Package ‘moko’

July 21, 2016

Type Package

Title MOKO: Multi-Objective Kriging Optimization

Version 0.0.0.9000

Description The package moko provides the user with methods for constrained and unconstraiend multiobjective optimization. Those methods are all based on the Kriging surogate modeling technique.

Depends R (>= 3.0.0)

License GPL-3

LazyData TRUE

Imports DiceKriging (>= 1.5.5), DiceOptim (>= 1.5), GenSA (>= 1.1.6), emoa (>= 0.5.0), mco (>= 1.0.15.1), GPareto (>= 1.0.2), methods (>= 3.0.0)

RoxygenNote 5.0.1

URL http://github.com/coldfir3/moko

BugReports http://github.com/coldfir3/moko/issues

Suggests testthat

Author Adriano Passos [aut, cre]

Maintainer Adriano Passos <adriano.utfpr@gmail.com>

R topics documented:

EHVI *EHVI: Constrained Expected Hypervolume Improvement*

Description

Multi-objective Expected Hypervolume Improvement with respect to the current Pareto front. It's based on the `crit_EHI` function of the `GPareto` package. However, the present implementation accounts for inequalty constrains embeded into a `mkm` model.

Usage

```
EHVI(x, model, control = NULL)
```

Arguments

`x`	a vector representing the input for which one wishes to calculate `EHI`,
`model`	An object of class `mkm`.
`control`	An optional list of control parameters, some of them passed to the `crit_EHI` function. One can control: `minimization` logical indicating if the EHVI is minimizing all objectives (`TRUE`, by default) or maximizing all objectives (`FALSE`). Mixed information not currently accepted, if the user needs so, it should invert the functions prior Kriging modeling `paretoFront` object of class `ps` containing the actual Pareto set. If not provided a Pareto set is built based on the current observations of `model`. `nb.samp` default: 50 `seed` default: 42 `refPoint` default: `min` or `max` values for the Pareto front for each objective

Value

The Constrained Expected Hypervolume Improvement at `x`.

Examples

```
# ------------------------
# The Nowacki Beam
# ------------------------
n <- 50
d <- 2
doe <- replicate(d,sample(0:n,n))/n
res <- t(apply(doe, 1, nowacki_beam))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower=rep(0.1,d)))
```

```
grid <- expand.grid(seq(0, 1, , 50),seq(0, 1, , 50))
### this computation may take some time ###
ehvi <- apply(grid, 1, EHVI, model)
contour(matrix(ehvi, 50))
points(model@design, col=ifelse(model@feasible,'blue','red'))
points(grid[which.max(ehvi),], col='green', pch=19)
```

EI *Constrained Expected Emprovement*

Description

This functions extends the EI function suplyied by the package DiceOptim. The enxtension allows to the usage of multiple expensive constraints. The constraints must be passed to the EI function embedded in the mkm object. Currently low-cost (explicit) constraints are not allowed.

Usage

```
EI(x, model, control = NULL)
```

Arguments

x
: A vector representing the input for which one wishes to calculate EI.

model
: An object of class mkm.

control
: An optional list of control parameters, some of them passed to the EI function. One can control:

  `minimization` logical specifying if EI is used in minimiziation or in maximization (default: TRUE)

  `plugin` optional scalar, if not provided, the minimum (or maximum) of the current feasible observations. If there isn't any feasible design plugin is set to `NA` and the algorithm returns the value of the probabilty of constraints be met.

  `envir` optional enviroment specifying where to assign intermediate values. Default is NULL.

Details

The way that the constraints are handled are based on the probability of feasibility. The strong assumption here is that the cost functions and the constraints are uncorrelated. With that assumption in mind, a simple closed-form solution can be derived that consists in the product of the probability that each constraint will be met and the expected improvemen of the objective. Another important consideration is that, by default, the value of the pluging passed to the EI is the best *feasible* observed value.

References

Forrester, A., Sobester, A., & Keane, A. (2008). *Engineering design via surrogate modelling: a practical guide.* John Wiley & Sons.

Examples

```
# --------------------------------------------
# Branin-Hoo function (with simple constraint)
# --------------------------------------------
n <- 10
d <- 2
doe <- replicate(d,sample(0:n,n))/n
fun_cost <- DiceKriging::branin
fun_cntr <- function(x) 0.2 - prod(x)
fun <- function(x) return(cbind(fun_cost(x),fun_cntr(x)))
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1, lower=c(0.1,0.1)))
grid <- expand.grid(seq(0,1,,50),seq(0,1,,50))
ei <- apply(grid, 1, EI, model) # this computation may take some time
contour(matrix(ei,50))
points(model@design, col=ifelse(model@feasible,'blue','red'))
points(grid[which.max(ei),], col='green')
```

HEGO *HEGO: Efficient Global Optimization Algorithm based on the Hypervolume criteria*

Description

Executes `nsteps` iterations of the HEGO method to an object of class `mkm`. 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 Hypervolume Improvement criterion (EHVI).

Usage

```
HEGO(model, fun, nsteps, lower = rep(0, model@d), upper = rep(1, model@d),
  quiet = TRUE, control = NULL, optimcontrol = NULL)
```

Arguments

`model`	An object of class `mkm`.
`fun`	The multi-objective and constraint cost function to be optimized. This function must return a vector with the size of `model@m + model@j` where `model@m` are the number of objectives and `model@j` the number of the constraints,
`nsteps`	An integer representing the desired number of iterations,
`lower`	Vector of lower bounds for the variables to be optimized over (default: 0 with length `model@d`),
`upper`	Vector of upper bounds for the variables to be optimized over (default: 1 with length `model@d`),
`quiet`	Logical indicating the verbosity of the routine,
`control`	An optional list of control parameters, some of them passed to the `crit_EHI` function. One can control:

`minimization` logical indicating if the EHVI is minimizing all objectives (`TRUE`, by default) or maximizing all objectives (`FALSE`). Mixed information not currently accepted, if the user needs so, it should invert the functions prior Kriging modeling

`paretoFront` object of class ps containing the actual Pareto set. If not provided a Pareto set is built based on the current observations of `model`.

`nb.samp` default: 50

`seed` default: 42

`refPoint` default: `min` or `max` values for the Pareto front for each objective

optimcontrol	Optional list of control parameters passed to the GenSA function. Please, note that the values are passed as the `control` parameter inside the `GenSA` function.

Value

updated mkm model

Examples

```
# ----------------
# The Nowacki Beam
# ----------------
n <- 20
d <- 2
fun <- nowacki_beam
doe <- replicate(d,sample(0:n,n))/n
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower = rep(0.1,d)))
model <- HEGO(model, fun, 20, quiet = FALSE, control = list(rho = 0.1))
plot(nowacki_beam_tps$set)
points(ps(model@response[which(model@feasible),model@objective])$set, col = 'green', pch = 19)
```

---

igd — *IGD: Inverted Generational Distance*

Description

The IGD is a perfomance measure function of Pareto front fidelity and corresponds to the average distance between all designs in the true set and the closest design of the current set. Thus, the lower the IGD value, the better the front is.

Usage

```
igd(aps, tps, method = "manhattan", norm = TRUE)
```

Arguments

aps	'ps' object containing the 'actual' pareto front
tps	'ps' object containing the 'true' pareto front
method	the distance measure to be used. This must be one of "euclidean" or "manhattan" (default).
norm	Logical indicating if the fronts should be normalized (default = TRUE).

Value

returns the IGD metric

References

Shimoyama, K., Jeong, S., & Obayashi, S. (2013, June). Kriging-surrogate-based optimization considering expected hypervolume improvement in non-constrained many-objective test problems. In 2013 IEEE *Congress on Evolutionary Computation* (pp. 658-665). IEEE.

Examples

```
aps <- ps(matrix(rnorm(1:1000),ncol=2))
tps <- ps(matrix(rnorm(1:2000),ncol=2))
igd(aps,tps)
```

---

max_EHVI — *max_EHVI: Maximization of the Expected Hypervolume Improvement criterion*

Description

Given an object of class mkm and a set of tuning parameters, max_EHVI performs the maximization of the Expected Hypervolume Improvement criterion and delivers the next point to be visited in an HEGO-like procedure.

Usage

```
max_EHVI(model, lower = rep(0, model@d), upper = rep(1, model@d),
  control = NULL, optimcontrol = NULL)
```

Arguments

`model`	An object of class `mkm`.
`lower`	Vector of lower bounds for the variables to be optimized over (default: 0 with length `model@d`),
`upper`	Vector of upper bounds for the variables to be optimized over (default: 1 with length `model@d`),
`control`	An optional list of control parameters, some of them passed to the `crit_EHI` function. One can control: `minimization` logical indicating if the EHVI is minimizing all objectives (`TRUE`, by default) or maximizing all objectives (`FALSE`). Mixed information not currently accepted, if the user needs so, it should invert the functions prior Kriging modeling `paretoFront` object of class `ps` containing the actual Pareto set. If not provided a Pareto set is built based on the current observations of `model`. `nb.samp` default: 50 `seed` default: 42 `refPoint` default: `min` or `max` values for the Pareto front for each objective
`optimcontrol`	Optional list of control parameters passed to the `GenSA` function. Please, note that the values are passed as the `control` parameter inside the `GenSA` function.

Value

A list with components:

`par` The best set of parameters found.

`value` The value of expected hypervolume improvement at par.

Examples

```
# ------------------------
# The Nowacki Beam
# ------------------------
n <- 20
d <- 2
doe <- replicate(d,sample(0:n,n))/n
res <- t(apply(doe, 1, nowacki_beam))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower=c(0.1,0.1)))
max_EHVI(model)
```

`max_EI` *max_EI: Maximization of the Constrained Expected Improvement criterion*

Description

Given an object of class `mkm` and a set of tuning parameters, max_EI performs the maximization of the Constrained Expected Improvement criterion and delivers the next point to be visited in an MEGO-like procedure.

Usage

```
max_EI(model, lower = rep(0, model@d), upper = rep(1, model@d),
  control = NULL, optimcontrol = NULL)
```

Arguments

`model`	An object of class `mkm`.
`lower`	Vector of lower bounds for the variables to be optimized over (default: 0 with length `model@d`),
`upper`	Vector of upper bounds for the variables to be optimized over (default: 1 with length `model@d`),
`control`	An optional list of control parameters, some of them passed to the `EI` function. One can control: `minimization` logical specifying if EI is used in minimiziation or in maximization (default: TRUE) `plugin` optional scalar, if not provided, the minimum (or maximum) of the current feasible observations. If there isn't any feasible design plugin is set to `NA` and the algorithm returns the value of the probabilty of constraints be met. `envir` optional enviroment specifying where to assign intermediate values. Default is NULL.
`optimcontrol`	Optional list of control parameters passed to the `GenSA` function. Please, note that the values are passed as the `control` parameter inside the `GenSA` function.

Value

A list with components:

`par` The best set of parameters found.

`value` The value of expected hypervolume improvement at par.

Vector. The best set of parameters found.

Examples

```
# ------------------------------------------
# Branin-Hoo function (with simple constraint)
# ------------------------------------------
n <- 10
d <- 2
doe <- replicate(d,sample(0:n,n))/n
fun_cost <- DiceKriging::branin
fun_cntr <- function(x) 0.2 - prod(x)
fun <- function(x) return(cbind(fun_cost(x),fun_cntr(x)))
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1, lower=c(0.1,0.1)))
max_EI(model)
```

`MEGO` *MEGO: Multi-Objective Efficient Global Optimization Algorithm based on scalarization of the objectives*

Description

Executes `nsteps` iterations of the MEGO method to an object of class `mkm`. At each step, a weighted 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 Constrained Expected Improvement criterion (EI).

Usage

```
MEGO(model, fun, nsteps, lower = rep(0, model@d), upper = rep(1, model@d),
  quiet = TRUE, control = NULL, optimcontrol = NULL)
```

Arguments

`model`	An object of class `mkm`.
`fun`	The multi-objective and constraint cost function to be optimized. This function must return a vector with the size of `model@m + model@j` where `model@m` are the number of objectives and `model@j` the number of the constraints,
`nsteps`	An integer representing the desired number of iterations,
`lower`	Vector of lower bounds for the variables to be optimized over (default: 0 with length `model@d`),
`upper`	Vector of upper bounds for the variables to be optimized over (default: 1 with length `model@d`),
`quiet`	Logical indicating the verbosity of the routine,

`control`
: An optional list of control parameters, some of them passed to the `EI` function. One can control:

  `minimization` logical specifying if EI is used in minimiziation or in maximization (default: TRUE)

  `plugin` optional scalar, if not provided, the minimum (or maximum) of the current feasible observations. If there isn't any feasible design plugin is set to `NA` and the algorithm returns the value of the probabilty of constraints be met.

  `envir` optional enviroment specifying where to assign intermediate values. Default is NULL.

`optimcontrol`
: Optional list of control parameters passed to the `GenSA` function. Please, note that the values are passed as the `control` parameter inside the `GenSA` function.

Value

updated `mkm` model

References

Knowles, J. (2006). ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1), 50-66.

Examples

```
# ----------------
# The Nowacki Beam
# ----------------
n <- 20
d <- 2
fun <- nowacki_beam
doe <- replicate(d,sample(0:n,n))/n
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower = rep(0.1,d)))
model <- MEGO(model, fun, 20, quiet = FALSE, control = list(rho = 0.1))
plot(nowacki_beam_tps$set)
points(ps(model@response[which(model@feasible),model@objective])$set, col = 'green', pch = 19)

############################################
#### some single objective optimization ####
############################################

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, DiceKriging::branin)
z.grid <- matrix(response.grid, n.grid, n.grid)

# ---------------------------------------
# Branin-Hoo function (unconstrained)
# ---------------------------------------
n <- 10
d <- 2
doe <- replicate(d,sample(0:n,n))/n
fun <- DiceKriging::branin
```

```
res <- apply(doe, 1, fun)
model <- mkm(doe, res, modelcontrol = list(lower=rep(0.1,d)))
model <- MEGO(model, fun, 20, quiet = FALSE)
contour(x.grid,y.grid,z.grid,40)
points(model@design, col=ifelse(model@feasible,'blue','red'))
# -----------------------------------------
# Branin-Hoo function (simple constraint)
# -----------------------------------------
n <- 10
d <- 2
doe <- replicate(d,sample(0:n,n))/n
fun_cost <- DiceKriging::branin
fun_cntr <- function(x) 0.2 - prod(x)
fun <- function(x) return(c(fun_cost(x),fun_cntr(x)))
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1, lower=rep(0.1,d)))
model <- MEGO(model, fun, 10, quiet = FALSE)
contour(x.grid,y.grid,z.grid,40)
points(model@design, col=ifelse(model@feasible,'blue','red'))
# -----------------------------------------
# Branin-Hoo function (narrow constraint)
# -----------------------------------------
n <- 10
d <- 2
doe <- replicate(d,sample(0:n,n))/n
fun_cost <- DiceKriging::branin
fun_cntr <- function(x){
 g1 <- 0.9 - sum(x)
 g2 <- sum(x) - 1.1
 g3 <- - x[1] + 0.75
 g4 <- x[2] - 0.25
 return(c(g1,g2,g3,g4))
}
fun <- function(x) return(c(fun_cost(x),fun_cntr(x)))
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1, lower=rep(0.1,d)))
model <- MEGO(model, fun, 10, quiet = FALSE)
contour(x.grid,y.grid,z.grid,40)
points(model@design, col=ifelse(model@feasible,'blue','red'))
# -----------------------------------------------
# Branin-Hoo function (disconnected constraint)
# -----------------------------------------------
n <- 10
d <- 2
doe <- replicate(d,sample(0:n,n))/n
Griewank <-  function(x) {
 ii <- c(1:length(x))
  sum <- sum(x^2/4000)
  prod <- prod(cos(x/sqrt(ii)))
  y <- sum - prod + 1
  return(y)
}
fun_cost <- DiceKriging::branin
fun_cntr <- function(x) 1.6 - Griewank(x*10-5)
fun <- function(x) return(c(fun_cost(x),fun_cntr(x)))
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1, lower=c(0.1,0.1)))
```

```
model <- MEGO(model, fun, 20, quiet = FALSE)
contour(x.grid,y.grid,z.grid,40)
points(model@design, col=ifelse(model@feasible,'blue','red'))
```

---

mkm *Multi-objective Kriging model*

Description

This function creates a multi-objective kriging model. It is based on the km function of the DiceKriging package and creates a structured list of km objects.

Usage

```
mkm(design, response, modelcontrol = NULL)
```

Arguments

design	Numeric data.frame of the designs (decision space)
response	Numeric data.frame of the observed responses (objectives and constraints) at each design point.
modelcontrol	An optional list of control parameters passed to the km function. One can control: objective default: 1:ncol(response) quiet default: TRUE formula default: ~1 covtype default: "matern5_2" nugget.estim default: FALSE estim.method default: "MLE" optim.method default: "BFGS" multistart default: 1 gr default: TRUE iso default: FALSE scaling default: FALSE type default: 'UK' se.compute default: TRUE light.return default: TRUE bias.correct default: FALSE checkNames default: FALSE For more details, one can check km.

Value

S4 An object of class mkm-class

Examples

```
# ------------------------
# The Nowacki Beam
# ------------------------
n <- 10
d <- 2
doe <- replicate(d,sample(0:n,n))/n
res <- t(apply(doe, 1, nowacki_beam))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2))
```

---

mkm-class *An S4 class of multiple Kriging models*

Description

An S4 class of multiple Kriging models

Usage

```
## S4 method for signature 'mkm'
show(object)
```

Arguments

object A mkm object.

Methods (by generic)

- show: Custom print for mkm objects

Slots

km A list of km objectives.

objective A Numeric vector representing the index of the objective models in km.

design Numeric data.frame of the designs (decision space).

d,n,m,j Numeric values for the number of dimensions, designs, objectives and constraints, respectively.

response Numeric data.frame of the observed responses (objectives and constraints) at each design point.

feasible Logical vector stating which designs are feasible.

control A list of controls for function backtracking, this list contains all the input parameters that are passed to the km function.

moko *moko: Multi-objective Kriging Optimization*

Description

The package moko provides the user with methods for constrained and unconstraiend multi-objective optimization based on the popular Kriging surrogate model.

Details

The main functions provided by moko are: MEGO, HEGO and VMPF.

nowacki_beam *Test function: The Nowacki Beam*

Description

This function is a variation of the classic multi-objective optimization problem (NOWACKI, 1980). In this problem the aim is to design a tip loaded cantilever eam for minimum cross-sectional aera and lowest bending stress subject to a number of constraints.

Usage

```
nowacki_beam(x, g = c(5, 240, 120, 10), l = 1500, F = 5000, E = 216620,
  G = 86650, v = 0.27, box = data.frame(b = c(10, 50), h = c(20, 250)))
```

Arguments

x	vector of length 2 correspon the normalized beath and height of the beam
g	vector of lenght 4 containing the limit of each constraint
l	numeric length of the beam
F	numeric force applied at the beam tip
E	numeric elastic longitudinal moduli
G	numeric elastic transversal moduli
v	numeric poison ratio
box	data.frame structure containing the upper and lower limits for b and h

Value

vector of objective and constrain responses

References

Forrester, A., Sobester, A., & Keane, A. (2008). *Engineering design via surrogate modelling: a practical guide.* John Wiley & Sons.

Examples

```
nowacki_beam(c(0.5,0.5))
```

`nowacki_beam_tps` *True pareto front for the nowacki beam problem*

Description

True pareto front for the nowacki beam problem

Usage

```
nowacki_beam_tps
```

Format

An object of class `ps` of length 4.

`pdist` *Distance betwen vector and matrix*

Description

This function computes and returns the minimum distance between a vector and a matrix

Usage

```
pdist(point, set, method = "manhattan")
```

Arguments

`point`	numeric vector
`set`	numeric matrix
`method`	the distance measure to be used. This must be one of "euclidean" or "manhattan" (default).

Value

numeric value indicating the minimum distance between point and set predict,mkm-method *Predictor for a multiobjective Kriging model*

Description

This functions performs predictions for a given dataset into a collection of Kriging models (mkm object)

Usage

```
## S4 method for signature 'mkm'
predict(object, newdata, modelcontrol = NULL)
```

Arguments

object	An object of class mkm
newdata	a vector, matrix or data frame containing the points where to perform predictions.
modelcontrol	An optional list of control parameters to the mkm function (default: object@control).

Examples

```
# ------------------------
# The Nowacki Beam
# ------------------------
n <- 100
d <- 2
N <- 50
doe <- replicate(d,sample(0:n,n))/n
res <- t(apply(doe, 1, nowacki_beam))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower = rep(0.01, d)))
newx <- expand.grid(replicate(d,seq(0,1,,N),FALSE))
pred <- predict(model, newx)
realv <- t(apply(newx, 1, nowacki_beam))
par(mfrow=c(2,3), mar=c(2,2,1,1))
for (i in 1:6){
  contour(matrix((realv[,i]),N), col='red', lty=2, labels='')
  contour(matrix((pred$mean[,i]),N), add = TRUE)
}
```

predict_front *Predicted Pareto front*

Description

This function creates a predicted pareto front based on the mean of Kriging models. The predicted mean of each objective and constraint is passed to the nsga2R algorithm that.

Usage

```
predict_front(model, lower, upper, control = NULL, modelcontrol = NULL)
```

Arguments

- `model` — Object of class `mkm`.
- `lower` — Vector of lower bounds for the variables to be optimized over (default: 0 with length `model@d`).
- `upper` — Vector of upper bounds for the variables to be optimized over (default: 1 with length `model@d`).
- `control` — An optional list of control parameters that controlls the optimization algorithm. One can control:

  `popsize` (default: `200`);

  `generations` (default: `30`);

  `cdist` (default: `1/model@d`);

  `mprob` (default: `15`);

  `mdist` (defult: `20`).
- `modelcontrol` — An optional list of control parameters to the `mkm` function (default: `object@control`).

Value

object of class `ps` containing the predicted Pareto front

Examples

```
# ------------------------
# The Nowacki Beam
# ------------------------
n <- 100
doe <- cbind(sample(0:n,n),sample(0:n,n))/n
res <- t(apply(doe, 1, nowacki_beam))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower=c(0.1,0.1)))
pf <- predict_front(model, c(0,0), c(1,1))
plot(nowacki_beam_tps$set)
points(pf$set, col='blue')
```

`ps` — *Creates a pareto set from given data*

Description

Return those points which are not dominated by another point in `y` This is the Pareto front approximation of the design set.

Usage

```
ps(y, minimization = TRUE, light.return = FALSE)
```

Arguments

- `y` — design space data
- `minimization` — logical representing if the set is to be minimized or not
- `light.return` — logical indicating if the indexes should be writen on the 'ps' object

Value

S3 class object that contains information of the Pareto set

Examples

```
aps <- ps(matrix(rnorm(1:1000),ncol=2))
print(aps)
```

`radviz` *Plot a multiresponse or multivariate dataset indo a 2d radViz graph*

Description

Description

Usage

```
radviz(data, ...)
```

Arguments

`data`	data.frame containing the variables or observations to be ploted
`...`	opitional plotting argumentos passed to `points` function.

Examples

```
data <- data.frame(matrix(rnorm(1:50),ncol=5))
radviz(data, col='red')
```

`Tchebycheff` *Augmented Tchebycheff function*

Description

The Augmented Tchebycheff function (KNOWLES, 2006) is a scalarizing function witch the advantages of having a non-linear term. That causes points on nonconvex regions of the Pareto front can bve minimizers of this function and, thus, nonsupported solutions can be obtained.

Usage

```
Tchebycheff(y, s = 100, rho = 0.1)
```

Arguments

`y`	Numerical matrix or data.frame containing the responses (on each column) to be scalarized.
`s`	Numerical integer (default: 100) setting the number of partitions the vector lambda has.
`rho`	A small positive value (default: 0.1) setting the "strenght" of the non-linear term.

References

Knowles, J. (2006). ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1), 50-66.

Examples

```
grid <- expand.grid(seq(0, 1, , 50),seq(0, 1, , 50))
 res <- t(apply(grid, 1, nowacki_beam))
 plot(nowacki_beam_tps$x, xlim=c(0,1), ylim=c(0,1))
 grid <- grid[which(as.logical(apply(res[,-(1:2)] < 0, 1, prod))),]
 res <- res[which(as.logical(apply(res[,-(1:2)] < 0, 1, prod))),1:2]
for (i in 1:10){
sres <- Tchebycheff(res[,1:2], s=100, rho=0.1)
points(grid[which.min(sres),], col='green')
}
```

`test_functions` *Test functions for optimization*

Description

This page is a collection of test functions commonly used to test optimization algorithms

Usage

```
Shaffer1(x)

Shaffer2(x)

Fonseca(x)

Kursawe(x)

Viennet(x)

Binh(x)
```

Arguments

`x,`	numeric value (or vector for multivariable functions)

References

https://en.wikipedia.org/wiki/Test_functions_for_optimization

http://www.sfu.ca/~ssurjano/optimization.html

Examples

```
#function should be evaluated in the -A < x < A interval,
#where A is from 10 to 10^5 and \length(x) = 1
Shaffer1(0)

#function should be evaluated in the -5 < x < 10 interval \length(x) = 1
Shaffer2(0)

#function should be evaluated in the -20 < x < 20 interval and \length(x) >= 1
Fonseca(rep(0,10))

#function should be evaluated in the -5 < x < 5 interval and \length(x) == 3
Kursawe(rep(0,3))

#function should be evaluated in the -3 < x < 3 interval and \length(x) == 2
Viennet(c(0.5,0.5))

#function should be evaluated in the 0 < x < (5,3) interval and \length(x) == 2
Binh(c(0,0))
```

VMPF — *VMPF: Variance Minimization of the Predicted Front*

Description

Executes `nsteps` iterations of the VMPF algorithm to an object of class `mkm`. At each step, a multi-objective kriging model is re-estimated (including covariance parameters re-estimation).

Usage

```
VMPF(model, fun, nsteps, lower = rep(0, model@d), upper = rep(1, model@d),
  quiet = TRUE, control = NULL, modelcontrol = NULL)
```

Arguments

`model`	An object of class `mkm`,
`fun`	The multi-objective and constraint cost function to be optimized. This function must return a vector with the size of `model@m + model@j` where `model@m` are the number of objectives and `model@j` the number of the constraints,
`nsteps`	An integer representing the desired number of iterations,
`lower`	Vector of lower bounds for the variables to be optimized over (default: 0 with length `model@d`),
`upper`	Vector of upper bounds for the variables to be optimized over (default: 1 with length `model@d`),
`quiet`	Logical indicating the verbosity of the routine,
`control`	An optional list of control parameters that controlls the optimization algorithm. One can control: `popsize` (default: `200`); `generations` (default: `30`);

cdist (default: 1/model@d);
mprob (default: 15);
mdist (defult: 20).

modelcontrol An optional list of control parameters to the mkm function (default: object@control).

Details

The infill point is sampled from the most uncertain design of a predicted Pareto set. This set is predicted using nsga-2 algorithm and the mean value of the mkm predictor.

Value

an updated object of class mkm.

Examples

```
# ----------------
# The Nowacki Beam
# ----------------
n <- 20
d <- 2
fun <- nowacki_beam
doe <- replicate(d,sample(0:n,n))/n
res <- t(apply(doe, 1, fun))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower = rep(0.1,d)))
model <- VMPF(model, fun, 20, quiet = FALSE, control = list(rho = 0.1))
plot(nowacki_beam_tps$set)
points(ps(model@response[which(model@feasible),model@objective])$set, col = 'green', pch = 19)
```

Index