Package ‘DiceOptim’

September 15, 2016

Version 2.0

Title Kriging-Based Optimization for Computer Experiments

Date 2016-09-06

Author V. Picheny, D. Ginsbourger, O. Roustant, with contributions by
M. Binois, C. Chevalier, S. Marmin, and T. Wagner

Maintainer V. Picheny <victor.picheny@toulouse.inra.fr>

Description Efficient Global Optimization (EGO) algorithm and adaptations for
parallel infill (multipoint EI), problems with noise, and problems with
constraints.

Depends DiceKriging (>= 1.2), methods

Imports randtoolbox, pbivnorm, rgenoud, mnormt, DiceDesign

Suggests KrigInv, GPareto, lhs

License GPL-2 | GPL-3

URL http://dice.emse.fr/

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-09-15 17:33:46

R topics documented:

`DiceOptim-package` *Kriging-based optimization methods for computer experiments*

Description

Sequential and parallel Kriging-based optimization methods relying on expected improvement criteria.

Details

Package:	DiceOptim
Type:	Package
Version:	2.0
Date:	June 2016
License:	GPL-2 | GPL-3

Note

This work is a follow-up of DiceOptim 1.0, which was conducted within the frame of the DICE (Deep Inside Computer Experiments) Consortium between ARMINES, Renault, EDF, IRSN, ONERA and TOTAL S.A. (http://dice.emse.fr/). Some of the more recent contributions were added in the framework of the ReDICE consortium, gathering industrial (CEA, EDF, IFPEN, IRSN, Renault) and academic (Mines Saint-Etienne, INRIA, and the University of Bern) partners around advanced methods for computer experiments.

The authors would like to thank Yves Deville for his precious advice in R programming and packaging, as well as the DICE and ReDICE members for useful feedbacks, and especially Yann Richet (IRSN) for numerous discussions concerning the user-friendliness of this package.

Package `rgenoud` >=5.3.3. is recommended.

Important functions or methods:

`EGO.nsteps`	Standard Efficient Global Optimization algorithm with a fixed number of iterations (nsteps) —with model updates including re-estimation of covariance hyperparameters
`EI`	Expected Improvement criterion (single infill point, noise-free, constraint free problems)
`max_EI`	Maximization of the EI criterion. No need to specify any objective function
`qEI.nsteps`	EGO algorithm with batch-sequential (parallel) infill strategy
`noisy.optimizer`	EGO algorithm for noisy objective functions
`EGO.cst`	EGO algorithm for (non-linear) constrained problems
`easyEGO.cst`	User-friendly wrapper for `EGO.cst`

Author(s)

Victor Picheny (INRA, Castanet-Tolosan, France)

David Ginsbourger (Idiap Research Institute and University of Bern, Switzerland)

Olivier Roustant (Mines Saint-Etienne, France).

with contributions by M. Binois, C. Chevalier, S. Marmin and T. Wagner

References

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

D. Ginsbourger (2009), *Multiples metamodeles pour l'approximation et l'optimisation de fonctions numeriques multivariables*, Ph.D. thesis, Ecole Nationale Superieure des Mines de Saint-Etienne, 2009. https://tel.archives-ouvertes.fr/tel-00772384

D. Ginsbourger, R. Le Riche, and L. Carraro (2010), chapter "Kriging is well-suited to parallelize optimization", in *Computational Intelligence in Expensive Optimization Problems*, Studies in Evolutionary Learning and Optimization, Springer.

D.R. Jones (2001), A taxonomy of global optimization methods based on response surfaces, *Journal of Global Optimization*, 21, 345-383.

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

W.R. Jr. Mebane and J.S. Sekhon (2009), in press, Genetic optimization using derivatives: The rgenoud package for R, *Journal of Statistical Software*.

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

V. Picheny and D. Ginsbourger (2013), Noisy kriging-based optimization methods: A unified implementation within the DiceOptim package, *Computational Statistics & Data Analysis*

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

B.D. Ripley (1987), *Stochastic Simulation*, Wiley.

O. Roustant, D. Ginsbourger and Yves Deville (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, http://www.jstatsoft.org/v51/i01/.

T.J. Santner, B.J. Williams, and W.J. Notz (2003), *The design and analysis of computer experiments*, Springer.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

Examples

```
set.seed(123)

###############################################################
### 2D optimization USING EGO.nsteps and qEGO.nsteps   ########
###############################################################

# a 9-points factorial design, and the corresponding response
d <- 2
n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.branin <- data.frame(apply(design.fact, 1, branin))
names(response.branin) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.branin,
covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

### EGO, 5 steps ##################
library(rgenoud)
nsteps <- 5
lower <- rep(0,d)
upper <- rep(1,d)
oEGO <- EGO.nsteps(model=fitted.model1, fun=branin, nsteps=nsteps,
lower=lower, upper=upper, control=list(pop.size=20, BFGSburnin=2))
print(oEGO$par)
print(oEGO$value)

# graphics
n.grid <- 15
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
```

```
response.grid <- apply(design.grid, 1, branin)
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid, y.grid, z.grid, 40)
title("EGO")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par, pch=19, col="red")
text(oEGO$par[,1], oEGO$par[,2], labels=1:nsteps, pos=3)

### Parallel EGO, 3 steps with batches of 3 ##############
nsteps <- 3
lower <- rep(0,d)
upper <- rep(1,d)
npoints <- 3 # The batchsize
oEGO <- qEGO.nsteps(model = fitted.model1, branin, npoints = npoints, nsteps = nsteps,
crit="exact", lower, upper, optimcontrol = NULL)
print(oEGO$par)
print(oEGO$value)

# graphics
contour(x.grid, y.grid, z.grid, 40)
title("qEGO")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par, pch=19, col="red")
text(oEGO$par[,1], oEGO$par[,2], labels=c(tcrossprod(rep(1,npoints),1:nsteps)), pos=3)

###########################################################################
### 2D OPTIMIZATION, NOISY OBJECTIVE                                    ###
###########################################################################

set.seed(10)
library(DiceDesign)
# Set test problem parameters
doe.size <- 9
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.1

# Build noisy simulator
funnoise <- function(x)
{     f.new <- test.function(x) + sqrt(noise.var)*rnorm(n=1)
      return(f.new)}

# Generate DOE and response
doe <- as.data.frame(lhsDesign(doe.size, dim)$design)
y.tilde <- funnoise(doe)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
     covtype="gauss", noise.var=rep(noise.var,1,doe.size),
     lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))
```

```
# Optimisation with noisy.optimizer
optim.param <- list()
optim.param$quantile <- .7
optim.result <- noisy.optimizer(optim.crit="EQI", optim.param=optim.param, model=model,
n.ite=5, noise.var=noise.var, funnoise=funnoise, lower=lower, upper=upper,
NoiseReEstimate=FALSE, CovReEstimate=FALSE)

print(optim.result$best.x)

##########################################################################
### 2D OPTIMIZATION, 2 INEQUALITY CONSTRAINTS                          ###
##########################################################################
set.seed(25468)
library(DiceDesign)

fun <- goldsteinprice
fun1.cst <- function(x){return(-branin(x) + 25)}
fun2.cst <- function(x){return(3/2 - x[1] - 2*x[2] - .5*sin(2*pi*(x[1]^2 - 2*x[2])))}
constraint <- function(x){return(c(fun1.cst(x), fun2.cst(x)))}

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

## Optimization using the Expected Feasible Improvement criterion
res <- easyEGO.cst(fun=fun, constraint=constraint, n.cst=2, lower=lower, upper=upper, budget=10,
                   control=list(method="EFI", inneroptim="genoud", maxit=20))

cat("best design found:", res$par, "\n")
cat("corresponding objective and constraints:", res$value, "\n")

# Objective function in colour, constraint boundaries in red
# Initial DoE: white circles, added points: blue crosses, best solution: red cross

n.grid <- 15
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun)
cst1.grid <- apply(test.grid, 1, fun1.cst)
cst2.grid <- apply(test.grid, 1, fun2.cst)
filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(obj.grid, n.grid), main = "Two inequality constraints",
               xlab = expression(x[1]), ylab = expression(x[2]), color = terrain.colors,
               plot.axes = {axis(1); axis(2);
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                                     matrix(cst1.grid, n.grid), level = 0, add=TRUE,
                                     drawlabels=FALSE, lwd=1.5, col = "red")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                              matrix(cst2.grid, n.grid), level = 0, add=TRUE,drawlabels=FALSE,
                                     lwd=1.5, col = "red")
                             points(res$history$X, col = "blue", pch = 4, lwd = 2)
                       points(res$par[1], res$par[2], col = "red", pch = 4, lwd = 2, cex=2)
               }
)
```

AEI *Augmented Expected Improvement*

Description

Evaluation of the Augmented Expected Improvement (AEI) criterion, which is a modification of the classical EI criterion for noisy functions. The AEI consists of the regular EI multiplied by a penalization function that accounts for the disminishing payoff of observation replicates. The current minimum y.min is chosen as the kriging predictor of the observation with smallest kriging quantile.

Usage

```
AEI(x, model, new.noise.var = 0, y.min = NULL, type = "UK",
  envir = NULL)
```

Arguments

x	the input vector at which one wants to evaluate the criterion
model	a Kriging model of "km" class
new.noise.var	the (scalar) noise variance of the future observation.
y.min	The kriging predictor at the current best point (point with smallest kriging quantile). If not provided, this quantity is evaluated.
type	Kriging type: "SK" or "UK"
envir	environment for saving intermediate calculations and reusing them within AEI.grad

Value

Augmented Expected Improvement

Author(s)

Victor Picheny

David Ginsbourger

References

D. Huang, T.T. Allen, W.I. Notz, and N. Zeng (2006), Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models, *Journal of Global Optimization*, 34, 441-466.

Examples

```
########################################################################
###    AEI SURFACE ASSOCIATED WITH AN ORDINARY KRIGING MODEL        ####
### OF THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
########################################################################

set.seed(421)

# Set test problem parameters
doe.size <- 12
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
            covtype="gauss", noise.var=rep(noise.var,1,doe.size),
    lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)
crit.grid <- rep(0,1,nt)
func.grid <- rep(0,1,nt)

crit.grid <- apply(design.grid, 1, AEI, model=model, new.noise.var=noise.var)
func.grid <- apply(design.grid, 1, test.function)

# Compute kriging mean and variance on a grid
names(design.grid) <- c("V1","V2")
pred <- predict.km(model, newdata=design.grid, type="UK")
mk.grid <- pred$m
sk.grid <- pred$sd

# Plot actual function
z.grid <- matrix(func.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Actual function");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})
```

```
# Plot Kriging mean
z.grid <- matrix(mk.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Kriging mean");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot Kriging variance
z.grid <- matrix(sk.grid^2, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Kriging variance");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot AEI criterion
z.grid <- matrix(crit.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("AEI");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})
```

`AEI.grad` *AEI's Gradient*

Description

Analytical gradient of the Augmented Expected Improvement (AEI) criterion.

Usage

```
AEI.grad(x, model, new.noise.var = 0, y.min = NULL, type = "UK",
  envir = NULL)
```

Arguments

`x`	the input vector at which one wants to evaluate the criterion
`model`	a Kriging model of "km" class
`new.noise.var`	the (scalar) noise variance of the new observation.
`y.min`	The kriging predictor at the current best point (point with smallest kriging quantile). If not provided, this quantity is evaluated.
`type`	Kriging type: "SK" or "UK"
`envir`	environment for inheriting intermediate calculations from AEI

Value

Gradient of the Augmented Expected Improvement

Author(s)

Victor Picheny

David Ginsbourger

Examples

```
## Not run:
set.seed(421)

# Set test problem parameters
doe.size <- 12
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
        covtype="gauss", noise.var=rep(noise.var,1,doe.size),
lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)

crit.grid <- apply(design.grid, 1, AEI, model=model, new.noise.var=noise.var)
crit.grad <- t(apply(design.grid, 1, AEI.grad, model=model, new.noise.var=noise.var))

z.grid <- matrix(crit.grid, n.grid, n.grid)
contour(x.grid,y.grid, z.grid, 30)
title("AEI and its gradient")
points(model@X[,1],model@X[,2],pch=17,col="blue")

options(warn=-1)
for (i in 1:nt)
{
 x <- design.grid[i,]
 arrows(x$Var1,x$Var2, x$Var1+crit.grad[i,1]*.6,x$Var2+crit.grad[i,2]*.6,
length=0.04,code=2,col="orange",lwd=2)
}
```

```
## End(Not run)
```

AKG *Approximate Knowledge Gradient (AKG)*

Description

Evaluation of the Approximate Knowledge Gradient (AKG) criterion.

Usage

```
AKG(x, model, new.noise.var = 0, type = "UK", envir = NULL)
```

Arguments

x	the input vector at which one wants to evaluate the criterion
model	a Kriging model of "km" class
new.noise.var	(scalar) noise variance of the future observation. Default value is 0 (noise-free observation).
type	Kriging type: "SK" or "UK"
envir	environment for saving intermediate calculations and reusing them within AKG.grad

Value

Approximate Knowledge Gradient

Author(s)

Victor Picheny

David Ginsbourger

References

Scott, W., Frazier, P., Powell, W. (2011). The correlated knowledge gradient for simulation optimization of continuous parameters using gaussian process regression. *SIAM Journal on Optimization*, 21(3), 996-1026.

Examples

```
############################################################################
###    AKG SURFACE ASSOCIATED WITH AN ORDINARY KRIGING MODEL          ####
### OF THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
############################################################################
set.seed(421)
# Set test problem parameters
doe.size <- 12
dim <- 2
```

```
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
  y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
            covtype="gauss", noise.var=rep(noise.var,1,doe.size),
            lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)

crit.grid <- apply(design.grid, 1, AKG, model=model, new.noise.var=noise.var)
func.grid <- apply(design.grid, 1, test.function)

# Compute kriging mean and variance on a grid
names(design.grid) <- c("V1","V2")
pred <- predict.km(model, newdata=design.grid, type="UK")
mk.grid <- pred$m
sk.grid <- pred$sd

# Plot actual function
z.grid <- matrix(func.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
               plot.axes = {title("Actual function");
                            points(model@X[,1],model@X[,2],pch=17,col="blue");
                            axis(1); axis(2)})

# Plot Kriging mean
z.grid <- matrix(mk.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
               plot.axes = {title("Kriging mean");
                            points(model@X[,1],model@X[,2],pch=17,col="blue");
                            axis(1); axis(2)})

# Plot Kriging variance
z.grid <- matrix(sk.grid^2, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
               plot.axes = {title("Kriging variance");
                            points(model@X[,1],model@X[,2],pch=17,col="blue");
                            axis(1); axis(2)})
```

```
# Plot AKG criterion
z.grid <- matrix(crit.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
               plot.axes = {title("AKG");
                            points(model@X[,1],model@X[,2],pch=17,col="blue");
                            axis(1); axis(2)})
```

AKG.grad *AKG's Gradient*

Description

Gradient of the Approximate Knowledge Gradient (AKG) criterion.

Usage

```
AKG.grad(x, model, new.noise.var = 0, type = "UK", envir = NULL)
```

Arguments

x	the input vector at which one wants to evaluate the criterion
model	a Kriging model of "km" class
new.noise.var	(scalar) noise variance of the future observation. Default value is 0 (noise-free observation).
type	Kriging type: "SK" or "UK"
envir	optional: environment for reusing intermediate calculations from AKG

Value

Gradient of the Approximate Knowledge Gradient

Author(s)

Victor Picheny

Examples

```
##########################################################################
###    AKG SURFACE AND ITS GRADIENT ASSOCIATED WITH AN ORDINARY       ####
###                                            KRIGING MODEL          ####
### OF THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
##########################################################################
## Not run:
set.seed(421)

# Set test problem parameters
doe.size <- 12
```

```
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
        covtype="gauss", noise.var=rep(noise.var,1,doe.size),
lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)

crit.grid <- apply(design.grid, 1, AKG, model=model, new.noise.var=noise.var)
crit.grad <- t(apply(design.grid, 1, AKG.grad, model=model, new.noise.var=noise.var))

z.grid <- matrix(crit.grid, n.grid, n.grid)
contour(x.grid,y.grid, z.grid, 30)
title("AKG and its gradient")
points(model@X[,1],model@X[,2],pch=17,col="blue")
options(warn=-1)
for (i in 1:nt)
{
 x <- design.grid[i,]
 arrows(x$Var1,x$Var2, x$Var1+crit.grad[i,1]*.2,x$Var2+crit.grad[i,2]*.2,
length=0.04,code=2,col="orange",lwd=2)
}

## 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

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

Arguments

x	a vector representing the input to check,
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", "covdist" (default) 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 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.

Value

TRUE if the point should not be tested.

Author(s)

Mickael Binois

---

critcst_optimizer *Maximization of constrained Expected Improvement criteria*

Description

Given objects of class km for the objective and constraints, and a set of tuning parameters (lower, upper and critcontrol), critcst_optimizer performs the maximization of a constrained Expected Improvement or SUR criterion and delivers the next point to be visited in an EGO-like procedure.

The latter maximization relies either on a genetic algorithm using derivatives, genoud or exhaustive search at pre-specified points. It is important to remark that the information needed about the objective and constraint functions reduces here to the vector of response values embedded in the models (no call to the objective/constraint functions or simulators (except possibly for the objective)).

Usage

```
critcst_optimizer(crit = "EFI", model.fun, model.constraint,
  equality = FALSE, lower, upper, type = "UK", critcontrol = NULL,
  optimcontrol = NULL)
```

Arguments

crit	sampling criterion. Three choices are available : "EFI", "AL" and "SUR",
model.fun	object of class km corresponding to the objective function, or, if the objective function is fast-to-evaluate, either the objective function to be minimized or a fastfun object, see details and examples below,
model.constraint	either one or a list of models of class km, one for each constraint,
equality	either FALSE if all constraints are inequalities, or a Boolean vector indicating which are equalities,
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" (default), depending whether uncertainty related to trend estimation has to be taken into account.
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 occuring 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" and "genoud". For each method, further parameters can be set. For "discrete", one has to provide the argument "candidate.points". For "genoud", one can control, among others, "pop.size" (default : [N = 3*2^dim for dim < 6 and N = 32*dim otherwise]), "max.generations" (12), "wait.generations" (2)), see genoud. Numbers into brackets are the default values.

Details

Extension of the function max_EI for constrained optimization.

Available infill criteria with crit are :

- Expected Probability of Feasibily (EFI) crit_EFI,
- Augmented Lagrangian (AL) crit_AL,
- Stepwise Uncertainty Reduction of the excursion volume (SUR) crit_SUR_cst.

Depending on the selected criterion, parameters can be given with critcontrol. Also options for checkPredict are available. More precisions are given in the corresponding help pages.

If the objective function to minimize is inexpensive, i.e. no need of a kriging model, then one can provide it in model.obj, which is handled next with class fastfun (or directly as a fastfun object). See example below.

In the case of equality constraints, it is possible to define them with equality. Additionally, one can modify the tolerance on the constraints using the tolConstraints component of critcontrol: an optional vector giving a tolerance for each of the constraints (equality or inequality). It is highly recommended to use it when there are equality constraints since the default tolerance of 0.05 (resp. 0 for inequality constraints) in such case might not be suited.

Value

A list with components:

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

Author(s)

Victor Picheny

Mickael Binois

References

W.R. Jr. Mebane and J.S. Sekhon (2011), Genetic optimization using derivatives: The rgenoud package for R, *Journal of Statistical Software*, 42(11), 1-26

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

See Also

critcst_optimizer, crit_EFI, crit_AL, crit_SUR_cst

Examples

```
#---------------------------------------------------------------------------
# 2D objective function, 2 cases
#---------------------------------------------------------------------------

set.seed(2546)
library(DiceDesign)

n_var <- 2
fun <- branin

fun1.cst <- function(x){return(goldsteinprice(x)+.5)}
fun2.cst <- function(x){return(3/2 - x[1] - 2*x[2] - .5*sin(2*pi*(x[1]^2 - 2*x[2])))}

# Constraint function with vectorial output
```

```
cstfun <- function(x){return(c(fun1.cst(x), fun2.cst(x)))}

n.grid <- 31
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun)
cst1.grid <- apply(test.grid, 1, fun1.cst)
cst2.grid <- apply(test.grid, 1, fun2.cst)

n_appr <- 12
design.grid <- round(maximinESE_LHS(lhsDesign(n_appr, n_var, seed = 2)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun)
cst1.init <- apply(design.grid, 1, fun1.cst)
cst2.init <- apply(design.grid, 1, fun2.cst)
model.fun <- km(~., design = design.grid, response = obj.init)
model.constraint1 <- km(~., design = design.grid, response = cst1.init, lower=c(.2,.2))
model.constraint2 <- km(~., design = design.grid, response = cst2.init, lower=c(.2,.2))
models.cst <- list(model.constraint1, model.constraint2)

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

#---------------------------------------------------------------------------
# Augmented Lagrangian Improvement, fast objective function, two ineq constraints,
# optimization with genoud
#---------------------------------------------------------------------------
critcontrol <- list(lambda=c(.5,2), rho=.5)
optimcontrol <- list(method = "genoud", max.generations=10, pop.size=20)

AL_grid <- apply(test.grid, 1, crit_AL, model.fun = fastfun(fun, design.grid),
                 model.constraint = models.cst, critcontrol=critcontrol)

cstEGO1 <- critcst_optimizer(crit = "AL", model.fun = fun,
                             model.constraint = models.cst, equality = FALSE,
                             lower = lower, upper = upper,
                             optimcontrol = optimcontrol, critcontrol=critcontrol)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(AL_grid, n.grid), main = "AL map and its maximizer (blue)",
               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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                        matrix(obj.grid, n.grid), nlevels = 10, add=TRUE,drawlabels=TRUE,
                                   col = "black")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                         matrix(cst1.grid, n.grid), level = 0, add=TRUE,drawlabels=FALSE,
                                   lwd=1.5, col = "red")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                         matrix(cst2.grid, n.grid), level = 0, add=TRUE,drawlabels=FALSE,
                                   lwd=1.5, col = "red")
                            points(cstEGO1$par, col = "blue", pch = 4, lwd = 2)
                            }
              )
```

```
#---------------------------------------------------------------------------
# SUR, expensive objective function, one equality constraint,
# optimization with genoud, integration on a regular grid
#---------------------------------------------------------------------------
optimcontrol <- list(method = "genoud", s = 40, maxit = 40)
critcontrol  <- list(tolConstraints = .15, integration.points=as.matrix(test.grid))

SUR_grid <- apply(test.grid, 1, crit_SUR_cst, model.fun = model.fun,
              model.constraint = model.constraint1, equality = TRUE, critcontrol = critcontrol)

cstEGO2 <- critcst_optimizer(crit = "SUR", model.fun = model.fun,
                             model.constraint = model.constraint1, equality = TRUE,
                             lower = lower, upper = upper,
                             optimcontrol = optimcontrol, critcontrol = critcontrol)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(SUR_grid, n.grid), main = "SUR map and its maximizer (blue)",
               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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(obj.grid, n.grid), nlevels = 10, add=TRUE,
                            drawlabels=TRUE, col = "black")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                        matrix(cst1.grid, n.grid), level = c(-critcontrol$tolConstraints,
                            critcontrol$tolConstraints),
                            add=TRUE, drawlabels=FALSE,lwd=1.5, col = "orange")
                            points(cstEGO2$par, col = "blue", pch = 4, lwd = 2)
                            }
              )
```

`crit_AL` *Expected Augmented Lagrangian Improvement*

Description

Computes the Expected Augmented Lagrangian Improvement at current location, with our without slack variables. Depending on the cases, the computation is either analytical (very fast), based on MC integration (slow) or on the CDF of a weighted sum of non-central chi-square (WNCS) variates (intermediate)

Usage

```
crit_AL(x, model.fun, model.constraint, equality = FALSE,
  critcontrol = NULL, type = "UK")
```

Arguments

x either a vector representing the design or the design AND slack variables (see details)

model.fun object of class km correspostnding to the objective function, or, if the objective function is fast-to-evaluate, a fastfun object,

model.constraint either one or a list of objects of class km, one for each constraint function,

equality either FALSE if all constraints are for inequalities, or a vector of Booleans indicating which are equalities

critcontrol optional list with the following arguments:

- slack: logical. If TRUE, slack variables are used for inequality constraints (see Details)
- rho: penalty term (scalar),
- lambda: Lagrange multipliers (vector of size the number of constraints),
- elit: logical. If TRUE, sets the criterion to zero for all x's not improving the objective function
- n.mc: number of Monte-Carlo drawings used to evaluate the criterion (see Details)
- nt: number of discretization points for the WNCS distribution (see Details)
- tolConstraints, an optional vector giving a tolerance (> 0) for each of the constraints (equality or inequality). It is highly recommended to use it when there are equality constraints since the default tolerance of 0.05 in such case might not be suited.

Options for the checkPredict function: threshold (1e-4) and distance (covdist) are used to avoid numerical issues occuring 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 AL can be used with or without the help of slack variables for the inequality constraints. If critcontrol$slack=FALSE: With a single constraint (inequality or equality) and a fast objective, a very fast formula is used to compute the criterion (recommended setting). Otherwise, an MC estimator of the criterion is used, which is much more costly. The argument critcontrol$n.mc tunes the precision of the estimator. On both cases x must be of size d.

If critcontrol$slack=TRUE: Slack variables are used to handle the inequality constraints. They can be provided directly through x, which should be of size d+ the number of inequality constraints. The last values of x are slack variables scaled to [0,1].

If x is of size d, estimates of optimal slack variable are used.

Value

The Expected Augmented Lagrangian Improvement at x.

Author(s)

Victor Picheny

Mickael Binois

References

R.B. Gramacy, G.A. Gray, S. Le Digabel, H.K.H Lee, P. Ranjan, G. Wells, Garth, and S.M. Wild (2014+), Modeling an augmented Lagrangian for improved blackbox constrained optimization, *arXiv preprint arXiv:1403.4890*.

See Also

EI from package DiceOptim, crit_EFI, crit_SUR_cst.

Examples

```
#---------------------------------------------------------------------------
# Expected Augmented Lagrangian Improvement surface with one inequality constraint,
# fast objective
#---------------------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

n_var <- 2
fun.obj <- goldsteinprice
fun.cst <- function(x){return(-branin(x) + 25)}
n.grid <- 31
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun.obj)
cst.grid <- apply(test.grid, 1, fun.cst)
n.init <- 15
design.grid <- round(maximinESE_LHS(lhsDesign(n.init, n_var, seed = 42)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun.obj)
cst.init <- apply(design.grid, 1, fun.cst)
model.constraint <- km(~., design = design.grid, response = cst.init)
model.fun <- fastfun(fun.obj, design.grid)
AL_grid <- apply(test.grid, 1, crit_AL, model.fun = model.fun,
                 model.constraint = model.constraint)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(AL_grid, n.grid), main = "Expected AL 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")
                        contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                              matrix(obj.grid, n.grid), nlevels = 10,
                                     add=TRUE,drawlabels=TRUE, col = "black")
                        contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                              matrix(cst.grid, n.grid), level = 0, add=TRUE,
                                     drawlabels=FALSE,lwd=1.5, col = "red")
```

```
                        }
                )

#---------------------------------------------------------------------------
# Expected AL Improvement surface with one inequality and one equality constraint,
# using slack variables
#---------------------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

n_var <- 2
fun.obj <- goldsteinprice
fun.cstineq <- function(x){return(3/2 - x[1] - 2*x[2] - .5*sin(2*pi*(x[1]^2 - 2*x[2])))}
fun.csteq <- function(x){return(branin(x) - 25)}
n.grid <- 51
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun.obj)
cstineq.grid <- apply(test.grid, 1, fun.cstineq)
csteq.grid <- apply(test.grid, 1, fun.csteq)
n.init <- 25
design.grid <- round(maximinESE_LHS(lhsDesign(n.init, n_var, seed = 42)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun.obj)
cstineq.init <- apply(design.grid, 1, fun.cstineq)
csteq.init <- apply(design.grid, 1, fun.csteq)
model.fun <- km(~., design = design.grid, response = obj.init)
model.constraintineq <- km(~., design = design.grid, response = cstineq.init)
model.constrainteq <- km(~., design = design.grid, response = csteq.init)

models.cst <- list(model.constraintineq, model.constrainteq)

AL_grid <- apply(test.grid, 1, crit_AL, model.fun = model.fun, model.constraint = models.cst,
               equality = c(FALSE, TRUE), critcontrol = list(tolConstraints = c(0.05, 3),
                  slack=TRUE))

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(AL_grid, n.grid), main = "Expected AL 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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(obj.grid, n.grid), nlevels = 10,
                                   add=TRUE,drawlabels=TRUE, col = "black")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(cstineq.grid, n.grid), level = 0, add=TRUE,
                                   drawlabels=FALSE,lwd=1.5, col = "red")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(csteq.grid, n.grid), level = 0, add=TRUE,
                                   drawlabels=FALSE,lwd=1.5, col = "orange")
                        }
                )
```

crit_EFI *Expected Feasible Improvement*

Description

Computes the Expected Feasible Improvement at current location. The current feasible minimum of the observations can be replaced by an arbitrary value (plugin), which is usefull in particular in noisy frameworks.

Usage

```
crit_EFI(x, model.fun, model.constraint, equality = FALSE,
  critcontrol = NULL, plugin = NULL, type = "UK")
```

Arguments

`x`	a vector representing the input for which one wishes to calculate `EFI`,
`model.fun`	object of class `km` corresponding to the objective function, or, if the objective function is fast-to-evaluate, a `fastfun` object,
`model.constraint`	either one or a list of objects of class `km`, one for each constraint function,
`equality`	either `FALSE` if all constraints are for inequalities, else a vector of boolean indicating which are equalities,
`critcontrol`	optional list with argument `tolConstraints`, an optional vector giving a tolerance (> 0) for each of the constraints (equality or inequality). It is highly recommended to use it when there are equality constraints since the default tolerance of 0.05 in such case might not be suited. Options for the `checkPredict` function: `threshold` (`1e-4`) and `distance` (`covdist`) are used to avoid numerical issues occuring when adding points too close to the existing ones.
`plugin`	optional scalar: if provided, it replaces the feasible minimum of the current observations. If set to `Inf`, e.g. when their is no feasible solution, then the criterion is equal to the probability of feasibility,
`type`	`"SK"` or `"UK"` (by default), depending whether uncertainty related to trend estimation has to be taken into account.

Value

The Expected Feasible Improvement at `x`.

Author(s)

Victor Picheny

Mickael Binois

References

M. Schonlau, W.J. Welch, and D.R. Jones (1998), Global versus local search in constrained optimization of computer models, *Lecture Notes-Monograph Series*, 11-25.

M.J. Sasena, P. Papalambros, and P.Goovaerts (2002), Exploration of metamodeling sampling criteria for constrained global optimization, *Engineering optimization*, 34, 263-278.

See Also

EI from package DiceOptim, crit_AL, crit_SUR_cst.

Examples

```
#---------------------------------------------------------------------------
# Expected Feasible Improvement surface with one inequality constraint
#---------------------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

n_var <- 2
fun.obj <- goldsteinprice
fun.cst <- function(x){return(-branin(x) + 25)}
n.grid <- 51
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun.obj)
cst.grid <- apply(test.grid, 1, fun.cst)
n.init <- 15
design.grid <- round(maximinESE_LHS(lhsDesign(n.init, n_var, seed = 42)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun.obj)
cst.init <- apply(design.grid, 1, fun.cst)
model.fun <- km(~., design = design.grid, response = obj.init)
model.constraint <- km(~., design = design.grid, response = cst.init)

EFI_grid <- apply(test.grid, 1, crit_EFI, model.fun = model.fun,
                  model.constraint = model.constraint)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(EFI_grid, n.grid), main = "Expected Feasible 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")
                      contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                           matrix(obj.grid, n.grid), nlevels = 10,
                                  add=TRUE,drawlabels=TRUE, col = "black")
                      contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                           matrix(cst.grid, n.grid), level = 0, add=TRUE,
                                  drawlabels=FALSE,lwd=1.5, col = "red")
                           }
              )

#---------------------------------------------------------------------------
```

```
# Expected Feasible Improvement surface with one inequality and one equality constraint
#---------------------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

n_var <- 2
fun.obj <- goldsteinprice
fun.cstineq <- function(x){return(3/2 - x[1] - 2*x[2] - .5*sin(2*pi*(x[1]^2 - 2*x[2])))}
fun.csteq <- function(x){return(branin(x) - 25)}
n.grid <- 51
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun.obj)
cstineq.grid <- apply(test.grid, 1, fun.cstineq)
csteq.grid <- apply(test.grid, 1, fun.csteq)
n.init <- 25
design.grid <- round(maximinESE_LHS(lhsDesign(n.init, n_var, seed = 42)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun.obj)
cstineq.init <- apply(design.grid, 1, fun.cstineq)
csteq.init <- apply(design.grid, 1, fun.csteq)
model.fun <- km(~., design = design.grid, response = obj.init)
model.constraintineq <- km(~., design = design.grid, response = cstineq.init)
model.constrainteq <- km(~., design = design.grid, response = csteq.init)

models.cst <- list(model.constraintineq, model.constrainteq)

EFI_grid <- apply(test.grid, 1, crit_EFI, model.fun = model.fun, model.constraint = models.cst,
              equality = c(FALSE, TRUE), critcontrol = list(tolConstraints = c(0.05, 3)))

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(EFI_grid, n.grid), main = "Expected Feasible 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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(obj.grid, n.grid), nlevels = 10,
                                   add=TRUE,drawlabels=TRUE, col = "black")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(cstineq.grid, n.grid), level = 0, add=TRUE,
                                   drawlabels=FALSE,lwd=1.5, col = "red")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(csteq.grid, n.grid), level = 0, add=TRUE,
                                   drawlabels=FALSE,lwd=1.5, col = "orange")
                            }
              )
```

`crit_SUR_cst` *Stepwise Uncertainty Reduction criterion*

Description

Computes the Stepwise Uncertainty Reduction (SUR) criterion at current location

Usage

```
crit_SUR_cst(x, model.fun, model.constraint, equality = FALSE,
  critcontrol = NULL, type = "UK")
```

Arguments

`x`	a vector representing the input for which one wishes to calculate `SUR`,
`model.fun`	object of class `km` corresponding to the objective function, or, if the objective function is fast-to-evaluate, a `fastfun` object,
`model.constraint`	either one or a list of objects of class `km`, one for each constraint function,
`equality`	either `FALSE` if all constraints are for inequalities, else a vector of boolean indicating which are equalities
`critcontrol`	optional list with arguments: • `tolConstraints` optional vector giving a tolerance (> 0) for each of the constraints (equality or inequality). It is highly recommended to use it when there are equality constraints since the default tolerance of 0.05 in such case might not be suited; • `integration.points` and `integration.weights`: optional matrix and vector of integration points; • `precalc.data.cst`, `precalc.data.obj`, `mn.X.cst`, `sn.X.cst`, `mn.X.obj`, `sn.X.obj`: useful quantities for the fast evaluation of the criterion. • Options for the `checkPredict` function: `threshold` (`1e-4`) and `distance` (`covdist`) are used to avoid numerical issues occuring 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

The Stepwise Uncertainty Reduction criterion at `x`.

Author(s)

Victor Picheny

Mickael Binois

References

V. Picheny (2014), A stepwise uncertainty reduction approach to constrained global optimization, *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, JMLR W&CP 33, 787-795.

See Also

EI from package DiceOptim, crit_EFI, crit_AL.

Examples

```
#---------------------------------------------------------------------------
# Stepwise Uncertainty Reduction criterion surface with one inequality constraint
#---------------------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

n_var <- 2
fun.obj <- goldsteinprice
fun.cst <- function(x){return(-branin(x) + 25)}
n.grid <- 21
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun.obj)
cst.grid <- apply(test.grid, 1, fun.cst)

n_appr <- 15
design.grid <- round(maximinESE_LHS(lhsDesign(n_appr, n_var, seed = 42)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun.obj)
cst.init <- apply(design.grid, 1, fun.cst)
model.fun <- km(~., design = design.grid, response = obj.init)
model.constraint <- km(~., design = design.grid, response = cst.init)

integration.param <- integration_design_cst(integcontrol =list(integration.points = test.grid),
                                           lower = rep(0, n_var), upper = rep(1, n_var))

SUR_grid <- apply(test.grid, 1, crit_SUR_cst, model.fun = model.fun,
                  model.constraint = model.constraint, critcontrol=integration.param)

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(SUR_grid, n.grid), main = "SUR 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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(obj.grid, n.grid), nlevels = 10,
                                   add=TRUE,drawlabels=TRUE, col = "black")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(cst.grid, n.grid), level = 0, add=TRUE,
                                   drawlabels=FALSE,lwd=1.5, col = "red")
                            }
              )

#---------------------------------------------------------------------------
# SUR with one inequality and one equality constraint
#---------------------------------------------------------------------------

set.seed(25468)
```

```
library(DiceDesign)

n_var <- 2
fun.obj <- goldsteinprice
fun.cstineq <- function(x){return(3/2 - x[1] - 2*x[2] - .5*sin(2*pi*(x[1]^2 - 2*x[2])))}
fun.csteq <- function(x){return(branin(x) - 25)}
n.grid <- 21
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun.obj)
cstineq.grid <- apply(test.grid, 1, fun.cstineq)
csteq.grid <- apply(test.grid, 1, fun.csteq)
n_appr <- 25
design.grid <- round(maximinESE_LHS(lhsDesign(n_appr, n_var, seed = 42)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun.obj)
cstineq.init <- apply(design.grid, 1, fun.cstineq)
csteq.init <- apply(design.grid, 1, fun.csteq)
model.fun <- km(~., design = design.grid, response = obj.init)
model.constraintineq <- km(~., design = design.grid, response = cstineq.init)
model.constrainteq <- km(~., design = design.grid, response = csteq.init)

models.cst <- list(model.constraintineq, model.constrainteq)

SUR_grid <- apply(test.grid, 1, crit_SUR_cst, model.fun = model.fun, model.constraint = models.cst,
               equality = c(FALSE, TRUE), critcontrol = list(tolConstraints = c(0.05, 3),
                  integration.points=integration.param$integration.points))

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(SUR_grid, n.grid), main = "SUR 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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(obj.grid, n.grid), nlevels = 10,
                                   add=TRUE,drawlabels=TRUE, col = "black")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(cstineq.grid, n.grid), level = 0, add=TRUE,
                                   drawlabels=FALSE,lwd=1.5, col = "red")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                            matrix(csteq.grid, n.grid), level = 0, add=TRUE,
                                   drawlabels=FALSE,lwd=1.5, col = "orange")
                            }
              )
```

`easyEGO.cst` *EGO algorithm with constraints*

Description

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

Usage

```
easyEGO.cst(fun, constraint, n.cst = 1, budget, lower, upper,
  cheapfun = FALSE, equality = FALSE, X = NULL, y = NULL, C = NULL,
  control = list(method = "EFI", trace = 1, inneroptim = "genoud", maxit =
  100, seed = 42), ...)
```

Arguments

`fun`	scalar function to be minimized,
`constraint`	vectorial function corresponding to the constraints, see details below,
`n.cst`	number of constraints,
`budget`	total number of calls to the objective and constraint 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,
`cheapfun`	optional boolean, `TRUE` if the objective is a fast-to-evaluate function that does not need a kriging model
`equality`	either `FALSE` if all constraints are inequalities, else a Boolean vector indicating which are equalities
`X`	initial design of experiments. If not provided, X is taken as a maximin LHD with budget/3 points
`y`	initial set of objective observations $f(X)$. Computed if not provided.
`C`	initial set of constraint observations $g(X)$. Computed if not provided.
`control`	an optional list of control parameters. See "Details".
`...`	additional parameters to be given to BOTH the objective `fun` and `constraints`.

Details

Does not require knowledge on kriging models (objects of class `km`)

The problem considered is of the form: $min f(x)$ s.t. $g(x) \leq 0$, g having a vectorial output. By default all its components are supposed to be inequalities, but one can use a Boolean vector in `equality` to specify which are equality constraints, hence of the type $g(x) = 0$. The `control` argument is a list that can supply any of the following components:

- `method`: choice of constrained improvement function: "AL", "EFI" or "SUR" (see `crit_EFI`, `crit_AL`, `crit_SUR_cst`)
- `trace`: if positive, tracing information on the progress of the optimization is produced.
- `inneroptim`: choice of the inner optimization algorithm: "genoud" or "random" (see `genoud`).
- `maxit`: maximum number of iterations of the inner loop.
- `seed`: to fix the random variable generator

For additional details, see `EGO.cst`.

Value

A list with components:

- `par`: the best feasible point
- `values`: a vector of the objective and constraints at the point given in `par`,
- `history`: a list containing all the points visited by the algorithm (`X`) and their corresponding objectives (`y`) and constraints (`C`)

  If no feasible point is found, `par` returns the most feasible point (in the least square sense).

Author(s)

Victor Picheny

Mickael Binois

References

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

M. Schonlau, W.J. Welch, and D.R. Jones (1998), Global versus local search in constrained optimization of computer models, *Lecture Notes-Monograph Series*, 11-25.

M.J. Sasena, P. Papalambros, and P.Goovaerts (2002), Exploration of metamodeling sampling criteria for constrained global optimization, *Engineering optimization*, 34, 263-278.

R.B. Gramacy, G.A. Gray, S. Le Digabel, H.K.H Lee, P. Ranjan, G. Wells, Garth, and S.M. Wild (2014+), Modeling an augmented Lagrangian for improved blackbox constrained optimization, *arXiv preprint arXiv:1403.4890*.

J.M. Parr (2012), *Improvement criteria for constraint handling and multiobjective optimization*, University of Southampton.

V. Picheny (2014), A stepwise uncertainty reduction approach to constrained global optimization, *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, JMLR W&CP 33, 787-795.

Examples

```
#---------------------------------------------------------------------------
# 2D objective function, 3 cases
#---------------------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

n_var <- 2
fun <- goldsteinprice
fun1.cst <- function(x){return(-branin(x) + 25)}
fun2.cst <- function(x){return(3/2 - x[1] - 2*x[2] - .5*sin(2*pi*(x[1]^2 - 2*x[2])))}

# Constraint function with vectorial output
constraint <- function(x){return(c(fun1.cst(x), fun2.cst(x)))}
```

```
# For illustration purposes
n.grid <- 31
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun)
cst1.grid <- apply(test.grid, 1, fun1.cst)
cst2.grid <- apply(test.grid, 1, fun2.cst)

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

#---------------------------------------------------------------------------
# 1- Expected Feasible Improvement criterion, expensive objective function,
# two inequality constraints, 15 observations budget, using genoud
#---------------------------------------------------------------------------
res <- easyEGO.cst(fun=fun, constraint=constraint, n.cst=2, lower=lower, upper=upper, budget=15,
                   control=list(method="EFI", inneroptim="genoud", maxit=20))

cat("best design found:", res$par, "\n")
cat("corresponding objective and constraints:", res$value, "\n")

# Objective function in colour, constraint boundaries in red
# Initial DoE: white circles, added points: blue crosses, best solution: red cross

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(obj.grid, n.grid), main = "Two inequality constraints",
               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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                                     matrix(cst1.grid, n.grid), level = 0, add=TRUE,
                                     drawlabels=FALSE, lwd=1.5, col = "red")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                             matrix(cst2.grid, n.grid), level = 0, add=TRUE,drawlabels=FALSE,
                                     lwd=1.5, col = "red")
                            points(res$history$X, col = "blue", pch = 4, lwd = 2)
                       points(res$par[1], res$par[2], col = "red", pch = 4, lwd = 2, cex=2)
               }
)

#---------------------------------------------------------------------------
# 2- Augmented Lagrangian Improvement criterion, expensive objective function,
# one inequality and one equality constraint, 25 observations budget, using random search
#---------------------------------------------------------------------------
res2 <- easyEGO.cst(fun=fun, constraint=constraint, n.cst=2, lower=lower, upper=upper, budget=25,
                   equality = c(TRUE, FALSE),
                   control=list(method="AL", inneroptim="random", maxit=100))

cat("best design found:", res2$par, "\n")
cat("corresponding objective and constraints:", res2$value, "\n")

# Objective function in colour, inequality constraint boundary in red, equality
# constraint in orange
```

```
# Initial DoE: white circles, added points: blue crosses, best solution: red cross

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
              matrix(obj.grid, n.grid), xlab = expression(x[1]), ylab = expression(x[2]),
            main = "Inequality (red) and equality (orange) constraints", color = terrain.colors,
               plot.axes = {axis(1); axis(2);
                         points(design.grid[,1], design.grid[,2], pch = 21, bg = "white")
                      contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                                    matrix(cst1.grid, n.grid), level = 0, add=TRUE,
                                    drawlabels=FALSE,lwd=1.5, col = "orange")
                      contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                                    matrix(cst2.grid, n.grid), level = 0, add=TRUE,
                                    drawlabels=FALSE,lwd=1.5, col = "red")
                            points(res2$history$X, col = "blue", pch = 4, lwd = 2)
                      points(res2$par[1], res2$par[2], col = "red", pch = 4, lwd = 2, cex=2)
               }
)

#---------------------------------------------------------------------------
# 3- Stepwise Uncertainty Reduction criterion, fast objective function,
# single inequality constraint, with initial DOE given + 10 observations,
# using genoud
#---------------------------------------------------------------------------
n.init <- 12
design.grid <- round(maximinESE_LHS(lhsDesign(n.init, n_var, seed = 42)$design)$design, 1)
cst2.init <- apply(design.grid, 1, fun2.cst)

res3 <- easyEGO.cst(fun=fun, constraint=fun2.cst, n.cst=1, lower=lower, upper=upper, budget=10,
                    X=design.grid, C=cst2.init,
               cheapfun=TRUE, control=list(method="SUR", inneroptim="genoud", maxit=20))

cat("best design found:", res3$par, "\n")
cat("corresponding objective and constraint:", res3$value, "\n")

# Objective function in colour, inequality constraint boundary in red
# Initial DoE: white circles, added points: blue crosses, best solution: red cross

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(obj.grid, n.grid), main = "Single constraint, fast objective",
               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")
                      contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                                    matrix(obj.grid, n.grid), nlevels = 10, add = TRUE,
                                    drawlabels = TRUE)
                      contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                                    matrix(cst2.grid, n.grid), level = 0, add = TRUE,
                                    drawlabels = FALSE,lwd = 1.5, col = "red")
                            points(res3$history$X, col = "blue", pch = 4, lwd = 2)
                      points(res3$par[1], res3$par[2], col = "red", pch = 4, lwd = 2, cex=2)
                                           }
                )
```

EGO.cst *Sequential constrained Expected Improvement maximization and model re-estimation, with a number of iterations fixed in advance by the user*

Description

Executes nsteps iterations of EGO methods integrating constraints, based on 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 constrained Expected Improvement criteria available.

Usage

```
EGO.cst(model.fun = NULL, fun, cheapfun = NULL, model.constraint,
  constraint, equality = FALSE, crit = "EFI", nsteps, lower, upper,
  type = "UK", cov.reestim = TRUE, critcontrol = NULL,
  optimcontrol = list(method = "genoud", threshold = 1e-05, distance =
  "euclidean", notrace = FALSE), ...)
```

Arguments

model.fun	object of class km corresponding to the objective function,
fun	scalar function to be minimized, corresponding to model.fun found by a call to match.fun,
cheapfun	optional scalar function to use if the objective is a fast-to-evaluate function (handled next with class fastfun, through the use of match.fun), which does not need a kriging model, see details below,
model.constraint	either one or a list of models of class km, one per constraint,
constraint	vectorial function corresponding to the constraints, see details below,
equality	either FALSE if all constraints are for inequalities, else a vector of boolean indicating which are equalities
crit	choice of constrained improvement function: "AL", "EFI" 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,
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,

optimcontrol — an optional list of control parameters for optimization of the selected infill criterion:

- method can be set to "discrete" or "genoud". For "discrete", a matrix candidate.points must be given, For "genoud", specific parameters to the chosen method can also be specified (see genoud).
- Options for the checkPredict function: threshold (1e-4) and distance (covdist) are used to avoid numerical issues occuring when adding points too close to the existing ones.
- notrace can be set to TRUE to suppress printing of the optimization progresses.

... — additional parameters to be given to the objective fun and constraint.

Details

Extension of the function EGO.nsteps to constrained optimization.

The problem considered is of the form: $min f(x)$ s.t. $g(x) \leq 0$, g having a vectorial output. By default all its components are supposed to be inequalities, but one can use a boolean vector in equality to specify which are equality constraints. In this case one can modify the tolerance on the constraints using the tolConstraints component of critcontrol: an optional vector giving a tolerance for each of the constraints (equality or inequality). It is highly recommended to use it when there are equality constraints since the default tolerance of 0.05 in such case might not be suited.

Available infill criteria with crit are:

- Expected Probability of Feasibily (EFI) crit_EFI,
- Augmented Lagrangian (AL) crit_AL,
- Stepwise Uncertainty Reduction of the excursion volume (SUR) crit_SUR_cst.

Depending on the selected criterion, various parameters are available. More precisions are given in the corresponding help pages.

It is possible to consider a cheap to evaluate objective function submitted to expensive constraints. In this case, provide only a function in cheapfun, with both model.fun and fun to NULL, see examples below.

Value

A list with components:

- par: a matrix representing the additional points visited during the algorithm,
- values: a vector representing the response (objective) values at the points given in par,
- constraint: a matrix representing the constraints values at the points given in par,
- feasibility: a boolean vector saying if points given in par respect the constraints,

- `nsteps`: an integer representing the desired number of iterations (given in argument),
- `lastmodel.fun`: an object of class `km` corresponding to the objective function,
- `lastmodel.constraint`: one or a list of objects of class `km` corresponding to the last kriging models fitted to the constraints.

  If a problem occurs during either model updates or criterion maximization, the last working model and corresponding values are returned.

Author(s)

Victor Picheny

Mickael Binois

References

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

M. Schonlau, W.J. Welch, and D.R. Jones (1998), Global versus local search in constrained optimization of computer models, *Lecture Notes-Monograph Series*, 11-25.

M.J. Sasena, P. Papalambros, and P.Goovaerts (2002), Exploration of metamodeling sampling criteria for constrained global optimization, *Engineering optimization*, 34, 263-278.

R.B. Gramacy, G.A. Gray, S. Le Digabel, H.K.H Lee, P. Ranjan, G. Wells, Garth, and S.M. Wild (2014+), Modeling an augmented Lagrangian for improved blackbox constrained optimization, *arXiv preprint arXiv:1403.4890*.

J.M. Parr (2012), *Improvement criteria for constraint handling and multiobjective optimization*, University of Southampton.

V. Picheny (2014), A stepwise uncertainty reduction approach to constrained global optimization, *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, JMLR W&CP 33, 787-795.

See Also

`critcst_optimizer`, `crit_EFI`, `crit_AL`, `crit_SUR_cst`, `easyEGO.cst`

Examples

```
#---------------------------------------------------------------------------
# 2D objective function, 3 cases
#---------------------------------------------------------------------------

set.seed(25468)
library(DiceDesign)

n_var <- 2
fun <- goldsteinprice
fun1.cst <- function(x){return(-branin(x) + 25)}
fun2.cst <- function(x){return(3/2 - x[1] - 2*x[2] - .5*sin(2*pi*(x[1]^2 - 2*x[2])))}
```

```
# Constraint function with vectorial output
cstfun <- function(x){
  return(c(fun1.cst(x), fun2.cst(x)))
}

# For illustration purposes
n.grid <- 31
test.grid <- expand.grid(X1 = seq(0, 1, length.out = n.grid), X2 = seq(0, 1, length.out = n.grid))
obj.grid <- apply(test.grid, 1, fun)
cst1.grid <- apply(test.grid, 1, fun1.cst)
cst2.grid <- apply(test.grid, 1, fun2.cst)

# Initial set of observations and models
n.init <- 12
design.grid <- round(maximinESE_LHS(lhsDesign(n.init, n_var, seed = 42)$design)$design, 1)
obj.init <- apply(design.grid, 1, fun)
cst1.init <- apply(design.grid, 1, fun1.cst)
cst2.init <- apply(design.grid, 1, fun2.cst)
model.fun <- km(~., design = design.grid, response = obj.init)
model.constraint1 <- km(~., design = design.grid, response = cst1.init, lower=c(.2,.2))
model.constraint2 <- km(~., design = design.grid, response = cst2.init, lower=c(.2,.2))
model.constraint <- list(model.constraint1, model.constraint2)

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

#---------------------------------------------------------------------------
# 1- Expected Feasible Improvement criterion, expensive objective function,
# two inequality constraints, 5 iterations, using genoud
#---------------------------------------------------------------------------

cstEGO <- EGO.cst(model.fun = model.fun, fun = fun, model.constraint = model.constraint,
                  crit = "EFI", constraint = cstfun, equality = FALSE, lower = lower,
              upper = upper, nsteps = 5, optimcontrol = list(method = "genoud", maxit = 20))

# Plots: objective function in colour, constraint boundaries in red
# Initial DoE: white circles, added points: blue crosses, best solution: red cross

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(obj.grid, n.grid), main = "Two inequality constraints",
               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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                             matrix(cst1.grid, n.grid), level = 0, add=TRUE,drawlabels=FALSE,
                                    lwd=1.5, col = "red")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                             matrix(cst2.grid, n.grid), level = 0, add=TRUE,drawlabels=FALSE,
                                    lwd=1.5, col = "red")
                            points(cstEGO$par, col = "blue", pch = 4, lwd = 2)
               }
)
```

```
#---------------------------------------------------------------------------
# 2- Augmented Lagrangian Improvement criterion, expensive objective function,
# one inequality and one equality constraint, using a discrete set of candidates (grid)
#---------------------------------------------------------------------------
cstEGO2 <- EGO.cst(model.fun = model.fun, fun = fun, model.constraint = model.constraint,
                crit = "AL", constraint = cstfun, equality = c(TRUE, FALSE), lower = lower,
                    upper = upper, nsteps = 10,
                    critcontrol = list(tolConstraints = c(2, 0), always.update=TRUE),
                    optimcontrol=list(method="discrete", candidate.points=test.grid))

# Plots: objective function in colour, inequality constraint boundary in red,
# equality constraint in orange
# Initial DoE: white circles, added points: blue crosses, best solution: red cross

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(obj.grid, n.grid),
               main = "Inequality (red) and equality (orange) constraints",
               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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                               matrix(cst1.grid, n.grid), level = 0, add=TRUE,
                               drawlabels=FALSE,lwd=1.5, col = "orange")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                               matrix(cst2.grid, n.grid), level = 0, add=TRUE,
                               drawlabels=FALSE,lwd=1.5, col = "red")
                            points(cstEGO2$par, col = "blue", pch = 4, lwd = 2)
               }
)

#---------------------------------------------------------------------------
# 3- Stepwise Uncertainty Reduction criterion, fast objective function,
# single inequality constraint, 5 steps, importance sampling scheme
#---------------------------------------------------------------------------

cstEGO3 <- EGO.cst(model.fun = NULL, fun = NULL, cheapfun = fun,
                    model.constraint = model.constraint2, constraint = fun2.cst,
                    crit = "SUR", lower = lower, upper = upper,
                    nsteps =5, critcontrol=list(distrib="SUR"))

# Plots: objective function in colour, inequality constraint boundary in red,
# Initial DoE: white circles, added points: blue crosses, best solution: red cross

filled.contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid), nlevels = 50,
               matrix(obj.grid, n.grid), main = "Single constraint, fast objective",
               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")
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                               matrix(obj.grid, n.grid), nlevels = 10, add = TRUE,
                               drawlabels = TRUE)
                       contour(seq(0, 1, length.out = n.grid), seq(0, 1, length.out = n.grid),
                               matrix(cst2.grid, n.grid), level = 0, add=TRUE,
```

```
                        drawlabels=FALSE,lwd=1.5, col = "black")
                points(cstEGO3$par, col = "blue", pch = 4, lwd = 2)
                                }
)
```

EGO.nsteps — *Sequential EI maximization and model re-estimation, with a number of iterations fixed in advance by the user*

Description

Executes *nsteps* iterations of the EGO method to an object of class km. At each step, a kriging model is 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 the Expected Improvement criterion (EI).

Usage

```
EGO.nsteps(model, fun, nsteps, lower, upper, parinit = NULL, control = NULL,
  kmcontrol = NULL)
```

Arguments

model	an object of class km ,
fun	the objective function to be minimized,
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,
parinit	optional vector of initial values for the variables to be optimized over,
control	an optional list of control parameters for optimization. One can control "pop.size" (default : [4+3*log(nb of variables)]), "max.generations" (default :5), "wait.generations" (default :2), "BFGSburnin" (default :0), of the function genoud.
kmcontrol	an optional list representing the control variables for the re-estimation of the kriging model. The items are the same as in km : penalty, optim.method, parinit, control. The default values are those contained in model, typically corresponding to the variables used in km to estimate a kriging model from the initial design points.

Value

A list with components:

`par`	a data frame representing the additional points visited during the algorithm,
`value`	a data frame representing the response values at the points given in `par`,
`npoints`	an integer representing the number of parallel computations (=1 here),
`nsteps`	an integer representing the desired number of iterations (given in argument),
`lastmodel`	an object of class `km` corresponding to the last kriging model fitted.

Note

Most EGO-like methods (EI algorithms) usually work with Ordinary Kriging (constant trend), by maximization of the expected improvement. Here, the EI maximization is also possible with any linear trend. However, note that the optimization may perform much faster and better when the trend is a constant since it is the only case where the analytical gradient is available.

For more details on `kmcontrol`, see the documentation of `km`.

Author(s)

David Ginsbourger

Olivier Roustant

References

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

T.J. Santner, B.J. Williams, and W.J. Notz (2003), *The design and analysis of computer experiments*, Springer.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

`EI`, `max_EI`, `EI.grad`

Examples

```
set.seed(123)
################################################################
###  10 ITERATIONS OF EGO ON THE BRANIN FUNCTION,     ####
###  STARTING FROM A 9-POINTS FACTORIAL DESIGN         ####
################################################################

# a 9-points factorial design, and the corresponding response
d <- 2
n <- 9
```

```
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.branin <- apply(design.fact, 1, branin)
response.branin <- data.frame(response.branin)
names(response.branin) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.branin,
covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

# EGO n steps
library(rgenoud)
nsteps <- 5 # Was 10, reduced to 5 for speeding up compilation
lower <- rep(0,d)
upper <- rep(1,d)
oEGO <- EGO.nsteps(model=fitted.model1, fun=branin, nsteps=nsteps,
lower=lower, upper=upper, control=list(pop.size=20, BFGSburnin=2))
print(oEGO$par)
print(oEGO$value)

# graphics
n.grid <- 15 # Was 20, reduced to 15 for speeding up compilation
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
response.grid <- apply(design.grid, 1, branin)
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid, y.grid, z.grid, 40)
title("Branin function")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par, pch=19, col="red")
text(oEGO$par[,1], oEGO$par[,2], labels=1:nsteps, pos=3)


###############################################################
###  20 ITERATIONS OF EGO ON THE GOLDSTEIN-PRICE,      ####
###  STARTING FROM A 9-POINTS FACTORIAL DESIGN       ####
###############################################################
## Not run:
# a 9-points factorial design, and the corresponding response
d <- 2
n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.goldsteinPrice <- apply(design.fact, 1, goldsteinPrice)
response.goldsteinPrice <- data.frame(response.goldsteinPrice)
names(response.goldsteinPrice) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.goldsteinPrice,
```

```
covtype="gauss", control=list(pop.size=50, max.generations=50,
wait.generations=5, BFGSburnin=10,trace=FALSE), parinit=c(0.5, 0.5), optim.method="BFGS")

# EGO n steps
library(rgenoud)
nsteps <- 10 # Was 20, reduced to 10 for speeding up compilation
lower <- rep(0,d)
upper <- rep(1,d)
oEGO <- EGO.nsteps(model=fitted.model1, fun=goldsteinPrice, nsteps=nsteps,
lower, upper, control=list(pop.size=20, BFGSburnin=2))
print(oEGO$par)
print(oEGO$value)

# graphics
n.grid <- 15 # Was 20, reduced to 15 for speeding up compilation
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
response.grid <- apply(design.grid, 1, goldsteinPrice)
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid, y.grid, z.grid, 40)
title("Goldstein-Price Function")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par, pch=19, col="red")
text(oEGO$par[,1], oEGO$par[,2], labels=1:nsteps, pos=3)

## End(Not run)

#######################################################################
###  nsteps ITERATIONS OF EGO ON THE HARTMAN6 FUNCTION,      ####
###   STARTING FROM A 10-POINTS UNIFORM DESIGN          ####
#######################################################################

## Not run:
fonction<-hartman6
data(mydata)
a <- mydata
nb<-10
nsteps <- 3 # Maybe be changed to a larger value
x1<-a[[1]][1:nb];x2<-a[[2]][1:nb];x3<-a[[3]][1:nb]
x4<-a[[4]][1:nb];x5<-a[[5]][1:nb];x6<-a[[6]][1:nb]
design <- data.frame(cbind(x1,x2,x3,x4,x5,x6))
names(design)<-c("x1", "x2","x3","x4","x5","x6")
n <- nrow(design)

response <- data.frame(q=apply(design,1,fonction))
names(response) <- "y"
fitted.model1 <- km(~1, design=design, response=response, covtype="gauss",
control=list(pop.size=50, max.generations=20, wait.generations=5, BFGSburnin=5,
trace=FALSE), optim.method="gen", parinit=rep(0.8,6))

res.nsteps <- EGO.nsteps(model=fitted.model1, fun=fonction, nsteps=nsteps,
lower=rep(0,6), upper=rep(1,6), parinit=rep(0.5,6), control=list(pop.size=50,
max.generations=20, wait.generations=5, BFGSburnin=5), kmcontrol=NULL)
```

```
print(res.nsteps)
plot(res.nsteps$value,type="l")

## End(Not run)
```

EI *Analytical expression of the Expected Improvement criterion*

Description

Computes the Expected Improvement at current location. The current minimum of the observations can be replaced by an arbitrary value (plugin), which is usefull in particular in noisy frameworks.

Usage

```
EI(x, model, plugin = NULL, type = "UK", minimization = TRUE,
  envir = NULL)
```

Arguments

x	a vector representing the input for which one wishes to calculate EI,
model	an object of class km,
plugin	optional scalar: if provided, it replaces the minimum of the current observations,
type	"SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account,
minimization	logical specifying if EI is used in minimiziation or in maximization,
envir	an optional environment specifying where to assign intermediate values for future gradient calculations. Default is NULL.

Value

The expected improvement, defined as

$$EI(x) := E[(min(Y(X)) - Y(x))^{+}|Y(X) = y(X)],$$

where X is the current design of experiments and Y is the random process assumed to have generated the objective function y. If a plugin is specified, it replaces

$$min(Y(X))$$

in the previous formula.

Author(s)

David Ginsbourger

Olivier Roustant

Victor Picheny

References

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

T.J. Santner, B.J. Williams, and W.J. Notz (2003), *The design and analysis of computer experiments*, Springer.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

max_EI, EGO.nsteps, qEI

Examples

```
set.seed(123)
##########################################################################
###    EI SURFACE ASSOCIATED WITH AN ORDINARY KRIGING MODEL        ####
###    OF THE BRANIN FUNCTION KNOWN AT A 9-POINTS FACTORIAL DESIGN    ####
##########################################################################

# a 9-points factorial design, and the corresponding response
d <- 2; n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.branin <- apply(design.fact, 1, branin)
response.branin <- data.frame(response.branin)
names(response.branin) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.branin,
covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

# graphics
n.grid <- 12
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
#response.grid <- apply(design.grid, 1, branin)
EI.grid <- apply(design.grid, 1, EI,fitted.model1)
z.grid <- matrix(EI.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid,25)
title("Expected Improvement for the Branin function known at 9 points")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
```

EI.grad *Analytical gradient of the Expected Improvement criterion*

Description

Computes the gradient of the Expected Improvement at the current location. The current minimum of the observations can be replaced by an arbitrary value (plugin), which is usefull in particular in noisy frameworks.

Usage

```
EI.grad(x, model, plugin = NULL, type = "UK", minimization = TRUE,
  envir = NULL)
```

Arguments

x	a vector representing the input for which one wishes to calculate EI.
model	an object of class km.
plugin	optional scalar: if provided, it replaces the minimum of the current observations,
type	Kriging type: "SK" or "UK"
minimization	logical specifying if EI is used in minimiziation or in maximization,
envir	an optional environment specifying where to get intermediate values calculated in EI.

Value

The gradient of the expected improvement criterion with respect to x. Returns 0 at design points (where the gradient does not exist).

Author(s)

David Ginsbourger

Olivier Roustant

Victor Picheny

References

D. Ginsbourger (2009), *Multiples metamodeles pour l'approximation et l'optimisation de fonctions numeriques multivariables*, Ph.D. thesis, Ecole Nationale Superieure des Mines de Saint-Etienne, 2009. https://tel.archives-ouvertes.fr/tel-00772384/

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

T.J. Santner, B.J. Williams, and W.J. Notz (2003), *The design and analysis of computer experiments*, Springer.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

EI

Examples

```
## Not run:
set.seed(123)
# a 9-points factorial design, and the corresponding response
d <- 2; n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.branin <- apply(design.fact, 1, branin)
response.branin <- data.frame(response.branin)
names(response.branin) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.branin,
covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

# graphics
n.grid <- 50
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
#response.grid <- apply(design.grid, 1, branin)
EI.grid <- apply(design.grid, 1, EI,fitted.model1)
#EI.grid <- apply(design.grid, 1, EI.plot,fitted.model1, gr=TRUE)

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

contour(x.grid,y.grid,z.grid,20)
title("Expected Improvement for the Branin function known at 9 points")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")

# graphics
n.gridx <- 15
n.gridy <- 20
x.grid2 <- seq(0,1,length=n.gridx)
y.grid2 <- seq(0,1,length=n.gridy)
design.grid2 <- expand.grid(x.grid2, y.grid2)

EI.envir <- new.env()
environment(EI) <- environment(EI.grad) <- EI.envir

options(warn=-1)
for(i in seq(1, nrow(design.grid2)) )
{
x <- design.grid2[i,]
ei <- EI(x, model=fitted.model1, envir=EI.envir)
eigrad <- EI.grad(x , model=fitted.model1, envir=EI.envir)
if(!(is.null(ei)))
```

```
{
arrows(x$Var1,x$Var2,
x$Var1 + eigrad[1]*2.2*10e-5, x$Var2 + eigrad[2]*2.2*10e-5,
length = 0.04, code=2, col="orange", lwd=2)
}
}

## End(Not run)
```

EQI *Expected Quantile Improvement*

Description

Evaluation of the Expected Quantile Improvement (EQI) criterion.

Usage

```
EQI(x, model, new.noise.var = 0, beta = 0.9, q.min = NULL, type = "UK",
  envir = NULL)
```

Arguments

`x`	the input vector at which one wants to evaluate the criterion
`model`	a Kriging model of "km" class
`new.noise.var`	(scalar) noise variance of the future observation. Default value is 0 (noise-free observation).
`beta`	Quantile level (default value is 0.9)
`q.min`	Best kriging quantile. If not provided, this quantity is evaluated.
`type`	Kriging type: "SK" or "UK"
`envir`	environment for saving intermediate calculations and reusing them within EQI.grad

Value

Expected Quantile Improvement

Author(s)

Victor Picheny

David Ginsbourger

References

Picheny, V., Ginsbourger, D., Richet, Y., Caplin, G. (2013). Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics*, 55(1), 2-13.

Examples

```
##########################################################################
###    EQI SURFACE ASSOCIATED WITH AN ORDINARY KRIGING MODEL        ####
### OF THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
##########################################################################

set.seed(421)

# Set test problem parameters
doe.size <- 12
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
            covtype="gauss", noise.var=rep(noise.var,1,doe.size),
    lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)

crit.grid <- apply(design.grid, 1, EQI, model=model, new.noise.var=noise.var, beta=.9)
func.grid <- apply(design.grid, 1, test.function)

# Compute kriging mean and variance on a grid
names(design.grid) <- c("V1","V2")
pred <- predict(model, newdata=design.grid, type="UK", checkNames = FALSE)
mk.grid <- pred$m
sk.grid <- pred$sd

# Plot actual function
z.grid <- matrix(func.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Actual function");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot Kriging mean
```

```
z.grid <- matrix(mk.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Kriging mean");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot Kriging variance
z.grid <- matrix(sk.grid^2, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Kriging variance");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot EQI criterion
z.grid <- matrix(crit.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("EQI");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})
```

EQI.grad *EQI's Gradient*

Description

Analytical gradient of the Expected Quantile Improvement (EQI) criterion.

Usage

```
EQI.grad(x, model, new.noise.var = 0, beta = 0.9, q.min = NULL,
  type = "UK", envir = NULL)
```

Arguments

x	the input vector at which one wants to evaluate the criterion
model	a Kriging model of "km" class
new.noise.var	(scalar) noise variance of the future observation. Default value is 0 (noise-free observation).
beta	Quantile level (default value is 0.9)
q.min	Best kriging quantile. If not provided, this quantity is evaluated.
type	Kriging type: "SK" or "UK"
envir	environment for inheriting intermediate calculations from EQI

Value

Gradient of the Expected Quantile Improvement

Author(s)

Victor Picheny

David Ginsbourger

Examples

```
## Not run:
set.seed(421)

# Set test problem parameters
doe.size <- 12
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
        covtype="gauss", noise.var=rep(noise.var,1,doe.size),
lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)

crit.grid <- apply(design.grid, 1, EQI, model=model, new.noise.var=noise.var, beta=.9)
crit.grad <- t(apply(design.grid, 1, EQI.grad, model=model, new.noise.var=noise.var, beta=.9))

z.grid <- matrix(crit.grid, n.grid, n.grid)
contour(x.grid,y.grid, z.grid, 30)
title("EQI and its gradient")
points(model@X[,1],model@X[,2],pch=17,col="blue")

options(warn=-1)
for (i in 1:nt)
{
 x <- design.grid[i,]
 arrows(x$Var1,x$Var2, x$Var1+crit.grad[i,1]*.2,x$Var2+crit.grad[i,2]*.2,
length=0.04,code=2,col="orange",lwd=2)
}
```

```
## End(Not run)
```

fastfun *Fastfun function*

Description

Modification of an R function to be used as 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

```
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

```
#########################################################
## Example with a fast to evaluate objective
#########################################################
## Not run:
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 = "pso")
omEGO1 <- GParetoptim(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))^2 + 5/pi*b1 - 6)^2 +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))^2 - 6)^2
         - 1/3*((1 - 1/(8*pi))*cos(b1) + 1))
}

optimcontrol <- list(method = "pso")
model2 <- list(mf1)
omEGO2 <- GParetoptim(model = model2, fn = f1, cheapfn = f2, crit = "SMS", nsteps = nsteps,
                     lower = lower, upper = upper, optimcontrol = optimcontrol)
print(omEGO2$par)
print(omEGO2$values)

points(omEGO2$values, col = "red", pch = 15)

## End(Not run)
```

`integration_design_cst` *Generic function to build integration points (for the SUR criterion)*

Description

Modification of the function `integration_design` from the package `KrigInv` to be usable for SUR-based optimization with constraints.

Usage

```
integration_design_cst(integcontrol = NULL, lower, upper, model.fun = NULL,
  model.constraint = NULL, equality = FALSE, critcontrol = NULL,
  min.prob = 0.001)
```

Arguments

`integcontrol` Optional list specifying the procedure to build the integration points and weights. Many options are possible.
A) If nothing is specified, 100*d points are chosen using the Sobol sequence.
B) One can directly set the field `integration.points` (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.

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

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

`model.fun` object of class `km` corresponding to the objective functions, or, if the objective function is fast-to-evaluate, a `fastfun` object,

`model.constraint` either one or a list of objects of class `km`, one for each constraint function,

`equality` either `FALSE` if all constraints are for inequalities, else a vector of boolean indicating which are equalities

`critcontrol` optional list of parameters (see `crit_SUR_cst`); here only the component `tolConstraints` is used.

`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`.

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

Author(s)

Victor Picheny

Mickael Binois

References

Chevalier C., Picheny V., Ginsbourger D. (2012), The KrigInv package: An efficient and user-friendly R implementation of Kriging-based inversion algorithms, *Computational Statistics and Data Analysis*, 71, 1021-1034.

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set, *Technometrics*, 56(4), 455-465.

V. Picheny (2014), A stepwise uncertainty reduction approach to constrained global optimization, *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, JMLR W&CP 33, 787-795.

See Also

`crit_SUR_cst` `integration_design`

`kriging.quantile` *Kriging quantile*

Description

Evaluation of a kriging quantile a a new point. To be used in an optimization loop.

Usage

```
kriging.quantile(x, model, beta = 0.1, type = "UK", envir = NULL)
```

Arguments

x	the input vector at which one wants to evaluate the criterion
model	a Kriging model of "km" class
beta	Quantile level (default value is 0.1)
type	Kriging type: "SK" or "UK"
envir	an optional environment specifying where to assign intermediate values for future gradient calculations. Default is NULL.

Value

Kriging quantile

Author(s)

Victor Picheny

David Ginsbourger

Examples

```
###########################################################################
###    KRIGING QUANTILE SURFACE                                        ####
### OF THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
###########################################################################

set.seed(421)

# Set test problem parameters
doe.size <- 12
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
            covtype="gauss", noise.var=rep(noise.var,1,doe.size),
    lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
```

```
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)

crit.grid <- apply(design.grid, 1, kriging.quantile, model=model, beta=.1)
func.grid <- apply(design.grid, 1, test.function)

# Compute kriging mean and variance on a grid
names(design.grid) <- c("V1","V2")
pred <- predict(model, newdata=design.grid, type="UK", checkNames = FALSE)
mk.grid <- pred$m
sk.grid <- pred$sd

# Plot actual function
z.grid <- matrix(func.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Actual function");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot Kriging mean
z.grid <- matrix(mk.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Kriging mean");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot Kriging variance
z.grid <- matrix(sk.grid^2, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("Kriging variance");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})

# Plot kriging.quantile criterion
z.grid <- matrix(crit.grid, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title("kriging.quantile");
points(model@X[,1],model@X[,2],pch=17,col="blue");
axis(1); axis(2)})
```

`kriging.quantile.grad` *Analytical gradient of the Kriging quantile of level beta*

Description

Computes the gradient of the Kriging quantile of level beta at the current location. Only available for Universal Kriging with constant trend (Ordinary Kriging).

Usage

```
kriging.quantile.grad(x, model, beta = 0.1, type = "UK", envir = NULL)
```

Arguments

`x`	a vector representing the input for which one wishes to calculate kriging.quantile.grad.
`model`	an object of class `km`.
`beta`	A quantile level (between 0 and 1)
`type`	Kriging type: "SK" or "UK"
`envir`	environment for inheriting intermediate calculations from `"kriging.quantile"`

Value

The gradient of the Kriging mean predictor with respect to x. Returns 0 at design points (where the gradient does not exist).

Author(s)

Victor Picheny

David Ginsbourger

References

O. Roustant, D. Ginsbourger, Y. Deville, *DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization*, submitted to J. Stat. Soft., 2010. https://hal.archives-ouvertes.fr/hal-00495766/

D. Ginsbourger (2009), *Multiples metamodeles pour l'approximation et l'optimisation de fonctions numeriques multivariables*, Ph.D. thesis, Ecole Nationale Superieure des Mines de Saint-Etienne, 2009. https://tel.archives-ouvertes.fr/tel-00772384

See Also

`EI.grad`

Examples

```
## Not run:
##########################################################################
###    KRIGING QUANTILE SURFACE AND ITS GRADIENT FOR                 ####
###    THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
##########################################################################
set.seed(421)

# Set test problem parameters
doe.size <- 12
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
```

```
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {
y.tilde[i] <- test.function(doe[i,]) + sqrt(noise.var)*rnorm(n=1)
}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
        covtype="gauss", noise.var=rep(noise.var,1,doe.size),
lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
nt <- nrow(design.grid)

crit.grid <- apply(design.grid, 1, kriging.quantile, model=model, beta=.1)
crit.grad <- t(apply(design.grid, 1, kriging.quantile.grad, model=model, beta=.1))

z.grid <- matrix(crit.grid, n.grid, n.grid)
contour(x.grid,y.grid, z.grid, 30)
title("kriging.quantile and its gradient")
points(model@X[,1],model@X[,2],pch=17,col="blue")

for (i in 1:nt)
{
 x <- design.grid[i,]
 arrows(x$Var1,x$Var2, x$Var1+crit.grad[i,1]*.01,x$Var2+crit.grad[i,2]*.01,
length=0.04,code=2,col="orange",lwd=2)
}

## End(Not run)
```

---

max_AEI *Maximizer of the Augmented Expected Improvement criterion function*

Description

Maximization, based on the package rgenoud of the Augmented Expected Improvement (AEI) criterion.

Usage

```
max_AEI(model, new.noise.var = 0, y.min = NULL, type = "UK", lower, upper,
  parinit = NULL, control = NULL)
```

Arguments

model	a Kriging model of "km" class
new.noise.var	the (scalar) noise variance of the new observation.
y.min	The kriging mean prediction at the current best point (point with smallest kriging quantile). If not provided, this quantity is evaluated inside the AEI function (may increase computational time).
type	Kriging type: "SK" or "UK"
lower	vector containing the lower bounds of the variables to be optimized over
upper	optional vector containing the upper bounds of the variables to be optimized over
parinit	optional vector containing the initial values for the variables to be optimized over
control	optional list of control parameters for optimization. One can control "pop.size" (default : [N=3*2^dim for dim<6 and N=32*dim otherwise]), "max.generations" (12), "wait.generations" (2) and "BFGSburnin" (2) of function "genoud" (see genoud). Numbers into brackets are the default values

Value

A list with components:

par	the best set of parameters found.
value	the value AEI at par.

Author(s)

Victor Picheny

David Ginsbourger

Examples

```
library(DiceDesign)
set.seed(100)

# Set test problem parameters
doe.size <- 10
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(lhsDesign(doe.size, dim)$design)
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {y.tilde[i] <- test.function(doe[i,])
+ sqrt(noise.var)*rnorm(n=1)}
```

```
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
     covtype="gauss", noise.var=rep(noise.var,1,doe.size),
     lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Optimisation using max_AEI
res <- max_AEI(model, new.noise.var=noise.var, type = "UK",
lower=c(0,0), upper=c(1,1))
X.genoud <- res$par

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
names(design.grid) <- c("V1","V2")
nt <- nrow(design.grid)
crit.grid <- apply(design.grid, 1, AEI, model=model, new.noise.var=noise.var)

## Not run:
# # 2D plots
z.grid <- matrix(crit.grid, n.grid, n.grid)
tit <- "Green: best point found by optimizer"
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title(tit);points(model@X[,1],model@X[,2],pch=17,col="blue");
points(X.genoud[1],X.genoud[2],pch=17,col="green");
axis(1); axis(2)})

## End(Not run)
```

`max_AKG` *Maximizer of the Expected Quantile Improvement criterion function*

Description

Maximization, based on the package rgenoud of the Expected Quantile Improvement (AKG) criterion.

Usage

```
max_AKG(model, new.noise.var = 0, type = "UK", lower, upper,
  parinit = NULL, control = NULL)
```

Arguments

`model`	a Kriging model of "km" class
`new.noise.var`	the (scalar) noise variance of an observation. Default value is 0 (noise-free observation).

type	Kriging type: "SK" or "UK"
lower	vector containing the lower bounds of the variables to be optimized over
upper	vector containing the upper bounds of the variables to be optimized over
parinit	optional vector containing the initial values for the variables to be optimized over
control	optional list of control parameters for optimization. One can control "pop.size" (default : [N=3*2^dim for dim<6 and N=32*dim otherwise]), "max.generations" (12), "wait.generations" (2) and "BFGSburnin" (2) of function "genoud" (see genoud). Numbers into brackets are the default values

Value

A list with components:

par	the best set of parameters found.
value	the value AKG at par.

Author(s)

Victor Picheny

David Ginsbourger

Examples

```
##########################################################################
###    AKG SURFACE AND OPTIMIZATION PERFORMED BY GENOUD              ####
###    FOR AN ORDINARY KRIGING MODEL                                 ####
### OF THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
##########################################################################
set.seed(10)

# Set test problem parameters
doe.size <- 10
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
library(DiceDesign)
doe <- as.data.frame(lhsDesign(doe.size, dim)$design)
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {y.tilde[i] <- test.function(doe[i,])
+ sqrt(noise.var)*rnorm(n=1)}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
     covtype="gauss", noise.var=rep(noise.var,1,doe.size),
```

```
     lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Optimisation using max_AKG
res <- max_AKG(model, new.noise.var=noise.var, type = "UK",
lower=c(0,0), upper=c(1,1))
X.genoud <- res$par

## Not run:
# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
names(design.grid) <- c("V1","V2")
nt <- nrow(design.grid)
crit.grid <- apply(design.grid, 1, AKG, model=model, new.noise.var=noise.var)

# # 2D plots
z.grid <- matrix(crit.grid, n.grid, n.grid)
tit <- "Green: best point found by optimizer"
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
plot.axes = {title(tit);points(model@X[,1],model@X[,2],pch=17,col="blue");
points(X.genoud[1],X.genoud[2],pch=17,col="green");
axis(1); axis(2)})

## End(Not run)
```

---

max_EI *Maximization of the Expected Improvement criterion*

Description

Given an object of class km and a set of tuning parameters (lower,upper,parinit, and control), max_EI performs the maximization of the Expected Improvement criterion and delivers the next point to be visited in an EGO-like procedure.

Usage

```
max_EI(model, plugin = NULL, type = "UK", lower, upper, parinit = NULL,
  minimization = TRUE, control = NULL)
```

Arguments

model	an object of class km ,
plugin	optional scalar: if provided, it replaces the minimum of the current observations,
type	Kriging type: "SK" or "UK"
lower	vector of lower bounds for the variables to be optimized over,
upper	vector of upper bounds for the variables to be optimized over,
parinit	optional vector of initial values for the variables to be optimized over,

`minimization` logical specifying if EI is used in minimiziation or in maximization,

`control` optional list of control parameters for optimization. One can control "pop.size" (default : [N=3*2^dim for dim<6 and N=32*dim otherwise]), "max.generations" (12), "wait.generations" (2) and "BFGSburnin" (2) of function "genoud" (see genoud). Numbers into brackets are the default values

Details

The latter maximization relies on a genetic algorithm using derivatives, genoud. This function plays a central role in the package since it is in constant use in the proposed algorithms. It is important to remark that the information needed about the objective function reduces here to the vector of response values embedded in `model` (no call to the objective function or simulator).

The current minimum of the observations can be replaced by an arbitrary value (plugin), which is usefull in particular in noisy frameworks.

Value

A list with components:

`par` The best set of parameters found.

`value` The value of expected improvement at par.

Author(s)

David Ginsbourger

Olivier Roustant

Victor Picheny

References

D. Ginsbourger (2009), *Multiples metamodeles pour l'approximation et l'optimisation de fonctions numeriques multivariables*, Ph.D. thesis, Ecole Nationale Superieure des Mines de Saint-Etienne, 2009. https://tel.archives-ouvertes.fr/tel-00772384/

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

W.R. Jr. Mebane and J.S. Sekhon (2009), in press, Genetic optimization using derivatives: The rgenoud package for R, *Journal of Statistical Software*.

Examples

```
set.seed(123)
##########################################################
### "ONE-SHOT" EI-MAXIMIZATION OF THE BRANIN FUNCTION ####
###  KNOWN AT A 9-POINTS FACTORIAL DESIGN          ####
##########################################################

# a 9-points factorial design, and the corresponding response
d <- 2
```

```
n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact) <- c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact) <- c("x1", "x2")
response.branin <- apply(design.fact, 1, branin)
response.branin <- data.frame(response.branin)
names(response.branin) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.branin,
covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

# EGO one step
library(rgenoud)
lower <- rep(0,d)
upper <- rep(1,d)     # domain for Branin function
oEGO <- max_EI(fitted.model1, lower=lower, upper=upper,
control=list(pop.size=20, BFGSburnin=2))
print(oEGO)

# graphics
n.grid <- 20
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
response.grid <- apply(design.grid, 1, branin)
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid,40)
title("Branin Function")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par[1], oEGO$par[2], pch=19, col="red")


#############################################################
### "ONE-SHOT" EI-MAXIMIZATION OF THE CAMELBACK FUNCTION ####
### KNOWN AT A 16-POINTS FACTORIAL DESIGN            ####
#############################################################
## Not run:
# a 16-points factorial design, and the corresponding response
d <- 2
n <- 16
design.fact <- expand.grid(seq(0,1,length=4), seq(0,1,length=4))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact) <- c("x1", "x2")
response.camelback <- apply(design.fact, 1, camelback)
response.camelback <- data.frame(response.camelback)
names(response.camelback) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.camelback,
covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))
```

```
# EI maximization
library(rgenoud)
lower <- rep(0,d)
upper <- rep(1,d)
oEGO <- max_EI(fitted.model1, lower=lower, upper=upper,
control=list(pop.size=20, BFGSburnin=2))
print(oEGO)

# graphics
n.grid <- 20
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
response.grid <- apply(design.grid, 1, camelback)
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid,40)
title("Camelback Function")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par[1], oEGO$par[2], pch=19, col="red")

## End(Not run)

####################################################################
### "ONE-SHOT" EI-MAXIMIZATION OF THE GOLDSTEIN-PRICE FUNCTION #####
###       KNOWN AT A 9-POINTS FACTORIAL DESIGN              #####
####################################################################

## Not run:
# a 9-points factorial design, and the corresponding response
d <- 2
n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.goldsteinPrice <- apply(design.fact, 1, goldsteinPrice)
response.goldsteinPrice <- data.frame(response.goldsteinPrice)
names(response.goldsteinPrice) <- "y"

# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.goldsteinPrice,
covtype="gauss", control=list(pop.size=50, max.generations=50,
wait.generations=5, BFGSburnin=10, trace=FALSE), parinit=c(0.5, 0.5), optim.method="gen")

# EI maximization
library(rgenoud)
lower <- rep(0,d); upper <- rep(1,d);     # domain for Branin function
oEGO <- max_EI(fitted.model1, lower=lower, upper=upper, control
=list(pop.size=50, max.generations=50, wait.generations=5, BFGSburnin=10))
print(oEGO)

# graphics
n.grid <- 20
x.grid <- y.grid <- seq(0,1,length=n.grid)
```

```
design.grid <- expand.grid(x.grid, y.grid)
response.grid <- apply(design.grid, 1, goldsteinPrice)
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid,40)
title("Goldstein-Price Function")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par[1], oEGO$par[2], pch=19, col="red")

## End(Not run)
```

---

max_EQI *Maximizer of the Expected Quantile Improvement criterion function*

Description

Maximization, based on the package rgenoud of the Expected Quantile Improvement (EQI) criterion.

Usage

```
max_EQI(model, new.noise.var = 0, beta = 0.9, q.min = NULL, type = "UK",
  lower, upper, parinit = NULL, control = NULL)
```

Arguments

`model`	a Kriging model of "km" class
`new.noise.var`	the (scalar) noise variance of an observation. Default value is 0 (noise-free observation).
`beta`	Quantile level (default value is 0.9)
`q.min`	The current best kriging quantile. If not provided, this quantity is evaluated inside the EQI function (may increase computational time).
`type`	Kriging type: "SK" or "UK"
`lower`	vector containing the lower bounds of the variables to be optimized over
`upper`	optional vector containing the upper bounds of the variables to be optimized over
`parinit`	optional vector containing the initial values for the variables to be optimized over
`control`	optional list of control parameters for optimization. One can control `"pop.size"` (default : [N=3*2^dim for dim<6 and N=32*dim otherwise]), `"max.generations"` (12), `"wait.generations"` (2) and `"BFGSburnin"` (2) of function `"genoud"` (see genoud). Numbers into brackets are the default values

Value

A list with components:

`par`	the best set of parameters found.
`value`	the value EQI at par.

Author(s)

Victor Picheny

David Ginsbourger

Examples

```
set.seed(10)

# Set test problem parameters
doe.size <- 10
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {y.tilde[i] <- test.function(doe[i,])
+ sqrt(noise.var)*rnorm(n=1)}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
     covtype="gauss", noise.var=rep(noise.var,1,doe.size),
     lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Optimisation using max_EQI
res <- max_EQI(model, new.noise.var=noise.var, type = "UK",
lower=c(0,0), upper=c(1,1))
X.genoud <- res$par

## Not run:
# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
names(design.grid) <- c("V1","V2")
nt <- nrow(design.grid)
crit.grid <- apply(design.grid, 1, EQI, model=model, new.noise.var=noise.var, beta=.9)

# # 2D plots
z.grid <- matrix(crit.grid, n.grid, n.grid)
tit <- "Green: best point found by optimizer"
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = rainbow,
plot.axes = {title(tit);points(model@X[,1],model@X[,2],pch=17,col="blue");
points(X.genoud[1],X.genoud[2],pch=17,col="green");
axis(1); axis(2)})

## End(Not run)
```

max_qEI *Maximization of multipoint expected improvement criterion (qEI)*

Description

Maximization of the qEI criterion. Two options are available : Constant Liar (CL), and brute force qEI maximization with Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, or GENetic Optimization Using Derivative (genoud) algorithm.

Usage

```
max_qEI(model, npoints, lower, upper, crit = "exact", minimization = TRUE,
  optimcontrol = NULL)
```

Arguments

model	an object of class km ,
npoints	an integer representing the desired number of iterations,
lower	vector of lower bounds,
upper	vector of upper bounds,
crit	"exact", "CL" : a string specifying the criterion used. "exact" triggers the maximization of the multipoint expected improvement at each iteration (see max_qEI), "CL" applies the Constant Liar heuristic,
minimization	logical specifying if the qEI to be maximized is used in minimiziation or in maximization,
optimcontrol	an optional list of control parameters for optimization. See details.

Details

- CL is a heuristic method. First, the regular Expected Improvement EI is maximized (max_EI). Then, for the next points, the Expected Improvement is maximized again, but with an artificially updated Kriging model. Since the response values corresponding to the last best point obtained are not available, the idea of CL is to replace them by an arbitrary constant value L (a "lie") set by the user (default is the minimum of all currently available observations).

- The BFGS algorithm is implemented in the standard function optim. Analytical formulae of qEI and its gradient qEI.grad are used. The nStarts starting points are by default sampled with respect to the regular EI (sampleFromEI) criterion.

- The "genoud" method calls the function genoud using analytical formulae of qEI and its gradient qEI.grad.

The parameters of list optimcontrol are :

- optimcontrol$method : "BFGS" (default), "genoud" ; a string specifying the method used to maximize the criterion (irrelevant when crit is "CL" because this method always uses genoud),

- when crit="CL" :

+ `optimcontrol$parinit` : optional matrix of initial values (must have model@d columns, the number of rows is not constrained),

+ `optimcontrol$L` : "max", "min", "mean" or a scalar value specifying the liar ; "min" takes `model@min`, "max" takes `model@max`, "mean" takes the prediction of the model ; When L is `NULL`, "min" is taken if `minimization==TRUE`, else it is "max".

+ The parameters of function `genoud`. Main parameters are : `"pop.size"` (default : [N=3*2^model@d for dim<6 and N=32*model@d otherwise]), `"max.generations"` (default : 12), `"wait.generations"` (default : 2) and `"BFGSburnin"` (default : 2).

- when `optimcontrol$method = "BFGS"` :

+ `optimcontrol$nStarts` (default : 4),

+ `optimcontrol$fastCompute` : if TRUE (default), a fast approximation method based on a semi-analytic formula is used, see [Marmin 2014] for details,

+ `optimcontrol$samplingFun` : a function which sample a batch of starting point (default : `sampleFromEI`),

+ `optimcontrol$parinit` : optional 3d-array of initial (or candidate) batches (for all `k`, parinit[,,k] is a matrix of size `npoints*model@d` representing one batch). The number of initial batches (length(parinit[1,1,])) is not contrained and does not have to be equal to `nStarts`. If there is too few initial batches for `nStarts`, missing batches are drawn with `samplingFun` (default : `NULL`),

- when `optimcontrol$method = "genoud"` :

+ `optimcontrol$fastCompute` : if TRUE (default), a fast approximation method based on a semi-analytic formula is used, see [Marmin 2014] for details,

+ `optimcontrol$parinit` : optional matrix of candidate starting points (one row corresponds to one point),

+ The parameters of the `genoud` function. Main parameters are `"pop.size"` (default : `[50*(model@d)*(npoints)]`), `"max.generations"` (default : 5), `"wait.generations"` (default : 2), `"BFGSburnin"` (default : 2).

Value

A list with components:

`par`	A matrix containing the `npoints` input vectors found.
`value`	A value giving the qEI computed in `par`.

Author(s)

Sebastien Marmin

Clement Chevalier

David Ginsbourger

References

C. Chevalier and D. Ginsbourger (2013), Fast computation of the multi-points expected improvement with applications in batch selection. In G. Nicosia and P. Pardalos (Eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp 59-69, Springer.

D. Ginsbourger, R. Le Riche, L. Carraro (2007), A Multipoint Criterion for Deterministic Parallel Global Optimization based on Kriging. The International Conference on Non Convex Programming, 2007.

D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization (2010), In Lim Meng Hiot, Yew Soon Ong, Yoel Tenne, and Chi-Keong Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, Adaptation Learning and Optimization, pages 131-162. Springer Berlin Heidelberg.

S. Marmin, C. Chevalier and D. Ginsbourger (2015), Differentiating the multipoint Expected Improvement for optimal batch design. In P. Pardalos et al. (Eds.) Machine Learning, Optimization, and Big Data, Lecture Notes in Computer Science 9432, pp 37-48.

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

qEI, qEI.grad

Examples

```
## Not run:

set.seed(000)
# 3-points EI maximization.
# 9-points factorial design, and the corresponding response
d <- 2
n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.branin <- apply(design.fact, 1, branin)
response.branin <- data.frame(response.branin)
names(response.branin) <- "y"
lower <- c(0,0)
upper <- c(1,1)

# number of point in the bacth
batchSize <- 3

# model identification
fitted.model <- km(~1, design=design.fact, response=response.branin,
               covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

# maximization of qEI

# With a multistarted BFGS algorithm
maxBFGS <- max_qEI(model = fitted.model, npoints = batchSize, lower = lower, upper = upper,
crit = "exact",optimcontrol=list(nStarts=3,method = "BFGS"))
# With a genetic algorithme using derivatives
```

```
maxGen  <- max_qEI(model = fitted.model, npoints = batchSize, lower = lower, upper = upper,
crit = "exact", optimcontrol=list(nStarts=3,method = "genoud",pop.size=100,max.generations = 15))
# With the constant liar heuristic
maxCL   <- max_qEI(model = fitted.model, npoints = batchSize, lower = lower, upper = upper,
crit = "CL",optimcontrol=list(pop.size=20))

# comparison
print(maxBFGS$value)
print(maxGen$value)
print(maxCL$value)


## End(Not run)
```

---

min_quantile *Minimization of the Kriging quantile.*

Description

Minimization, based on the package rgenoud of the kriging quantile.

Usage

```
min_quantile(model, beta = 0.1, type = "UK", lower, upper, parinit = NULL,
  control = NULL)
```

Arguments

model	a Kriging model of "km" class
beta	Quantile level (default value is 0.1)
type	Kriging type: "SK" or "UK"
lower	vector containing the lower bounds of the variables to be optimized over
upper	vector containing the upper bounds of the variables to be optimized over
parinit	optional vector containing the initial values for the variables to be optimized over
control	optional list of control parameters for optimization. One can control "pop.size" (default : [N=3*2^dim for dim<6 and N=32*dim otherwise]), "max.generations" (12), "wait.generations" (2) and "BFGSburnin" (2) of function "genoud" (see genoud). Numbers into brackets are the default values

Value

A list with components:

par	the best set of parameters found.
value	the value of the krigign quantile at par.

Author(s)

Victor Picheny

David Ginsbourger

Examples

```
##########################################################################
###    KRIGING QUANTILE SURFACE AND OPTIMIZATION PERFORMED BY GENOUD  ####
###    FOR AN ORDINARY KRIGING MODEL                                  ####
### OF THE BRANIN FUNCTION KNOWN AT A 12-POINT LATIN HYPERCUBE DESIGN ####
##########################################################################
set.seed(10)

# Set test problem parameters
doe.size <- 10
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.2

# Generate DOE and response
doe <- as.data.frame(matrix(runif(doe.size*dim),doe.size))
y.tilde <- rep(0, 1, doe.size)
for (i in 1:doe.size)  {y.tilde[i] <- test.function(doe[i,])
+ sqrt(noise.var)*rnorm(n=1)}
y.tilde <- as.numeric(y.tilde)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
     covtype="gauss", noise.var=rep(noise.var,1,doe.size),
     lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Optimisation using max_kriging.quantile
res <- min_quantile(model, beta=0.1, type = "UK", lower=c(0,0), upper=c(1,1))
X.genoud <- res$par

# Compute actual function and criterion on a grid
n.grid <- 12 # Change to 21 for a nicer picture
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
names(design.grid) <- c("V1","V2")
nt <- nrow(design.grid)
crit.grid <- apply(design.grid, 1, kriging.quantile, model=model, beta=.1)

# # 2D plots
z.grid <- matrix(crit.grid, n.grid, n.grid)
tit <- "Green: best point found by optimizer"
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
plot.axes = {title(tit);points(model@X[,1],model@X[,2],pch=17,col="blue");
points(X.genoud[1],X.genoud[2],pch=17,col="green");
axis(1); axis(2)})
```

noisy.optimizer *Optimization of homogenously noisy functions based on Kriging*

Description

Sequential optimization of kriging-based criterion conditional on noisy observations, with model update after each evaluation. Eight criteria are proposed to choose the next observation: random search, sequential parameter optimization (SPO), reinterpolation, Expected Improvement (EI) with plugin, Expected Quantile Improvement (EQI), quantile minimization, Augmented Expected Improvement (AEI) and Approximate Knowledge Gradient (AKG). The criterion optimization is based on the package rgenoud.

Usage

```
noisy.optimizer(optim.crit, optim.param = NULL, model, n.ite,
  noise.var = NULL, funnoise, lower, upper, parinit = NULL,
  control = NULL, CovReEstimate = TRUE, NoiseReEstimate = FALSE,
  nugget.LB = 1e-05, estim.model = NULL, type = "UK")
```

Arguments

optim.crit	String defining the criterion to be optimized at each iteration. Possible values are: "random.search", "SPO", "reinterpolation", "EI.plugin", "EQI", "min.quantile", "AEI", "AKG".
optim.param	List of parameters for the chosen criterion. For "EI.plugin": optim.param$plugin.type is a string defining which plugin is to be used. Possible values are "ytilde", "quantile" and "other". If "quantile" is chosen, optim.param$quantile defines the quantile level. If "other" is chosen, optim.param$plugin directly sets the plugin value. For "EQI": optim.param$quantile defines the quantile level. If not provided, default value is 0.9. For "min.quantile": optim.param$quantile defines the quantile level. If not provided, default value is 0.1. For "AEI": optim.param$quantile defines the quantile level to choose the current best point. If not provided, default value is 0.75.
model	a Kriging model of "km" class
n.ite	Number of iterations
noise.var	Noise variance (scalar). If noiseReEstimate=TRUE, it is an initial guess for the unknown variance (used in optimization).
funnoise	objective (noisy) function
lower	vector containing the lower bounds of the variables to be optimized over
upper	vector containing the upper bounds of the variables to be optimized over
parinit	optional vector of initial values for the variables to be optimized over

`control` optional list of control parameters for optimization. One can control `"pop.size"` (default : [N=3*2^dim for dim<6 and N=32*dim otherwise]]), `"max.generations"` (N), `"wait.generations"` (2) and `"BFGSburnin"` (0) of function `"genoud"` (see genoud). Numbers into brackets are the default values

`CovReEstimate` optional boolean specfiying if the covariance parameters should be re-estimated at every iteration (default value = TRUE)

`NoiseReEstimate` optional boolean specfiying if the noise variance should be re-estimated at every iteration (default value = FALSE)

`nugget.LB` optional scalar of minimal value for the estimated noise variance. Default value is 1e-5.

`estim.model` optional kriging model of "km" class with homogeneous nugget effect (no noise.var). Required if noise variance is reestimated and the initial "model" has heterogenenous noise variances.

`type` "SK" or "UK" for Kriging with known or estimated trend

Value

A list with components:

`model` the current (last) kriging model of "km" class

`best.x` The best design found

`best.y` The objective function value at best.x

`best.index` The index of best.x in the design of experiments

`history.x` The added observations

`history.y` The added observation values

`history.hyperparam` The history of the kriging parameters

`estim.model` If noiseReEstimate=TRUE, the current (last) kriging model of "km" class for estimating the noise variance.

`history.noise.var` If noiseReEstimate=TRUE, the history of the noise variance estimate.

Author(s)

Victor Picheny

References

V. Picheny and D. Ginsbourger (2013), Noisy kriging-based optimization methods: A unified implementation within the DiceOptim package, *Computational Statistics & Data Analysis*

Examples

```
##########################################################################
### EXAMPLE 1: 3 OPTIMIZATION STEPS USING EQI WITH KNOWN NOISE         ###
### AND KNOWN COVARIANCE PARAMETERS FOR THE BRANIN FUNCTION            ###
##########################################################################

set.seed(10)
library(DiceDesign)
# Set test problem parameters
doe.size <- 9
dim <- 2
test.function <- get("branin2")
lower <- rep(0,1,dim)
upper <- rep(1,1,dim)
noise.var <- 0.1

# Build noisy simulator
funnoise <- function(x)
{     f.new <- test.function(x) + sqrt(noise.var)*rnorm(n=1)
      return(f.new)}

# Generate DOE and response
doe <- as.data.frame(lhsDesign(doe.size, dim)$design)
y.tilde <- funnoise(doe)

# Create kriging model
model <- km(y~1, design=doe, response=data.frame(y=y.tilde),
     covtype="gauss", noise.var=rep(noise.var,1,doe.size),
     lower=rep(.1,dim), upper=rep(1,dim), control=list(trace=FALSE))

# Optimisation with noisy.optimizer (n.ite can be increased)
n.ite <- 2
optim.param <- list()
optim.param$quantile <- .9
optim.result <- noisy.optimizer(optim.crit="EQI", optim.param=optim.param, model=model,
n.ite=n.ite, noise.var=noise.var, funnoise=funnoise, lower=lower, upper=upper,
NoiseReEstimate=FALSE, CovReEstimate=FALSE)

new.model <- optim.result$model
best.x    <- optim.result$best.x
new.doe   <- optim.result$history.x

## Not run:
##### DRAW RESULTS #####
# Compute actual function on a grid
n.grid <- 12
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
names(design.grid) <- c("V1","V2")
nt <- nrow(design.grid)
func.grid <- rep(0,1,nt)
```

```
for (i in 1:nt)
{ func.grid[i] <- test.function(x=design.grid[i,])}

# Compute initial and final kriging on a grid
pred <- predict(object=model, newdata=design.grid, type="UK", checkNames = FALSE)
mk.grid1 <- pred$m
sk.grid1 <- pred$sd

pred <- predict(object=new.model, newdata=design.grid, type="UK", checkNames = FALSE)
mk.grid2 <- pred$m
sk.grid2 <- pred$sd

# Plot initial kriging mean
z.grid <- matrix(mk.grid1, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
plot.axes = {title("Initial kriging mean");
points(model@X[,1],model@X[,2],pch=17,col="black");
axis(1); axis(2)})

# Plot initial kriging variance
z.grid <- matrix(sk.grid1^2, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
plot.axes = {title("Initial kriging variance");
points(model@X[,1],model@X[,2],pch=17,col="black");
axis(1); axis(2)})

# Plot final kriging mean
z.grid <- matrix(mk.grid2, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
plot.axes = {title("Final kriging mean");
points(new.model@X[,1],new.model@X[,2],pch=17,col="black");
axis(1); axis(2)})

# Plot final kriging variance
z.grid <- matrix(sk.grid2^2, n.grid, n.grid)
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
plot.axes = {title("Final kriging variance");
points(new.model@X[,1],new.model@X[,2],pch=17,col="black");
axis(1); axis(2)})

# Plot actual function and observations
z.grid <- matrix(func.grid, n.grid, n.grid)
tit <- "Actual function - Black: initial points; red: added points"
filled.contour(x.grid,y.grid, z.grid, nlevels=50, color = topo.colors,
plot.axes = {title(tit);points(model@X[,1],model@X[,2],pch=17,col="black");
points(new.doe[1,],new.doe[2,],pch=15,col="red");
axis(1); axis(2)})

## End(Not run)

##########################################################################
### EXAMPLE 2: 3 OPTIMIZATION STEPS USING EQI WITH UNKNOWN NOISE       ###
### AND UNKNOWN COVARIANCE PARAMETERS FOR THE BRANIN FUNCTION          ###
```

```
############################################################################
# Same initial model and parameters as for example 1
n.ite <- 2 # May be changed to a larger value
res <- noisy.optimizer(optim.crit="min.quantile",
optim.param=list(type="quantile",quantile=0.01),
model=model, n.ite=n.ite, noise.var=noise.var, funnoise=funnoise,
lower=lower, upper=upper,
control=list(print.level=0),CovReEstimate=TRUE, NoiseReEstimate=TRUE)

# Plot actual function and observations
plot(model@X[,1], model@X[,2], pch=17,xlim=c(0,1),ylim=c(0,1))
points(res$history.x[1,], res$history.x[2,], col="blue")

# Restart: requires the output estim.model of the previous run
# to deal with potential repetitions
res2 <- noisy.optimizer(optim.crit="min.quantile",
optim.param=list(type="quantile",quantile=0.01),
model=res$model, n.ite=n.ite, noise.var=noise.var, funnoise=funnoise,
lower=lower, upper=upper, estim.model=res$estim.model,
control=list(print.level=0),CovReEstimate=TRUE, NoiseReEstimate=TRUE)

# Plot new observations
points(res2$history.x[1,], res2$history.x[2,], col="red")
```

ParrConstraint *2D constraint function*

Description

Strongly multimdoal constraint function from Parr et al. (standardized version)

Usage

```
ParrConstraint(x)
```

Arguments

x	a 2-dimensional vector or a two-column matrix specifying the location(s) where the function is to be evaluated.

Value

A scalar

Examples

```
n.grid <- 20
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
response.grid <- apply(design.grid, 1, ParrConstraint)
```

```
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid,y.grid,z.grid,40)
title("Parr constraint function")
```

---

qEGO.nsteps *Sequential multipoint Expected improvement (qEI) maximizations and model re-estimation*

Description

Executes `nsteps` iterations of the multipoint EGO method to an object of class `km`. At each step, a kriging model (including covariance parameters) is re-estimated based on the initial design points plus the points visited during all previous iterations; then a new batch of points is obtained by maximizing the multipoint Expected Improvement criterion (`qEI`).

Usage

```
qEGO.nsteps(fun, model, npoints, nsteps, lower = rep(0, model@d),
  upper = rep(1, model@d), crit = "exact", minimization = TRUE,
  optimcontrol = NULL, cov.reestim = TRUE, ...)
```

Arguments

`fun`	the objective function to be optimized,
`model`	an object of class `km` ,
`npoints`	an integer repesenting the desired batchsize,
`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,
`crit`	"exact", "CL" : a string specifying the criterion used. "exact" triggers the maximization of the multipoint expected improvement at each iteration (see `max_qEI`), "CL" applies the Constant Liar heuristic,
`minimization`	logical specifying if we want to minimize or maximize `fun`,
`optimcontrol`	an optional list of control parameters for the qEI optimization (see details or `max_qEI`),
`cov.reestim`	optional boolean specifying if the kriging hyperparameters should be re-estimated at each iteration,
`...`	optional arguments for `fun`.

Details

The parameters of list optimcontrol are :

- optimcontrol$method : "BFGS" (default), "genoud" ; a string specifying the method used to maximize the criterion (irrelevant when crit is "CL" because this method always uses genoud),

- when crit="CL" :

+ optimcontrol$parinit : optional matrix of initial values (must have model@d columns, the number of rows is not constrained),

+ optimcontrol$L : "max", "min", "mean" or a scalar value specifying the liar ; "min" takes model@min, "max" takes model@max, "mean" takes the prediction of the model ; When L is NULL, "min" is taken if minimization==TRUE, else it is "max".

+ The parameters of function genoud. Main parameters are : "pop.size" (default : [N=3*2^model@d for dim<6 and N=32*model@d otherwise]), "max.generations" (default : 12), "wait.generations" (default : 2) and "BFGSburnin" (default : 2).

- when optimcontrol$method = "BFGS" :

+ optimcontrol$nStarts (default : 4),

+ optimcontrol$fastCompute : if TRUE (default), a fast approximation method based on a semi-analytic formula is used, see [Marmin 2014] for details,

+ optimcontrol$samplingFun : a function which sample a batch of starting point (default : sampleFromEI),

+ optimcontrol$parinit : optional 3d-array of initial (or candidate) batches (for all k, parinit[,,k] is a matrix of size npoints*model@d representing one batch). The number of initial batches (length(parinit[1,1,])) is not contrained and does not have to be equal to nStarts. If there is too few initial batches for nStarts, missing batches are drawn with samplingFun (default : NULL),

- when optimcontrol$method = "genoud" :

+ optimcontrol$fastCompute : if TRUE (default), a fast approximation method based on a semi-analytic formula is used, see [Marmin 2014] for details,

+ optimcontrol$parinit : optional matrix of candidate starting points (one row corresponds to one point),

+ The parameters of the genoud function. Main parameters are "pop.size" (default : [50*(model@d)*(npoints)]), "max.generations" (default : 5), "wait.generations" (default : 2), "BFGSburnin" (default : 2).

Value

A list with components:

par	a data frame representing the additional points visited during the algorithm,
value	a data frame representing the response values at the points given in par,
npoints	an integer representing the number of parallel computations,
nsteps	an integer representing the desired number of iterations (given in argument),
lastmodel	an object of class km corresponding to the last kriging model fitted,
history	a vector of size nsteps representing the current known optimum at each step.

Author(s)

Sebastien Marmin

Clement Chevalier

David Ginsbourger

References

C. Chevalier and D. Ginsbourger (2013), Fast computation of the multi-points expected improvement with applications in batch selection. In G. Nicosia and P. Pardalos (Eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp 59-69, Springer.

D. Ginsbourger, R. Le Riche, L. Carraro (2007), A Multipoint Criterion for Deterministic Parallel Global Optimization based on Kriging. The International Conference on Non Convex Programming, 2007.

D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization (2010), In Lim Meng Hiot, Yew Soon Ong, Yoel Tenne, and Chi-Keong Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, Adaptation Learning and Optimization, pages 131-162. Springer Berlin Heidelberg.

S. Marmin, C. Chevalier and D. Ginsbourger (2015), Differentiating the multipoint Expected Improvement for optimal batch design. In P. Pardalos et al. (Eds.) Machine Learning, Optimization, and Big Data, Lecture Notes in Computer Science 9432, pp 37-48.

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

qEI, max_qEI, qEI.grad

Examples

```
## Not run:
set.seed(123)
#####################################################
### 4 ITERATIONS OF EGO ON THE BRANIN FUNCTION,   ###
### STARTING FROM A 9-POINTS FACTORIAL DESIGN     ###
#####################################################

# a 9-points factorial design, and the corresponding response
d <- 2
n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.branin <- apply(design.fact, 1, branin)
response.branin <- data.frame(response.branin)
names(response.branin) <- "y"
```

```
# model identification
fitted.model1 <- km(~1, design=design.fact, response=response.branin,
covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

# EGO n steps
library(rgenoud)
nsteps <- 4 # Was 10, reduced to 4 for speeding up execution
lower <- rep(0,d)
upper <- rep(1,d)
npoints <- 3 # The batchsize
oEGO <- qEGO.nsteps(model = fitted.model1, branin, npoints = npoints, nsteps = nsteps,
crit="exact", lower, upper, optimcontrol = NULL)
print(oEGO$par)
print(oEGO$value)
plot(c(1:nsteps),oEGO$history,xlab='step',ylab='Current known minimum')

# graphics
n.grid <- 15 # Was 20, reduced to 15 for speeding up compilation
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
response.grid <- apply(design.grid, 1, branin)
z.grid <- matrix(response.grid, n.grid, n.grid)
contour(x.grid, y.grid, z.grid, 40)
title("Branin function")
points(design.fact[,1], design.fact[,2], pch=17, col="blue")
points(oEGO$par, pch=19, col="red")
text(oEGO$par[,1], oEGO$par[,2], labels=c(tcrossprod(rep(1,npoints),1:nsteps)), pos=3)

## End(Not run)
```

---

qEI — *Analytical expression of the multipoint expected improvement (qEI) criterion*

Description

Computes the multipoint expected improvement criterion.

Usage

```
qEI(x, model, plugin = NULL, type = "UK", minimization = TRUE,
  fastCompute = TRUE, eps = 10^(-5), envir = NULL)
```

Arguments

x	a matrix representing the set of input points (one row corresponds to one point) where to evaluate the qEI criterion,
model	an object of class km,
plugin	optional scalar: if provided, it replaces the minimum of the current observations,

`type`	"SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account,
`minimization`	logical specifying if EI is used in minimiziation or in maximization,
`fastCompute`	if TRUE, a fast approximation method based on a semi-analytic formula is used (see [Marmin 2014] for details),
`eps`	the value of *epsilon* of the fast computation trick. Relevant only if `fastComputation` is TRUE,
`envir`	an optional environment specifying where to get intermediate values calculated in `qEI`.

Value

The multipoint Expected Improvement, defined as

$$qEI(X_{new}) := E[(min(Y(X)) - min(Y(X_{new})))_+ | Y(X) = y(X)],$$

where X is the current design of experiments, X_{new} is a new candidate design, and Y is a random process assumed to have generated the objective function y.

Author(s)

Sebastien Marmin

Clement Chevalier

David Ginsbourger

References

C. Chevalier and D. Ginsbourger (2013), Fast computation of the multi-points expected improvement with applications in batch selection. In G. Nicosia and P. Pardalos (Eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp 59-69, Springer.

D. Ginsbourger, R. Le Riche, L. Carraro (2007), A Multipoint Criterion for Deterministic Parallel Global Optimization based on Kriging. The International Conference on Non Convex Programming, 2007.

S. Marmin, C. Chevalier and D. Ginsbourger (2015), Differentiating the multipoint Expected Improvement for optimal batch design. In P. Pardalos et al. (Eds.) Machine Learning, Optimization, and Big Data, Lecture Notes in Computer Science 9432, pp 37-48.

D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization (2010), In Lim Meng Hiot, Yew Soon Ong, Yoel Tenne, and Chi-Keong Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, Adaptation Learning and Optimization, pages 131-162. Springer Berlin Heidelberg.

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

`EI`

Examples

```
set.seed(007)

# Monte-Carlo validation

# a 4-d, 81-points grid design, and the corresponding response
d <- 4; n <- 3^d
design <- do.call(expand.grid,rep(list(seq(0,1,length=3)),d))
names(design) <- paste("x",1:d,sep="")
y <- data.frame(apply(design, 1, hartman4))
names(y) <- "y"

# learning
model <- km(~1, design=design, response=y, control=list(trace=FALSE))

# pick up 10 points sampled from the 1-point expected improvement
q <- 10
X <- sampleFromEI(model,n=q)
# simulation of the minimum of the kriging random vector at X
t1 <- proc.time()
newdata <- as.data.frame(X)
colnames(newdata) <- colnames(model@X)

krig  <- predict(object=model, newdata=newdata,type="UK",se.compute=TRUE, cov.compute=TRUE)
mk <- krig$mean
Sigma.q <- krig$cov
mychol <- chol(Sigma.q)
nsim <- 300000
white.noise <- rnorm(n=nsim*q)
minYsim <- apply(crossprod(mychol,matrix(white.noise,nrow=q)) + mk,2,min)
# simulation of the improvement (minimization)
qImprovement <- (min(model@y)-minYsim)*((min(model@y)-minYsim) > 0)

# empirical expectation of the improvement and confident interval (95%)
eiMC <- mean(qImprovement)
confInterv <- c(eiMC - 1.96*sd(qImprovement)*1/sqrt(nsim),eiMC + 1.96*sd(qImprovement)*1/sqrt(nsim))

# MC estimation of the qEI
print(eiMC)
t2 <- proc.time()
# qEI with analytical formula
qEI(X,model,fastCompute= FALSE)
t3 <- proc.time()
# qEI with fast computation trick
qEI(X,model)
t4 <- proc.time()
t2-t1  # Time of MC computation
t3-t2  # Time of normal computation
t4-t3  # Time of fast computation
```

qEI.grad *Gradient of the multipoint expected improvement (qEI) criterion*

Description

Computes an exact or approximate gradient of the multipoint expected improvement criterion

Usage

```
qEI.grad(x, model, plugin = NULL, type = "UK", minimization = TRUE,
  fastCompute = TRUE, eps = 10^(-6), envir = NULL)
```

Arguments

x	a matrix representing the set of input points (one row corresponds to one point) where to evaluate the gradient,
model	an object of class km,
plugin	optional scalar: if provided, it replaces the minimum of the current observations,
type	"SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account,
minimization	logical specifying if EI is used in minimiziation or in maximization,
fastCompute	if TRUE, a fast approximation method based on a semi-analytic formula is used (see [Marmin 2014] for details),
eps	the value of *epsilon* of the fast computation trick. Relevant only if fastComputation is TRUE,
envir	an optional environment specifying where to get intermediate values calculated in qEI.

Value

The gradient of the multipoint expected improvement criterion with respect to x. A 0-matrix is returned if the batch of input points contains twice the same point or a point from the design experiment of the km object (the gradient does not exist in these cases).

Author(s)

Sebastien Marmin

Clement Chevalier

David Ginsbourger

References

C. Chevalier and D. Ginsbourger (2013), Fast computation of the multi-points expected improvement with applications in batch selection. In G. Nicosia and P. Pardalos (Eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp 59-69, Springer.

D. Ginsbourger, R. Le Riche, L. Carraro (2007), A Multipoint Criterion for Deterministic Parallel Global Optimization based on Kriging. The International Conference on Non Convex Programming, 2007.

D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization (2010), In Lim Meng Hiot, Yew Soon Ong, Yoel Tenne, and Chi-Keong Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, Adaptation Learning and Optimization, pages 131-162. Springer Berlin Heidelberg.

S. Marmin, C. Chevalier and D. Ginsbourger (2015), Differentiating the multipoint Expected Improvement for optimal batch design. In P. Pardalos et al. (Eds.) Machine Learning, Optimization, and Big Data, Lecture Notes in Computer Science 9432, pp 37-48.

J. Mockus (1988), *Bayesian Approach to Global Optimization*. Kluwer academic publishers.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

qEI

Examples

```
## Not run:
set.seed(15)
# Example 1 - validation by comparison to finite difference approximations

# a 10-points optimum LHS design and the corresponding responses
d <- 2;n <- d*5
design <- maximinESE_LHS(lhsDesign(n,d)$design,1)$design
colnames(design)<-c("x1", "x2")
lower <- c(0,0)
upper <- c(1,1)
y <- data.frame(apply(design, 1, branin))
names(y) <- "y"

# learning
model <- km(~1, design=design, response=y)

# pick up 4 points sampled from the simple expected improvement
q <- 4
X <- sampleFromEI(model,n=q)

# compute the gradient at the 4-point batch
grad.analytic <- qEI.grad(X,model)
# numerically compute the gradient
grad.numeric <- matrix(NaN,q,d)
eps <- 10^(-6)
```

```
EPS <- matrix(0,q,d)
for (i in 1:q) {
  for (j in 1:d) {
    EPS[i,j] <- eps
   grad.numeric[i,j] <- 1/eps*(qEI(X+EPS,model,fastCompute=FALSE)-qEI(X,model,fastCompute=FALSE))
    EPS[i,j] <- 0
  }
}
print(grad.numeric)
print(grad.analytic)

# graphics: displays the EI criterion, the design points in black,
# the batch points in red and the gradient in blue.
nGrid <- 15
gridAxe1 <- seq(lower[1],upper[1],length=nGrid)
gridAxe2 <- seq(lower[2],upper[2],length=nGrid)
grid <- expand.grid(gridAxe1,gridAxe2)
aa <- apply(grid,1,EI,model=model)
myMat <- matrix(aa,nrow=nGrid)
image(x = gridAxe1, y = gridAxe2, z = myMat,
      col = colorRampPalette(c("darkgray","white"))(5*10),
      ylab = names(design)[1], xlab=names(design)[2],
      main = "qEI-gradient of a batch of 4 points", axes = TRUE,
      zlim = c(min(myMat), max(myMat)))
contour(x = gridAxe1, y = gridAxe2, z = myMat,
        add = TRUE, nlevels = 10)
points(X[,1],X[,2],pch=19,col='red')
points(model@X[,1],model@X[,2],pch=19)
arrows(X[,1],X[,2],X[,1]+0.012*grad.analytic[,1],X[,2]+0.012*grad.analytic[,2],col='blue')

## End(Not run)
```

`sampleFromEI` *Sampling points according to the expected improvement criterion*

Description

Samples n points from a distribution proportional to the expected improvement (EI) computed from a km object.

Usage

```
sampleFromEI(model, minimization = TRUE, n = 1, initdistrib = NULL,
  lower = rep(0, model@d), upper = rep(1, model@d), T = NULL)
```

Arguments

`model`	an object of class km,
`minimization`	logical specifying if EI is used in minimiziation or in maximization,

n	number of points to be sampled,
initdistrib	matrix of candidate points.
lower	vector of lower bounds,
upper	vector of upper bounds,
T	optional scalar : if provided, it replaces the current minimum (or maximum) of observations.

Value

A n*d matrix containing the sampled points. If NULL, 1000*d points are obtained by latin hypercube sampling,

Author(s)

Sebastien Marmin

Clement Chevalier

David Ginsbourger

References

D.R. Jones, M. Schonlau, and W.J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13, 455-492.

See Also

EI, km, qEI

Examples

```
set.seed(004)

# a 9-points factorial design, and the corresponding responses
d <- 2
n <- 9
design.fact <- expand.grid(seq(0,1,length=3), seq(0,1,length=3))
names(design.fact)<-c("x1", "x2")
design.fact <- data.frame(design.fact)
names(design.fact)<-c("x1", "x2")
response.branin <- apply(design.fact, 1, branin)
response.branin <- data.frame(response.branin)
lower <- c(0,0)
upper <- c(1,1)
names(response.branin) <- "y"


# model identification
fitted.model <- km(~1, design=design.fact, response=response.branin,
```

```
                covtype="gauss", control=list(pop.size=50,trace=FALSE), parinit=c(0.5, 0.5))

# sample a 30 point batch
batchSize <- 30
x <- sampleFromEI(model = fitted.model, n = batchSize, lower = lower, upper = upper)

# graphics
# displays the EI criterion, the design points in black and the EI-sampled points in red.
nGrid <- 15
gridAxe1 <- seq(lower[1],upper[1],length=nGrid)
gridAxe2 <- seq(lower[2],upper[2],length=nGrid)
grid <- expand.grid(gridAxe1,gridAxe2)
aa <- apply(grid,1,EI,model=fitted.model)
myMat <- matrix(aa,nrow=nGrid)
image(x = gridAxe1, y = gridAxe2, z = myMat,
      col = colorRampPalette(c("darkgray","white"))(5*10),
      ylab = names(design.fact)[1], xlab=names(design.fact)[2],
      main = "Sampling from the expected improvement criterion",
      axes = TRUE, zlim = c(min(myMat), max(myMat)))
contour(x = gridAxe1, y = gridAxe2, z = myMat,
        add = TRUE, nlevels = 10)
points(x[,1],x[,2],pch=19,col='red')
points(fitted.model@X[,1],fitted.model@X[,2],pch=19)
```

`test_feas_vec` *Test constraints violation (vectorized)*

Description

Test whether a set of constraints are violated or not, depending on their nature (equality or inequality) and tolerance parameters

Usage

```
test_feas_vec(cst, equality = FALSE, tolConstraints = NULL)
```

Arguments

`cst`	matrix of constraints (one column for each constraint function)
`equality`	either FALSE or a Boolean vector defining which constraints are treated as equalities
`tolConstraints`	tolerance (vector) for all constraints. If not provided, set to zero for inequalities and 0.05 for equalities

Value

A Boolean vector, TRUE if the point if feasible, FALSE if at least one constraint is violated

Author(s)

Mickael Binois

Victor Picheny

`update_km_noisyEGO` *Update of one or two Kriging models when adding new observation*

Description

Update of a noisy Kriging model when adding new observation, with or without covariance parameter re-estimation. When the noise level is unkown, a twin model "estim.model" is also updated.

Usage

```
update_km_noisyEGO(model, x.new, y.new, noise.var = 0, type = "UK",
  add.obs = TRUE, index.in.DOE = NULL, CovReEstimate = TRUE,
  NoiseReEstimate = FALSE, estim.model = NULL, nugget.LB = 1e-05)
```

Arguments

`model`	a Kriging model of "km" class
`x.new`	a matrix containing the new points of experiments
`y.new`	a matrix containing the function values on the points NewX
`noise.var`	scalar: noise variance
`type`	kriging type: "SK" or "UK"
`add.obs`	boolean: if TRUE, the new point does not exist already in the design of experiment model@X
`index.in.DOE`	optional integer: if add.obs=TRUE, it specifies the index of the observation in model@X corresponding to x.new
`CovReEstimate`	optional boolean specfiying if the covariance parameters should be re-estimated (default value = TRUE)
`NoiseReEstimate`	optional boolean specfiying if the noise variance should be re-estimated (default value = TRUE)
`estim.model`	optional input of "km" class. Required if NoiseReEstimate=TRUE, in order to deal with repetitions.
`nugget.LB`	optional scalar: is used to define a lower bound on the noise variance.

Value

A list containing:

`model`	The updated Kriging model
`estim.model`	If NoiseReEstimate=TRUE, the updated estim.model
`noise.var`	If NoiseReEstimate=TRUE, the re-estimated noise variance

Author(s)

Victor Picheny

References

V. Picheny and D. Ginsbourger (2013), Noisy kriging-based optimization methods: A unified implementation within the DiceOptim package, *Computational Statistics & Data Analysis*

Index