto front only
plotGPareto(res3)

## With noise
noise.var <- c(10, 2, 5)
funnoise <- function(x) {fname(x) + sqrt(noise.var)*rnorm(n=3)}
res4 <- easyGParetoptim(fn=funnoise, lower=lower, upper=upper, budget=100, noise.var=noise.var,
control=list(method="EHI", inneroptim="pso", maxit=20))

plotGPareto(res4, control=list(add_denoised_PF=FALSE))  # noisy observations only
plotGPareto(res4)

## End(Not run)

---

**plotParetoEmp**

**Pareto front visualization**

### Description
Plot the Pareto front with step functions.

### Usage

```r
plotParetoEmp(nondominatedPoints, add = TRUE, max = FALSE,
              bounds = NULL, alpha = 0.5, ...)
```

### Arguments

- **nondominatedPoints**: points considered to plot the Pareto front with segments, matrix with one point per row,
- **add**: optional boolean indicating whether a new graphic should be drawn,
- **max**: optional boolean indicating whether to display a Pareto front in a maximization context,
- **bounds**: for 3D, optional 2*nobj matrix of boundaries
plotParetoGrid

alpha for 3D, optional value in [0,1] for transparency

... additional values to be passed to the lines function.

Examples

#------------------------------------------------------------
# Simple example
#------------------------------------------------------------

x <- c(0.2, 0.4, 0.6, 0.8)
y <- c(0.8, 0.7, 0.5, 0.1)

plot(x, y, col = "green", pch = 20)
plotParetoEmp(cbind(x, y), col = "green")
## Alternative
plotParetoEmp(cbind(x, y), col = "red", add = FALSE)

## With maximization
plotParetoEmp(cbind(x, y), col = "blue", max = TRUE)

## 3D plots
library(rgl)
set.seed(5)
X <- matrix(runif(60), ncol=3)
Xnd <- t(nondominated_points(t(X)))
plot3d(X)
plot3d(Xnd, col="red", size=8, add=TRUE)
plot3d(x=min(Xnd[,1]), y=min(Xnd[,2]), z=min(Xnd[,3]), col="green", size=8, add=TRUE)
X.range <- diff(apply(X,2,range))
bounds <- rbind(apply(X,2,min)-0.1*X.range,apply(X,2,max)+0.1*X.range)
plotParetoEmp(nondominatedPoints = Xnd, add=TRUE, bounds=bounds, alpha=0.5)

plotParetoGrid 

Visualisation of Pareto front and set

Description

Plot the Pareto front and set for 2 variables 2 objectives test problems with evaluations on a grid.

Usage

plotParetoGrid(fname = "ZDT1", xlim = c(0, 1), ylim = c(0, 1),
n.grid = 100)
plotSymDevFun

Arguments

fname  
name of the function considered.
xlim, ylim  
numeric vectors of length 2, giving the x and y coordinates ranges, default is $[0,1] \times [0,1]$.
n.grid  
number of divisions of the grid in each dimension.

Examples

#-------------------------------------------------------------
# Examples with test functions
#-------------------------------------------------------------

plotParetoGrid("ZDT3", n.grid = 21)
plotParetoGrid("P1", n.grid = 21)
plotParetoGrid("MOP2", xlim = c(0, 1), ylim = c(0, 1), n.grid = 21)

plotSymDevFun  
Display the Symmetric Deviation Function

Description

Display the Symmetric Deviation Function for an object of class CPF.

Usage

plotSymDevFun(CPF, n.grid = 100)

Arguments

CPF  
CPF object, see CPF.
n.grid  
number of divisions of the grid in each dimension.

Details

Display observations in red and the corresponding Pareto front by a step-line. The blue line is the estimation of the location of the Pareto front of the kriging models, named Vorob’ev expectation. In grayscale is the intensity of the deviation (symmetrical difference) from the Vorob’ev expectation for couples of conditional simulations.

References

Examples

```r
library(DiceDesign)
set.seed(42)

nvar <- 2

# Test function
fname = "P1"

# Initial design
napp <- 10
design.grid <- maximinESE_LHS(lhsDesign(napp, nvar, seed = 42)$design)$design
response.grid <- t(apply(design.grid, 1, fname))
ParetoFront <- t(nonDominated_points(t(response.grid)))

# kriging models: matern5_2 covariance structure, linear trend, no nugget effect
mf1 <- km(~., design = design.grid, response = response.grid[, 1])
mf2 <- km(~., design = design.grid, response = response.grid[, 2])

# Conditional simulations generation with random sampling points
nsim <- 10 # increase for better results
npointssim <- 80 # increase for better results
Simu_f1 = matrix(0, nrow = nsim, ncol = npointssim)
Simu_f2 = matrix(0, nrow = nsim, ncol = npointssim)
design.sim = array(0, dim = c(npointssim, nvar, nsim))

for(i in 1:nsim){
  design.sim[, , i] <- matrix(runif(nvar*npointssim), npointssim, nvar)
  Simu_f1[i, , ] = simulate(mf1, nsim = 1, newdata = design.sim[, , i], cond = TRUE,
                            checkNames = FALSE, nugget.sim = 10^-8)
  Simu_f2[i, , ] = simulate(mf2, nsim = 1, newdata = design.sim[, , i], cond=TRUE,
                            checkNames = FALSE, nugget.sim = 10^-8)
}

# Attainment, Voreb’eV expectation and deviation estimation
CPF1 <- CPF(Simu_f1, Simu_f2, response.grid, ParetoFront)

# Symmetric deviation function
plotSymDevFun(CPF1)
```

Description

Plot the symmetrical difference between two Random Non-Dominated Point (RNP) sets.

plotSymDifRNP  Symmetrical difference of RNP sets
Usage

plotSymDifRNP(set1, set2, xlim, ylim, fill = "black", add = "FALSE", ...)  

Arguments

set1, set2  
RNP sets considered,

xlim, ylim  
numeric vectors of length 2, giving the x and y coordinates ranges for plotting,

fill  
optional color of the symmetric difference area,

add  
logical; if TRUE add to an already existing plot; if FALSE (default) start a new plot taking xlim, ylim as limits.

additional parameters for the plot and polygon graphic functions

Examples

#---------------------------------------------------------------  
# Simple example  
#---------------------------------------------------------------  
set1 <- rbind(c(0.2, 0.35, 0.5, 0.8),  
              c(0.8, 0.6, 0.55, 0.3))

set2 <- rbind(c(0.3, 0.4),  
              c(0.7, 0.4))

plotSymDifRNP(set1, set2, xlim = c(0, 1), ylim = c(0, 1), fill = "grey")
points(t(set1), col = "red", pch = 20)
points(t(set2), col = "blue", pch = 20)

plot_uncertainty  
Plot uncertainty

Description

Displays the probability of non-domination in the variable space. In dimension larger than two, projections in 2D subspaces are displayed.

Usage

plot_uncertainty(model, paretofront = NULL, type = "pn", lower, upper,  
resolution = 51, option = "mean", nintegpoints = 400)
plot_uncertainty

Arguments

- **model**: list of objects of class km, one for each objective functions,
- **paretoFront**: (optional) matrix corresponding to the Pareto front of size \([n.pareto \times n.obj]\),
- **type**: type of uncertainty, for now only the probability of improvement over the Pareto front,
- **lower**: vector of lower bounds for the variables,
- **upper**: vector of upper bounds for the variables,
- **resolution**: grid size (the total number of points is \(\text{resolution}^d\)),
- **option**: optional argument (string) for \(n > 2\) variables to define the projection type. The 3 possible values are "mean" (default), "max" and "min",
- **nintegpoints**: number of integration points for computation of mean, max and min values.

Details

Function inspired by the function `print_uncertainty` and `print_uncertainty_nd` from the package KrigInv. Non-dominated observations are represented with green diamonds, dominated ones by yellow triangles.

Examples

```r
## Not run:
#-----------------------------------------------
# 2D, bi-objective function
#-----------------------------------------------
set.seed(25468)
n_var <- 2
fname <- P1
lower <- rep(0, n_var)
upper <- rep(1, n_var)
res1 <- easyGParetoptim(fn=fname, lower=lower, upper=upper, budget=15,
control=list(method="EHI", inneroptim="pso", maxit=20))
plot_uncertainty(res1$model, lower = lower, upper = upper)

#-----------------------------------------------
# 4D, bi-objective function
#-----------------------------------------------
set.seed(25468)
n_var <- 4
fname <- DTLZ2
lower <- rep(0, n_var)
upper <- rep(1, n_var)
res <- easyGParetoptim(fn=fname, lower=lower, upper=upper, budget = 40,
control=list(method="EHI", inneroptim="pso", maxit=40))
plot_uncertainty(res$model, lower = lower, upper = upper, resolution = 31)

## End(Not run)
```
Predict function for list of km models.

**Description**

Predict function for list of km models.

**Usage**

```r
predict_kms(model, newdata, type, se.compute = TRUE,
              cov.compute = FALSE, light.return = FALSE, bias.correct = FALSE,
              checkNames = TRUE, ...)
```

**Arguments**

- `model`: list of km models
- `newdata, type, se.compute, cov.compute, light.return, bias.correct, checkNames, ...`

**Details**

So far only `light.return = TRUE` and `cov.compute = FALSE` handled.

---

**ZDT1**

Test functions of x

**Description**

Multi-objective test functions.

**Usage**

- `ZDT1(x)`
- `ZDT2(x)`
- `ZDT3(x)`
- `ZDT4(x)`
- `ZDT6(x)`
- `P1(x)`
- `P2(x)`
MOP2(x)
MOP3(x)

DTLZ1(x, nobj = 3)
DTLZ2(x, nobj = 3)
DTLZ3(x, nobj = 3)
DTLZ7(x, nobj = 3)

Arguments

x matrix specifying the location where the function is to be evaluated, one point per row,
nobj optional argument to select the number of objective for the DTLZ test functions.

Details

These functions are coming from different benchmarks: the ZDT test problems from an article of E. Zitzler et al., P1 from the thesis of J. Parr and P2 from an article of Poloni et al. MOP2 and MOP3 are from Van Veldhuizen and DTLZ functions are from Deb et al.

Domains (sometimes rescaled to $[0, 1]$):

- ZDT1-6: $[0, 1]^d$
- P1, P2: $[0, 1]^2$
- MOP2: $[0, 1]^d$
- MOP3: $[-3, 3]$, tri-objective, 2 variables
- DTLZ1-3, 7: $[0, 1]^d$, m-objective problems, with at least $d > m$ variables.

Value

Matrix of values corresponding to the objective functions, the number of columns is the number of objectives.

References

J. M. Parr (2012), Improvement Criteria for Constraint Handling and Multiobjective Optimization, University of Southampton, PhD thesis.


**Examples**

```bash
# ----------------------------------------
# 2-objectives test problems
# ----------------------------------------

plotParetoGrid("ZDT1", n.grid = 21)
plotParetoGrid("ZDT2", n.grid = 21)
plotParetoGrid("ZDT3", n.grid = 21)
plotParetoGrid("ZDT4", n.grid = 21)
plotParetoGrid("ZDT6", n.grid = 21)
plotParetoGrid("P1", n.grid = 21)
plotParetoGrid("P2", n.grid = 21)
plotParetoGrid("MOP2", n.grid = 21)
```
Index

checkPredict, 6, 10, 12, 14, 19, 21, 31
CPF, 2, 7, 41, 44
crit_EHI, 9, 12, 14, 19, 21, 23, 31
crit_EMI, 10, 11, 14, 19, 21, 23, 31
crit_optimizer, 2, 13, 21
crit_SMS, 10, 12, 14, 18, 21, 23, 31
crit_SUR, 10, 12, 14, 19, 20, 23, 31, 37
DiceKriging, 3
DiceOptim, 3
DTLZ1 (ZDT1), 48
DTLZ2 (ZDT1), 48
DTLZ3 (ZDT1), 48
DTLZ7 (ZDT1), 48
easyGParetoptim, 2, 22, 39, 40
EGO.nsteps, 31
EI, 10, 12
fastfun, 13, 22, 26, 30
genoud, 13, 14, 23, 28, 30
gGetDesign, 28
GPareto (GPareto-package), 2
GPareto-package, 2
GParetoptim, 2, 21–23, 29, 37, 39, 40
integration_design, 36, 37
integration_design_optim, 14, 21, 31, 35
kde, 39
km, 6, 10, 11, 13, 18, 20, 22–24, 26, 28–31, 38, 47, 48
KrigInv, 36, 47
ks, 39
lhsDesign, 38, 41
lines, 43
match.fun, 14, 22, 26, 30
max_EI, 14
MOP2 (ZDT1), 48
MOP3 (ZDT1), 48
nonDomSet, 37
optim, 14, 30
P1 (ZDT1), 48
P2 (ZDT1), 48
ParetoSetDensity, 38, 40
persp, 41
plot, 46
plot.kde, 41
plot_uncertainty, 40, 41, 46
plotGPareto, 2, 23, 31, 39
plotParetoEmp, 41, 42
plotParetoGrid, 43
plotSymDevFun, 40, 44
plotSymDifRNP, 45
polygon, 46
predict.km, 48
predict_kms, 48
print_uncertainty, 47
print_uncertainty_nd, 47
psos, 13
psoptim, 14, 23, 28, 30
simulate, 40
ZDT1, 48
ZDT2 (ZDT1), 48
ZDT3 (ZDT1), 48
ZDT4 (ZDT1), 48
ZDT6 (ZDT1), 48