Package ‘moko’

Type Package

Title Multi-Objective Kriging Optimization

Version 1.0.3

Description Multi-Objective optimization based on the Kriging metamodel. Important functions: mkm() (builder for the multiobjective models), MVPF() (sequential minimizer using variance reduction), MEGO() (generalization of ParEgo) and HEGO() (minimizer using the expected hypervolume improvement). References are Passos and Luersen (2018) <doi:10.1590/1679-78254324>.

Depends R (>= 3.3.0)

License GPL-3

LazyData TRUE

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

RoxygenNote 7.0.2


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

Suggests knitr, lhs

VignetteBuilder knitr

NeedsCompilation no

Author Adriano Passos [aut, cre], Marco Luersen [ctb]

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

Repository CRAN

Date/Publication 2020-06-02 15:50:02 UTC
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 package. However, the present implementation accounts for inequality constrains embedded into the mkm model.

Usage

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

Arguments

- **x**: a vector representing the input for which one wishes to calculate EHI, alternatively a matrix with one point per row.
- **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 optimization is not currently accepted, if the user needs it, the cost functions should be modified prior Kriging modeling (i.e. inverting or multiplying the output by $-1$).

paretoFront: object of class ps containing the actual Pareto set. If not provided a Pareto set is built based on the current feasible observations (model@response[model@feasible,]).

nb.samp: number of random samples from the posterior distribution (with more than two objectives), default to 50, increasing gives more reliable results at the cost of longer computation time.

seed: seed used for the random samples (with more than two objectives);

refPoint: reference point for Hypervolume Expected Improvement. If not provided, it is set to the maximum or minimum of each objective.

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 improvement of the objective.

Value

The constrained expected hypervolume improvement at $x$.

References


Examples

```r
# ------------------------
# The Nowacki Beam
# ------------------------

n <- 10
d <- 2
doe <- replicate(d, sample(0:n, n))/n
res <- t(apply(doe, 1, nowacki_beam, box = data.frame(b = c(10, 50), h = c(50, 250))))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower=rep(0.1,d)))
grid <- expand.grid(seq(0, 1, , 10),seq(0, 1, , 10))
ehvi <- apply(grid, 1, EHVI, model)
contour(matrix(ehvi, 20))
points(model@design, col=ifelse(model@feasible, 'blue', 'red'))
p points(grid[which.max(ehvi),], col='green', pch=19)
```
Constrained Expected Improvement

Description

This function extends the EI function supplied by the package archive package DiceOptim. This extension allows usage of multiple expensive constraints. The constraints are passed to the re-vamped EI function embedded inside the \texttt{mkm} object. Currently low-cost (explicit) constraints are not allowed.

Usage

\texttt{EI(x, model, control = NULL)}

Arguments

\begin{itemize}
\item \texttt{x} \hspace{1cm} A vector representing the input for which one wishes to calculate EI.
\item \texttt{model} \hspace{1cm} An object of class \texttt{mkm}. This model must have a single objective (model@m == 1).
\item \texttt{control} \hspace{1cm} An optional list of control parameters, some of them passed to the EI function. One can control:
  \begin{itemize}
  \item \texttt{minimization} \hspace{1cm} logical specifying if EI is used in minimization or in maximization (default: TRUE)
  \item \texttt{plugin} \hspace{1cm} 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 \texttt{NA} and the algorithm returns the value of the probability of constraints be met.
  \item \texttt{envir} \hspace{1cm} optional environment specifying where to assign intermediate values. Default: NULL.
  \end{itemize}
\end{itemize}

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 improvement of the objective. Another important consideration is that, by default, the value of the plugin passed to the EI is the best feasible observed value.

References

Examples

```r
# 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, , 25), seq(0, 1, , 25))
ei <- apply(grid, 1, EI, model) # this computation may take some time
contour(matrix(ei, 25))
points(model@design, col=ifelse(model@feasible, "blue", "red"))
points(grid[which.max(ei), ], col="green")
```

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

```r
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 optimization is not
currently accepted, if the user needs it, the cost functions should be mod-
ified prior Kriging modeling (i.e. inverting or multiplying the output by
-1).
paretoFront object of class ps containing the actual Pareto set. If not provided
a Pareto set is built based on the current feasible observations (model@response[model@feasible,]

nb.samp number of random samples from the posterior distribution (with more
than two objectives), default to 50, increasing gives more reliable results at
the cost of longer computation time
seed seed used for the random samples (with more than two objectives);
refPoint reference point for Hypervolume Expected Improvement. If not pro-
vided, it is set to the maximum or minimum of 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
(genSA(control = optimcontrol)).

Value
updated mkm model

Examples

# ----------------
# The Nowacki Beam
# ----------------
n <- 20
d <- 2
nsteps <- 1 # value has been set to 1 to save compilation time, change this value to 40.
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, nsteps, quiet = FALSE)
plot(nowacki_beam_tps$set)
points(ps(model@response[which(model@feasible),model@objective]$set, col = 'green', pch = 19)
Description

The IGD is a performance 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

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

Arguments

- `aps`: An object of type `ps` containing the "actual" Pareto front
- `tps`: An object of type `ps` containing the "true" Pareto front
- `method`: String stating which distance measure to be used. This must be one of: "euclidean" or "manhattan" (default).
- `norm`: Logical (default: `TRUE`) indicating if both fronts should be normalized.

Value

returns the IGD metric

References

considering expected hypervolume improvement in non-constrained many-objective test problems.
In 2013 IEEE Congress on Evolutionary Computation (pp. 658-665). IEEE.

Examples

```r
## Not run:
aps <- ps(matrix(rnorm(1:1000),ncol=2))
tps <- ps(matrix(rnorm(1:2000),ncol=2))
igd(aps,tps)

tps <- nowacki_beam_tps$set[1:50 * 10,]
aps <- tps * 1.2
igd(aps,tps)
## End(Not run)
```
### 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**

```r
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 optimization is not currently accepted, if the user needs it, the cost functions should be modified prior Kriging modeling (i.e. inverting or multiplying the output by -1).
  - `paretoFront` object of class `ps` containing the actual Pareto set. If not provided a Pareto set is built based on the current feasible observations (`model@response[model@feasible,]`).
  - `nb.samp` number of random samples from the posterior distribution (with more than two objectives), default to 50, increasing gives more reliable results at the cost of longer computation time.
  - `seed` seed used for the random samples (with more than two objectives).
  - `refPoint` reference point for Hypervolume Expected Improvement. If not provided, it is set to the maximum or minimum of 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 (`genSA(control = optimcontrol)`).
max_EI

Value

A list with components:

par  The best set of parameters found.
value  The value of expected hypervolume improvement at par.

Examples

```r
# ------------------------
# The Nowacki Beam
# ------------------------
n <- 10
d <- 2
doe <- replicate(d, sample(0:n:n))/n
res <- t(apply(doe, 1, nowacki_beam, box = data.frame(b = c(10, 50), h = c(50, 250))))
model <- mkm(doe, res, modelcontrol = list(objective = 1:2, lower=c(0.1,0.1)))
max_EHVI(model)
```

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

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

Arguments

- `model`: An object of class `mkm`. This model must have a single objective (model@m == 1).
- `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  A list of control parameters, some of them passed to the EI function.  One can control:

minimization logical specifying if EI is used in minimization 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 probability of constraints be met.

envir optional environment specifying where to assign intermediate values. Default: 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 (genSA(control = optimcontrol)).

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

```r
# --------------------------------------------
# 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. This model must have a single objective (model@m == 1).
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 minimization 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 probability of constraints be
          met.
          envir optional environment specifying where to assign intermediate values.
          Default: 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
          (genSA(control = optimcontrol)).

Details

Note that since MEGO is works by scalarizing a cost function, this technique is well suited for
single objective problems with multiple constraints.

Value

updated mkm model
References


Examples

```r
# ----------------
# The Nowacki Beam
# ----------------

n <- 20
d <- 2
nsteps <- 1 # value has been set to 1 to save compilation time, change this value to 40.
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, nsteps, 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 ####
########################################################################

# Not run:
## Those examples are flagged as "don't run" only to save compilation time. ##
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, 10, 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, 10, quiet = FALSE)
contour(x.grid,y.grid,z.grid,40)
points(model@design, col=ifelse(model@feasible,'blue','red'))

## End(Not run)

mkm

Multi-objective Kriging model
Description

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

Usage

\texttt{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 \texttt{km} function. One can control:

objective (default: 1:ncol(response))
quiet (default: \texttt{TRUE})
formula (default: \texttt{~1})
covtype (default: "matern5_2")
nugget.estim (default: \texttt{FALSE})
estim.method (default: "MLE")
optim.method (default: "BFGS")
multistart (default: 1)
gr (default: \texttt{TRUE})
iso (default: \texttt{FALSE})
scaling (default: \texttt{FALSE})
type (default: '\texttt{UK}'))
se.compute (default: \texttt{TRUE})
light.return (default: \texttt{TRUE})
bias.correct (default: \texttt{FALSE})
checkNames (default: \texttt{FALSE})
For more details, one can check \texttt{km}.

Value

S4 An object of class \texttt{mkm-class}

Examples

\begin{verbatim}
# ------------------------
# 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))
\end{verbatim}
Description
A 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.
optimisation 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 unconstrained multi-objective optimization based on the popular Kriging surrogate model.

Details
The main functions provided by moko are: MEGO, HEGO and MVPF.
MVPF: Minimization of the Variance of the Kriging-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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>An object of class mkm,</td>
</tr>
<tr>
<td>fun</td>
<td>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.</td>
</tr>
<tr>
<td>nsteps</td>
<td>An integer representing the desired number of iterations,</td>
</tr>
<tr>
<td>lower</td>
<td>Vector of lower bounds for the variables to be optimized over (default: 0 with length model@d),</td>
</tr>
<tr>
<td>upper</td>
<td>Vector of upper bounds for the variables to be optimized over (default: 1 with length model@d),</td>
</tr>
<tr>
<td>quiet</td>
<td>Logical indicating the verbosity of the routine,</td>
</tr>
<tr>
<td>control</td>
<td>An optional list of control parameters that controls the optimization algorithm. One can control:</td>
</tr>
<tr>
<td>modelcontrol</td>
<td>An optional list of control parameters to the mkm function (default: object@control).</td>
</tr>
</tbody>
</table>
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

```r
# ----------------
# The Nowacki Beam
# ----------------
n <- 20
d <- 2
nsteps <- 2 # value has been set to 2 to save compilation time, change this value to 40.
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 <- MVPF(model, fun, nsteps, quiet = FALSE)
plot(nowacki_beam_tps$set)
points(ps(model@response[which(model@feasible),model@objective]$set, col = 'green', pch = 19)
```

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 beam for minimum cross-sectional area and lowest bending stress subject to a number of constraints.

Usage

```r
nowacki_beam(
  x,
  g = c(5, 240, 120, 10, 2),
  l = 1500,
  F = 5000,
  E = 216620,
  G = 86650,
  v = 0.27,
  box = data.frame(b = c(10, 50), h = c(20, 250))
)
```
nowacki_beam_tps

Arguments

- **x**: vector of length 2 corresponds the normalized breadth and height of the beam.
- **g**: vector of length 5 containing the upper limits 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


Examples

```r
grid <- expand.grid(seq(0, 1, , 50),seq(0, 1, , 50))
res <- apply(grid, 1, nowacki_beam, box = data.frame(b = c(10, 50), h = c(50, 250)))
par(mfrow = c(3,3))
for(i in 1:nrow(res))
  contour(matrix(res[i,],50))
```

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 between 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*: String stating which 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 function performs predictions for a given dataset into a collection of Kriging models (mkm object).

**Usage**

```r
## 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

```r
# 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 nsga2 algorithm that builds.

**Usage**

```r
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 controls the optimization algorithm. One can control:
  - `popsise` (default: 200);
  - `generations` (default: 30);
  - `cdist` (default: `1/model@d`);
  - `mprob` (default: 15);
  - `mdist` (default: 20).
- **modelcontrol**: An optional list of control parameters to the `mkm` function (default: `object@control`).
ps

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 written 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)
Tchebycheff

Augmented Tchebycheff function

Description

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

Usage

\texttt{Tchebycheff(y, s = 100, rho = 0.1)}

Arguments

- \texttt{y} \hspace{1cm} \text{Numerical matrix or data.frame containing the responses (on each column) to be scalarized.}
- \texttt{s} \hspace{1cm} \text{Numerical integer (default: 100) setting the number of partitions the vector lambda has.}
- \texttt{rho} \hspace{1cm} \text{A small positive value (default: 0.1) setting the "strength" of the non-linear term.}

References


Examples

```r
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')
}
```
Description
This page is a collection of test functions commonly used to test optimization algorithms

Usage
- \texttt{Shaffer1}(x)
- \texttt{Shaffer2}(x)
- \texttt{Fonseca}(x)
- \texttt{Kursawe}(x)
- \texttt{Viennet}(x)
- \texttt{Binh}(x)

Arguments
- \texttt{x}, numeric value (or vector for multivariable functions)

References
- \url{https://en.wikipedia.org/wiki/Test_functions_for_optimization}
- \url{http://www.sfu.ca/~ssurjano/optimization.html}

Examples

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

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

#function should be evaluated in the \(-20 < x < 20\) interval and \text{\textbackslash length}(x) \geq 1 
\texttt{Fonseca}\left(\text{\textbackslash rep}(0,10)\right)

#function should be evaluated in the \(-5 < x < 5\) interval and \text{\textbackslash length}(x) = 3 
\texttt{Kursawe}\left(\text{\textbackslash rep}(0,3)\right)

#function should be evaluated in the \(-3 < x < 3\) interval and \text{\textbackslash length}(x) = 2 
\texttt{Viennet}\left(\text{\textbackslash c}(0.5,0.5)\right)
```
#function should be evaluated in the 0 < x < (5,3) interval and \(\text{length}(x) == 2\)
Binh(c(0,0))

## VMPF

** Deprecated function **

### Description

This function is deprecated and will be removed in a near future

### Usage

```r
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 controls the optimization algorithm. One can control:
  - `popsize` (default: 200);
  - `generations` (default: 30);
  - `cdist` (default: `1/model@d`);
  - `mprob` (default: 15);
  - `mdist` (default: 20).
- **modelcontrol**: An optional list of control parameters to the `mkm` function (default: `object@control`).
VMPF

See Also

MVPF
Index

*Topic datasets
  nowacki_beam_tps, 18

Binh (test_functions), 23
crit_EHI, 2, 6, 8
DiceKriging, 14
EHVI, 2
EI, 4
Fonseca (test_functions), 23
GenSA, 6, 8, 10, 11
HEGO, 5, 15
igd, 7
km, 14, 15
Kursawe (test_functions), 23
max_EHVI, 8
max_EI, 9
MEGO, 10, 15
mkm, 2, 4–6, 8–11, 13, 16, 19, 20, 24
mkm-class, 15
moko, 15
MVPF, 15, 16, 25

nowacki_beam, 17
nowacki_beam_tps, 18
nsga2, 20
pdist, 19
predict (predict, mkm-method), 19
predict, mkm-method, 19
predict_front, 20
ps, 3, 6–8, 21, 21
Shaffer1 (test_functions), 23
Shaffer2 (test_functions), 23
show, mkm-method (mkm-class), 15
Tchebycheff, 22
test_functions, 23
Viennet (test_functions), 23
VMPF, 24