Package 'ParamHelpers'

July 4, 2022

Title Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning

Version 1.14.1

Description Functions for parameter descriptions and operations in black-box optimization, tuning and machine learning. Parameters can be described (type, constraints, defaults, etc.), combined to parameter sets and can in general be programmed on. A useful OptPath object (archive) to log function evaluations is also provided.

License BSD_2_clause + file LICENSE

URL https://paramhelpers.mlr-org.com, https://github.com/mlr-org/ParamHelpers

BugReports https://github.com/mlr-org/ParamHelpers/issues

Imports backports, BBmisc (>= 1.10), checkmate (>= 1.8.2), fastmatch, methods

Suggests interp, covr, eaf, emoa, GGally, ggplot2, grid, gridExtra, irace (>= 2.1), lhs, plyr, reshape2, testthat

ByteCompile yes

Encoding UTF-8

RoxygenNote 7.1.0

NeedsCompilation yes

Author Bernd Bischl [aut] (<https://orcid.org/0000-0001-6002-6980>), Michel Lang [aut] (<https://orcid.org/0000-0001-9754-0393>), Jakob Richter [cre, aut] (<https://orcid.org/0000-0003-4481-5554>), Jakob Bossek [aut], Daniel Horn [aut], Karin Schork [ctb], Pascal Kerschke [aut]

Maintainer Jakob Richter <code@jakob-r.de>

Repository CRAN

Date/Publication 2022-07-04 11:29:28 UTC

R topics documented:

addOptPathEl . . . 3
as.data.frame.OptPathDF . . . 5
checkParamSet . . . 6
convertParamSetToIrace . . . 7
dfRowsToList . . . 7
discreteNameToValue . . . 8
discreteValueToName . . . 9
dropParams . . . 10
evaluateParamExpressions . . . 10
filterParams . . . 11
generateDesign . . . 13
generateDesignOfDefaults . . . 15
generateGridDesign . . . 16
generateRandomDesign . . . 17
getDefaults . . . 18
getLower . . . 19
getOptPathBestIndex . . . 21
getOptPathCol . . . 22
getOptPathCols . . . 23
getOptPathDOB . . . 24
getOptPathEl . . . 24
getOptPathEOL . . . 25
getOptPathErrorMessages . . . 26
getOptPathExecTimes . . . 27
getOptPathLength . . . 27
getOptPathParetoFront . . . 28
getOptPathX . . . 29
getOptPathY . . . 30
getParamIds . . . 31
getParamLengths . . . 31
getParamNr . . . 32
getParamSet . . . 33
getParamTypeCounts . . . 33
getParamTypes . . . 34
getRequirements . . . 35
getTypeStrings . . . 35
hasExpression . . . 36
hasFiniteBoxConstraints . . . 37
hasForbidden . . . 37
hasRequires . . . 38
hasTrafo . . . 38
hasType . . . 39
isEmpty . . . 39
isFeasible . . . 40
isForbidden . . . 41
isRequiresOk . . . 41

isSpecialValue . . . 42
isType . . . 43
isTypeString . . . 43
isVector . . . 44
LearnerParam . . . 45
listToDfOneRow . . . 48
makeParamSet . . . 49
OptPath . . . 50
Param . . . 52
paramValueToString . . . 56
plotEAF . . . 57
plotOptPath . . . 58
plotYTraces . . . 59
removeMissingValues . . . 60
renderOptPathPlot . . . 60
renderYTraces . . . 64
repairPoint . . . 64
sampleValue . . . 65
sampleValues . . . 66
setOptPathElDOB . . . 67
setOptPathElEOL . . . 68
setValueCNames . . . 68
trafoOptPath . . . 69
trafoValue . . . 70
updateParVals . . . 70

Index **72**

`addOptPathEl` *Add a new element to an optimization path.*

Description

Changes the argument in-place. Note that when adding parameters that have associated transformations, it is probably best to add the untransformed values to the path. Otherwise you have to switch off the feasibility check, as constraints might now not hold anymore.

Dependent parameters whose requirements are not satisfied must be represented by a scalar NA in the input.

Usage

```
addOptPathEl(
  op,
  x,
  y,
  dob = getOptPathLength(op) + 1L,
  eol = as.integer(NA),
```

```
  error.message = NA_character_,
  exec.time = NA_real_,
  extra = NULL,
  check.feasible = !op$add.transformed.x
)
```

Arguments

op	OptPath Optimization path.
x	(list) List of parameter values for a point in input space. Must be in same order as parameters.
y	(numeric) Vector of fitness values. Must be in same order as y.names.
dob	(integer(1)) Date of birth of the new parameters. Default is length of path + 1.
eol	(integer(1)) End of life of point. Default is NA.
error.message	(character(1)) Possible error message that occurred for this parameter values. Default is NA.
exec.time	(numeric(1)) Possible exec time for this evaluation. Default is NA.
extra	(list) Possible list of extra values to store. The list must be fully named. The list can contain nonscalar values, but these nonscalar entries must have a name starting with a dot (.). Other entries must be scalar, and must be in the same order of all calls of addOptPathEl. Watch out: if include.extra was set to TRUE in (makeOptPathDF()) the list of extras is mandatory. Default is NULL.
check.feasible	(logical(1)) Should x be checked with (isFeasible())? Default is TRUE.

Value

Nothing.

See Also

Other optpath: OptPath, getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

Examples

```
ps = makeParamSet(
  makeNumericParam("p1"),
```

```
  makeDiscreteParam("p2", values = c("a", "b"))
)
op = makeOptPathDF(par.set = ps, y.names = "y", minimize = TRUE)
addOptPathEl(op, x = list(p1 = 7, p2 = "b"), y = 1)
addOptPathEl(op, x = list(p1 = -1, p2 = "a"), y = 2)
as.data.frame(op)
```

`as.data.frame.OptPathDF` *Convert optimization path to data.frame.*

Description

The following types of columns are created:

x-numeric(vector)	`numeric`
x-integer(vector)	`integer`
x-discrete(vector)	`factor` (names of values = levels)
x-logical(vector)	`logical`
y-columns	`numeric`
dob	`integer`
eol	`integer`
error.message	`character`
exec.time	`numeric`
extra-columns	any

If you want to convert these, look at `BBmisc::convertDataFrameCols()`. Dependent parameters whose constraints are unsatisfied generate `NA` entries in their respective columns. Factor columns of discrete parameters always have their complete level set from the `param.set`.

Usage

```
## S3 method for class 'OptPathDF'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  include.x = TRUE,
  include.y = TRUE,
  include.rest = TRUE,
  dob = x$env$dob,
  eol = x$env$eol,
  ...
)
```

Arguments

x	(OptPath()) Optimization path.
row.names	character Row names for result. Default is none.
optional	(any) Currently ignored.
include.x	(logical(1)) Include all input params? Default is TRUE.
include.y	(logical(1)) Include all y-columns? Default is TRUE.
include.rest	(logical(1)) Include all other columns? Default is TRUE.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.
...	(any) Currently ignored.

Value

data.frame.

---

checkParamSet *Sanity check expressions of a parameter set.*

Description

Checks whether the default values of the numerical parameters are located within the corresponding boundaries. In case of discrete parameters it checks whether the values are a subset of the parameter's possible values.

Usage

```
checkParamSet(par.set, dict = NULL)
```

Arguments

par.set	ParamSet Parameter set.
dict	(environment | list | NULL) Environment or list which will be used for evaluating the variables of expressions within a parameter, parameter set or list of parameters. The default is NULL.

Value

TRUE on success. An exception is raised otherwise.

Examples

```
ps = makeParamSet(
  makeNumericParam("u", lower = expression(p)),
  makeIntegerParam("v", lower = 1, upper = expression(3 * p)),
  makeDiscreteParam("w", default = expression(z), values = c("a", "b")),
  makeDiscreteParam("x", default = "a", values = c("a", "b")),
  keys = c("p", "z")
)
checkParamSet(ps, dict = list(p = 3, z = "b"))
```

---

convertParamSetToIrace *Converts a ParamSet object to a parameter object of the irace package.*

Description

Converts to a textual description used in irace and then potentially calls readParameters.

Usage

```
convertParamSetToIrace(par.set, as.chars = FALSE)
```

Arguments

par.set	ParamSet Parameter set.
as.chars	(logical(1)) Return results as character vector of lines FALSE or call irace::readParameters() on it (TRUE). Default is FALSE.

Value

list() .

---

dfRowsToList *Convert a data.frame row to list of parameter-value-lists.*

Description

Please note that (naturally) the columns of df have to be of the correct type w.r.t. the corresponding parameter. The only exception are integer parameters where the corresponding columns in df are allowed to be numerics. And also see the argument enforce.col.types as a way around this restriction.

numeric(vector)	`numeric`
integer(vector)	`integer`
discrete(vector)	`factor` (names of values = levels)
logical(vector)	`logical`

Dependent parameters whose requirements are not satisfied are represented by a scalar NA in the output.

Usage

```
dfRowsToList(df, par.set, enforce.col.types = FALSE, ...)

dfRowToList(df, par.set, i, enforce.col.types = FALSE, ...)
```

Arguments

`df`	(`data.frame`) Data.frame, potentially from `OptPathDF()`. Columns are assumed to be in the same order as par.set.
`par.set`	ParamSet Parameter set.
`enforce.col.types`	(`logical(1)`) Should all `df` columns be initially converted to the type returned by `getParamTypes(df, df.cols = TRUE)`. This can help to work with "non-standard" data.frames where the types are slightly "off". But note that there is no guarantee that this will work if the types are really wrong and there is no naturally correct way to convert them. Default is `FALSE`.
`...`	(any) Arguments passed to `BBmisc::convertDataFrameCols()`
`i`	(`integer(1)`) Row index.

Value

`list`. Named by parameter ids.

`discreteNameToValue` *Convert encoding name(s) to discrete value(s).*

Description

For a discrete parameter or discrete vector. If the `name` is `NA`, indicating a missing parameter value due to unsatisfied requirements, `NA` is returned.

Usage

```
discreteNameToValue(par, name)
```

Arguments

par	Param Parameter.
name	(character) Name (string) encoding the value for a discrete parameter, or a character vector of names for a discrete vector.

Value

any. Parameter value for a discrete parameter or a list of values for a discrete vector.

Examples

```
p = makeDiscreteParam("u", values = c(x1 = "a", x2 = "b", x3 = "b"))
discreteNameToValue(p, "x3")
```

---

discreteValueToName *Convert discrete value(s) to encoding name(s).*

Description

For a discrete parameter or discrete vector. If the value x is NA, indicating a missing parameter value due to unsatisfied requirements, NA is returned.

Usage

```
discreteValueToName(par, x)
```

Arguments

par	Param Parameter.
x	any Parameter value or a list of values for a discrete vector.

Value

character. Single name for a discrete parameter or a character vector of names for a discrete vector.

Examples

```
p = makeDiscreteParam("u", values = c(x1 = "a", x2 = "b", x3 = "c"))
discreteValueToName(p, "b")
```

`dropParams` *Drop Params from ParamSet by ids.*

Description

Drop Params from ParamSet by ids.

Usage

```
dropParams(par.set, drop)
```

Arguments

`par.set`	ParamSet Parameter set.
`drop`	(`character`) ids of the `Param()`s in the `ParamSet()` to drop from the ParamSet.

Value

`ParamSet()` .

`evaluateParamExpressions`
Evaluates all expressions within a parameter.

Description

Evaluates the expressions of a parameter, parameter set or list of parameters for a given dictionary.

Usage

```
evaluateParamExpressions(obj, dict = NULL)
```

Arguments

`obj`	(`Param()` | `ParamHelpers::ParamSet()` | `list`) Parameter, parameter set or list of parameter values. Expressions within `len`, `lower` or `upper` boundaries, `default` or `values` will be evaluated using the provided dictionary (`dict`).
`dict`	(`environment` | list | `NULL`) Environment or list which will be used for evaluating the variables of expressions within a parameter, parameter set or list of parameters. The default is `NULL`.

Value

Param() | ParamHelpers::ParamSet() | list .

Examples

```
ps = makeParamSet(
  makeNumericParam("x", lower = expression(p), upper = expression(ceiling(3 * p))),
  makeIntegerParam("y", lower = 1, upper = 2)
)
evaluateParamExpressions(ps, dict = list(p = 3))

ps = makeParamSet(
  makeNumericParam("x", default = expression(sum(data$Species == "setosa"))),
  makeIntegerParam("y", lower = 1, upper = 2),
  keys = c("data", "Species")
)
evaluateParamExpressions(ps, dict = list(data = iris))

par.vals = list(
  x = expression(k),
  y = 5
)
evaluateParamExpressions(par.vals, dict = list(k = 3))
```

---

filterParams *Get parameter subset of only certain parameters.*

Description

Parameter order is not changed. It is possible to filter via multiple arguments, e.g., first filter based on id, then the type and lastly tunable. The order in which the filters are executed is always fixed (id > type > tunable).

Usage

```
filterParams(
  par.set,
  ids = NULL,
  type = NULL,
  tunable = c(TRUE, FALSE),
  check.requires = FALSE
)

filterParamsNumeric(
  par.set,
  ids = NULL,
  tunable = c(TRUE, FALSE),
  include.int = TRUE
```

```
)

filterParamsDiscrete(
  par.set,
  ids = NULL,
  tunable = c(TRUE, FALSE),
  include.logical = TRUE
)
```

Arguments

par.set	ParamSet Parameter set.
ids	(NULL | character) Vector with id strings containing the parameters to select. Has to be a subset of the parameter names within the parameter set. Per default (ids = NULL) no filtering based on names is done.
type	(NULL | character) Vector of allowed types, subset of: "numeric", "integer", "numericvector", "integervector", "discrete", "discretevector", "logical", "logicalvector", "character", "charactervector", "function", "untyped". Setting type = NULL, which is the default, allows the consideration of all types.
tunable	(logical) Vector of allowed values for the property tunable. Accepted arguments are TRUE, FALSE or c(TRUE, FALSE). The default is c(TRUE, FALSE), i.e. none of the parameters will be filtered out.
check.requires	(logical(1)) Toggle whether it should be checked that all requirements in the (ParamSet()) are still valid after filtering or not. This check is done after filtering and will throw an error if those Params are filtered which other Params need for their requirements. Default is FALSE.
include.int	(logical(1)) Are integers also considered to be numeric? Default is TRUE.
include.logical	(logical(1)) Are logicals also considered to be discrete? Default is TRUE.

Value

ParamSet().

Examples

```
ps = makeParamSet(
  makeNumericParam("u", lower = 1),
  makeIntegerParam("v", lower = 1, upper = 2),
  makeDiscreteParam("w", values = 1:2),
  makeLogicalParam("x"),
```

```
  makeCharacterParam("s"),
  makeNumericParam("y", tunable = FALSE)
)

# filter for numeric and integer parameters
filterParams(ps, type = c("integer", "numeric"))

# filter for tunable, numeric parameters
filterParams(ps, type = "numeric", tunable = TRUE)

# filter for all numeric parameters among "u", "v" and "x"
filterParams(ps, type = "numeric", ids = c("u", "v", "x"))
```

`generateDesign` *Generates a statistical design for a parameter set.*

Description

The following types of columns are created:

numeric(vector)	`numeric`
integer(vector)	`integer`
discrete(vector)	`factor` (names of values = levels)
logical(vector)	`logical`

If you want to convert these, look at `BBmisc::convertDataFrameCols()`. Dependent parameters whose constraints are unsatisfied generate `NA` entries in their respective columns. For discrete vectors the levels and their order will be preserved, even if not all levels are present.

Currently only lhs designs are supported.

The algorithm currently iterates the following steps:

1. We create a space filling design for all parameters, disregarding `requires`, a `trafo` or the forbidden region.
2. Forbidden points are removed.
3. Parameters are trafoed (potentially, depending on the setting of argument `trafo`); dependent parameters whose constraints are unsatisfied are set to `NA` entries.
4. Duplicated design points are removed. Duplicated points are not generated in a reasonable space-filling design, but the way discrete parameters and also parameter dependencies are handled make this possible.
5. If we removed some points, we now try to augment the design in a space-filling way and iterate.

Note that augmenting currently is somewhat experimental as we simply generate missing points via new calls to `lhs::randomLHS()`, but do not add points so they are maximally far away from the already present ones. The reason is that the latter is quite hard to achieve with complicated

dependencies and forbidden regions, if one wants to ensure that points actually get added... But we are working on it.

Note that if you have trafos attached to your params, the complete creation of the design (except for the detection of invalid parameters w.r.t to their `requires` setting) takes place on the UNTRANSFORMED scale. So this function creates, e.g., a maximin LHS design on the UNTRANSFORMED scale, but not necessarily the transformed scale.

`generateDesign` will NOT work if there are dependencies over multiple levels of parameters and the dependency is only given with respect to the "previous" parameter. A current workaround is to state all dependencies on all parameters involved. (We are working on it.)

Usage

```
generateDesign(
  n = 10L,
  par.set,
  fun,
  fun.args = list(),
  trafo = FALSE,
  augment = 20L
)
```

Arguments

`n`	(`integer(1)`) Number of samples in design. Default is 10.
`par.set`	ParamSet Parameter set.
`fun`	(`function`) Function from package lhs. Possible are: `lhs::maximinLHS()`, `lhs::randomLHS()`, `lhs::geneticLHS()`, `lhs::improvedLHS()`, `lhs::optAugmentLHS()`, `lhs::optimumLHS()` Default is `lhs::randomLHS()`.
`fun.args`	(`list`) List of further arguments passed to `fun`.
`trafo`	(`logical(1)`) Transform all parameters by using theirs respective transformation functions. Default is `FALSE`.
`augment`	(`integer(1)`) Duplicated values and forbidden regions in the parameter space can lead to the design becoming smaller than `n`. With this option it is possible to augment the design again to size `n`. It is not guaranteed that this always works (to full size) and `augment` specifies the number of tries to augment. If the the design is of size less than `n` after all tries, a warning is issued and the smaller design is returned. Default is 20.

Value

data.frame. Columns are named by the ids of the parameters. If the `par.set` argument contains a vector parameter, its corresponding column names in the design are the parameter id concatenated

with 1 to dimension of the vector. The result will have an `logical(1)` attribute "trafo", which is set to the value of argument `trafo`.

Examples

```
ps = makeParamSet(
  makeNumericParam("x1", lower = -2, upper = 1),
  makeIntegerParam("x2", lower = 10, upper = 20)
)
# random latin hypercube design with 5 samples:
generateDesign(5, ps)

# with trafo
ps = makeParamSet(
  makeNumericParam("x", lower = -2, upper = 1),
  makeNumericVectorParam("y", len = 2, lower = 0, upper = 1, trafo = function(x) x / sum(x))
)
generateDesign(10, ps, trafo = TRUE)
```

---

generateDesignOfDefaults

Generates a design with the defaults of a parameter set.

Description

The following types of columns are created:

numeric(vector)	`numeric`
integer(vector)	`integer`
discrete(vector)	`factor` (names of values = levels)
logical(vector)	`logical`

This will create a design containing only one point at the default values of the supplied param set. In most cases you will combine the resulting `data.frame` with a different generation function e.g. `generateDesign()`, `generateRandomDesign()` or `generateGridDesign()`. This is useful to force an evaluation at the default location of the parameters while still generating a design. Parameters default values, whose conditions (`requires`) are not fulfilled will be set to `NA` in the result.

Usage

```
generateDesignOfDefaults(par.set, trafo = FALSE)
```

Arguments

`par.set`	ParamSet Parameter set.

trafo (logical(1))
Transform all parameters by using theirs respective transformation functions. Default is FALSE.

Value

data.frame. Columns are named by the ids of the parameters. If the par.set argument contains a vector parameter, its corresponding column names in the design are the parameter id concatenated with 1 to dimension of the vector. The result will have an logical(1) attribute "trafo", which is set to the value of argument trafo.

---

generateGridDesign *Generates a grid design for a parameter set.*

Description

The following types of columns are created:

numeric(vector)	numeric
integer(vector)	integer
discrete(vector)	factor (names of values = levels)
logical(vector)	logical

If you want to convert these, look at BBmisc::convertDataFrameCols(). Dependent parameters whose constraints are unsatisfied generate NA entries in their respective columns. For discrete vectors the levels and their order will be preserved.

The algorithm currently performs these steps:

1. We create a grid. For numerics and integers we use the specified resolution. For discretes all values will be taken.
2. Forbidden points are removed.
3. Parameters are trafoed (potentially, depending on the setting of argument trafo); dependent parameters whose constraints are unsatisfied are set to NA entries.
4. Duplicated points are removed. Duplicated points are not generated in a grid design, but the way parameter dependencies are handled make this possible.

Note that if you have trafos attached to your params, the complete creation of the design (except for the detection of invalid parameters w.r.t to their requires setting) takes place on the UNTRANSFORMED scale. So this function creates a regular grid over the param space on the UNTRANSFORMED scale, but not necessarily the transformed scale.

generateDesign will NOT work if there are dependencies over multiple levels of parameters and the dependency is only given with respect to the "previous" parameter. A current workaround is to state all dependencies on all parameters involved. (We are working on it.)

Usage

```
generateGridDesign(par.set, resolution, trafo = FALSE)
```

Arguments

par.set	ParamSet Parameter set.
resolution	(integer) Resolution of the grid for each numeric/integer parameter in par.set. For vector parameters, it is the resolution per dimension. Either pass one resolution for all parameters, or a named vector.
trafo	(logical(1)) Transform all parameters by using theirs respective transformation functions. Default is FALSE.

Value

data.frame. Columns are named by the ids of the parameters. If the par.set argument contains a vector parameter, its corresponding column names in the design are the parameter id concatenated with 1 to dimension of the vector. The result will have an logical(1) attribute "trafo", which is set to the value of argument trafo.

Examples

```
ps = makeParamSet(
  makeNumericParam("x1", lower = -2, upper = 1),
  makeNumericParam("x2", lower = -2, upper = 2, trafo = function(x) x^2)
)
generateGridDesign(ps, resolution = c(x1 = 4, x2 = 5), trafo = TRUE)
```

---

generateRandomDesign *Generates a random design for a parameter set.*

Description

The following types of columns are created:

numeric(vector)	numeric
integer(vector)	integer
discrete(vector)	factor (names of values = levels)
logical(vector)	logical

If you want to convert these, look at BBmisc::convertDataFrameCols(). For discrete vectors the levels and their order will be preserved, even if not all levels are present.

The algorithm simply calls sampleValues() and arranges the result in a data.frame.

Parameters are trafoed (potentially, depending on the setting of argument trafo); dependent parameters whose constraints are unsatisfied are set to NA entries.

generateRandomDesign will NOT work if there are dependencies over multiple levels of parameters and the dependency is only given with respect to the "previous" parameter. A current

workaround is to state all dependencies on all parameters involved. (We are working on it.)

Note that if you have trafos attached to your params, the complete creation of the design (except for the detection of invalid parameters w.r.t to their `requires` setting) takes place on the UNTRANSFORMED scale. So this function samples from a uniform density over the param space on the UNTRANSFORMED scale, but not necessarily the transformed scale.

Usage

```
generateRandomDesign(n = 10L, par.set, trafo = FALSE)
```

Arguments

`n`	(`integer(1)`) Number of samples in design. Default is 10.
`par.set`	ParamSet Parameter set.
`trafo`	(`logical(1)`) Transform all parameters by using theirs respective transformation functions. Default is `FALSE`.

Value

data.frame. Columns are named by the ids of the parameters. If the `par.set` argument contains a vector parameter, its corresponding column names in the design are the parameter id concatenated with 1 to dimension of the vector. The result will have an `logical(1)` attribute "trafo", which is set to the value of argument `trafo`.

`getDefaults` *Return defaults of parameters in parameter set.*

Description

Return defaults of single parameters or parameters in a parameter set or a list of parameters.

Usage

```
getDefaults(obj, include.null = FALSE, dict = NULL)
```

Arguments

`obj`	(`Param()` | `ParamSet()` | `list`) Parameter, parameter set or list of parameters, whose defaults should be extracted. In case the default values contain expressions, they will be evaluated using the provided dictionary (`dict`).

include.null (logical(1))
Include NULL entries for parameters without default values in the result list? Note that this can be slightly dangerous as NULL might be used as default value for other parameters. Default is FALSE.

dict (environment | list | NULL)
Environment or list which will be used for evaluating the variables of expressions within a parameter, parameter set or list of parameters. The default is NULL.

Value

named list. Named (and in case of a ParamSet(), in the same order). Parameters without defaults are not present in the list.

Examples

```
ps1 = makeParamSet(
  makeDiscreteParam("x", values = c("a", "b"), default = "a"),
  makeNumericVectorParam("y", len = 2),
  makeIntegerParam("z", default = 99)
)
getDefaults(ps1, include.null = TRUE)

ps2 = makeParamSet(
  makeNumericVectorParam("a", len = expression(k), default = expression(p)),
  makeIntegerParam("b", default = 99),
  makeLogicalParam("c")
)
getDefaults(ps2, dict = list(k = 3, p = 5.4))
```

---

getLower *Get lower / upper bounds and allowed discrete values for parameters.*

Description

getLower and getUpper return a numerical vector of lower and upper bounds, getValues returns a list of possible value sets for discrete parameters.

Parameters for which such bound make no sense - due to their type - are not present in the result.

Usage

```
getLower(obj, with.nr = FALSE, dict = NULL)

getUpper(obj, with.nr = FALSE, dict = NULL)

getValues(obj, dict = NULL)
```

Arguments

obj	(Param()	ParamSet()	list) Parameter, parameter set or list of parameters, whose boundaries and/or values should be extracted. In case the boundaries or values contain expressions, they will be evaluated using the provided dictionary dict.
with.nr	(logical(1)) Should number from 1 to length be appended to names of vector params? Default is FALSE.
dict	(environment | list | NULL) Environment or list which will be used for evaluating the variables of expressions within a parameter, parameter set or list of parameters. The default is NULL.

Value

vector | list. Named by parameter ids.

Examples

```
ps = makeParamSet(
  makeNumericParam("u"),
  makeDiscreteParam("v", values = c("a", "b")),
  makeIntegerParam("w", lower = expression(ceiling(p / 3)), upper = 2),
  makeDiscreteParam("x", values = 1:2),
  makeNumericVectorParam("y", len = 2, lower = c(0, 10), upper = c(1, 11)),
  keys = "p"
)
getLower(ps, dict = list(p = 7))
getUpper(ps)

ps = makeParamSet(
  makeNumericParam("u"),
  makeDiscreteParam("w", values = list(a = list(), b = NULL))
)
getValues(ps)

par.vals = list(
  u = makeNumericParam("u"),
  v = makeIntegerParam("v", lower = 1, upper = 2),
  w = makeDiscreteParam("w", values = 1:2),
  x = makeNumericVectorParam("x", len = 2, lower = c(3, 1), upper = expression(n))
)
getLower(par.vals)
getUpper(par.vals, dict = list(n = 12))
```

`getOptPathBestIndex` *Get index of the best element from optimization path.*

Description

Get index of the best element from optimization path.

Usage

```
getOptPathBestIndex(
  op,
  y.name = op$y.names[1],
  dob = op$env$dob,
  eol = op$env$eol,
  ties = "last"
)
```

Arguments

`op`	OptPath Optimization path.
`y.name`	(`character(1)`) Name of target value to decide which element is best. Default is `y.names[1]`.
`dob`	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
`eol`	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in `eol` are selected. Default is all.
`ties`	(`character(1)`) How should ties be broken when more than one optimal element is found? "all": return all indices, "first": return first optimal element in path, "last": return last optimal element in path, "random": return random optimal element in path. Default is "last".

Value

`integer` Index or indices into path. See `ties`.

See Also

Other optpath: `OptPath`, `addOptPathEl()`, `getOptPathCols()`, `getOptPathCol()`, `getOptPathDOB()`, `getOptPathEOL()`, `getOptPathEl()`, `getOptPathErrorMessages()`, `getOptPathExecTimes()`, `getOptPathLength()`, `getOptPathParetoFront()`, `getOptPathX()`, `getOptPathY()`, `setOptPathElDOB()`, `setOptPathElEOL()`

Examples

```
ps = makeParamSet(makeNumericParam("x"))
op = makeOptPathDF(par.set = ps, y.names = "y", minimize = TRUE)
addOptPathEl(op, x = list(x = 1), y = 5)
addOptPathEl(op, x = list(x = 2), y = 3)
addOptPathEl(op, x = list(x = 3), y = 9)
addOptPathEl(op, x = list(x = 4), y = 3)
as.data.frame(op)
getOptPathBestIndex(op)
getOptPathBestIndex(op, ties = "first")
```

---

getOptPathCol *Get column from the optimization path.*

Description

Get column from the optimization path.

Usage

```
getOptPathCol(op, name, dob, eol)
```

Arguments

op	OptPath Optimization path.
name	(character(1)) Name of the column.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.

Value

Single column as a vector.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

getOptPathCols *Get columns from the optimization path.*

Description

Get columns from the optimization path.

Usage

```
getOptPathCols(op, names, dob, eol, row.names = NULL)
```

Arguments

op	OptPath Optimization path.
names	character Names of the columns.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.
row.names	character Row names for result. Default is none.

Value

data.frame.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

getOptPathDOB *Get date-of-birth vector from the optimization path.*

Description

Get date-of-birth vector from the optimization path.

Usage

```
getOptPathDOB(op, dob, eol)
```

Arguments

op	OptPath Optimization path.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.

Value

integer.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

getOptPathEl *Get an element from the optimization path.*

Description

Dependent parameters whose requirements are not satisfied are represented by a scalar NA in the elements of x of the return value.

Usage

```
getOptPathEl(op, index)
```

Arguments

op	OptPath Optimization path.
index	(integer(1)) Index of element.

Value

List with elements x (named list), y (named numeric), dob integer(1), eol integer(1). The elements error.message (character(1)), exec.time (numeric(1)) and extra (named list) are there if the respective options in OptPath() are enabled.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

---

getOptPathEOL *Get end-of-life vector from the optimization path.*

Description

Get end-of-life vector from the optimization path.

Usage

```
getOptPathEOL(op, dob, eol)
```

Arguments

op	OptPath Optimization path.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.

Value

integer.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

`getOptPathErrorMessages`

Get error-message vector from the optimization path.

Description

Get error-message vector from the optimization path.

Usage

```
getOptPathErrorMessages(op, dob, eol)
```

Arguments

`op`	OptPath Optimization path.
`dob`	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in `dob` are selected. Default is all.
`eol`	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in `eol` are selected. Default is all.

Value

character.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

getOptPathExecTimes *Get exec-time vector from the optimization path.*

Description

Get exec-time vector from the optimization path.

Usage

```
getOptPathExecTimes(op, dob, eol)
```

Arguments

op	OptPath Optimization path.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.

Value

numeric.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

getOptPathLength *Get the length of the optimization path.*

Description

Dependent parameters whose requirements are not satisfied are represented by a scalar NA in the output.

Usage

```
getOptPathLength(op)
```

Arguments

op	OptPath Optimization path.

Value

integer(1)

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

---

getOptPathParetoFront *Get indices of pareto front of optimization path.*

Description

Get indices of pareto front of optimization path.

Usage

```
getOptPathParetoFront(
  op,
  y.names = op$y.names,
  dob = op$env$dob,
  eol = op$env$eol,
  index = FALSE
)
```

Arguments

op	OptPath Optimization path.
y.names	character Names of performance measures to construct pareto front for. Default is all performance measures.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.
index	(logical(1)) Return indices into path of front or y-matrix of nondominated points? Default is FALSE.

Value

matrix | integer. Either matrix (with named columns) of points of front in objective space or indices into path for front.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

Examples

```
ps = makeParamSet(makeNumericParam("x"))
op = makeOptPathDF(par.set = ps, y.names = c("y1", "y2"), minimize = c(TRUE, TRUE))
addOptPathEl(op, x = list(x = 1), y = c(5, 3))
addOptPathEl(op, x = list(x = 2), y = c(2, 4))
addOptPathEl(op, x = list(x = 3), y = c(9, 4))
addOptPathEl(op, x = list(x = 4), y = c(4, 9))
as.data.frame(op)
getOptPathParetoFront(op)
getOptPathParetoFront(op, index = TRUE)
```

---

getOptPathX — *Get data.frame of input points (X-space) referring to the param set from the optimization path.*

Description

Get data.frame of input points (X-space) referring to the param set from the optimization path.

Usage

```
getOptPathX(op, dob, eol)
```

Arguments

op	OptPath Optimization path.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.

Value

data.frame.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

---

getOptPathY *Get y-vector or y-matrix from the optimization path.*

Description

Get y-vector or y-matrix from the optimization path.

Usage

```
getOptPathY(op, names, dob, eol, drop = TRUE)
```

Arguments

op	OptPath Optimization path.
names	character Names of performance measure. Default is all performance measures in path.
dob	integer Vector of date-of-birth values to further subset the result. Only elements with a date-of-birth included in dob are selected. Default is all.
eol	integer Vector of end-of-life values to further subset the result. Only elements with an end-of-life included in eol are selected. Default is all.
drop	(logical(1)) Return vector instead of matrix when only one y-column was selected? Default is TRUE.

Value

(numeric | matrix). The columns of the matrix are always named.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), setOptPathElDOB(), setOptPathElEOL()

getParamIds *Return ids of parameters in parameter set.*

Description

Useful if vectors are included.

Usage

```
getParamIds(par, repeated = FALSE, with.nr = FALSE)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.
repeated	(logical(1)) Should ids be repeated length-times if parameter is a vector? Default is FALSE.
with.nr	(logical(1)) Should number from 1 to length be appended to id if repeated is TRUE? Otherwise ignored. Default is FALSE.

Value

character.

Examples

```
ps = makeParamSet(
  makeNumericParam("u"),
  makeIntegerVectorParam("v", len = 2)
)
getParamIds(ps)
getParamIds(ps, repeated = TRUE)
getParamIds(ps, repeated = TRUE, with.nr = TRUE)
```

getParamLengths *Return lengths of single parameters or parameters in parameter set.*

Description

Useful for vector parameters.

Usage

```
getParamLengths(par, dict = NULL)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.
dict	(environment | list | NULL) Environment or list which will be used for evaluating the variables of expressions within a parameter, parameter set or list of parameters. The default is NULL.

Value

(integer). Named and in the same order as the input for ParamSet() input.

Examples

```
ps = makeParamSet(
  makeNumericParam("u"),
  makeIntegerParam("v", lower = 1, upper = 2),
  makeDiscreteParam("w", values = 1:2),
  makeDiscreteVectorParam("x", len = 2, values = c("a", "b"))
)
getParamLengths(ps)
# the length of the vector x is 2, for all other single value parameters the length is 1.

par = makeNumericVectorParam("x", len = expression(k), lower = 0)
getParamLengths(par, dict = list(k = 4))
```

---

getParamNr *Return number of parameters in set.*

Description

Either number of parameters or sum over parameter lengths.

Usage

```
getParamNr(par.set, devectorize = FALSE)
```

Arguments

par.set	ParamSet Parameter set.
devectorize	(logical(1)) Sum over length of vector parameters? Default is codeFALSE.

Value

integer.

Examples

```
ps = makeParamSet(
  makeNumericParam("u"),
  makeDiscreteVectorParam("x", len = 2, values = c("a", "b"))
)
getParamNr(ps)
getParamNr(ps, devectorize = TRUE)
```

---

getParamSet *Get parameter set.*

Description

`getParamSet` is a generic and can be called to extract the `ParamSet` from different objects.

Usage

```
getParamSet(x)
```

Arguments

`x`	`(object)` Object to extract the ParamSet from.

Value

`ParamHelpers::ParamSet()`

---

getParamTypeCounts *Returns information on the number of parameters of a each type.*

Description

Returns information on the number of parameters of a each type.

Usage

```
getParamTypeCounts(par.set)
```

Arguments

`par.set`	ParamSet Parameter set.

Value

`list` Named list which contains for each supported parameter type the number of parameters of this type in the given ParamSet.

getParamTypes *Returns type information for a parameter set.*

Description

Returns type information for a parameter set.

Usage

```
getParamTypes(
  par.set,
  df.cols = FALSE,
  df.discretes.as.factor = TRUE,
  use.names = FALSE,
  with.nr = TRUE
)
```

Arguments

par.set	ParamSet Parameter set.
df.cols	(logical(1)) If FALSE simply return the parameter types in the set, i.e., par$type. If TRUE, convert types so they correspond to R types of a data.frame where values of this set might be stored. This also results in replication of output types for vector parameters. Default is FALSE.
df.discretes.as.factor	(logical(1)) If df.cols is TRUE: Should type for discrete params be factor or character? Default is TRUE.
use.names	(logical(1)) Name the result vector? Default is FALSE.
with.nr	(logical(1)) Should number from 1 to length be appended to name? Only used if use.names and df.cols are TRUE. Default is TRUE.

Value

character.

getRequirements *Return all require-expressions of a param set.*

Description

Returns all requires-objects of a param set as a list.

Usage

```
getRequirements(par.set, remove.null = TRUE)
```

Arguments

par.set	ParamSet Parameter set.
remove.null	(logical(1)) If not set, params without a requires-setting will result in a NULL element in the returned list, otherwise they are removed. Default is codeTRUE.

Value

xnamed list. Named list of require-call-objects, lengths corresponds to number of params (potentially only the subset with requires-field), named with with param ids.

getTypeStrings *Get parameter type-strings.*

Description

Returns type strings used in param$type for certain groups of parameters.

Usage

```
getTypeStringsAll()

getTypeStringsNumeric(include.int = TRUE)

getTypeStringsNumericStrict()

getTypeStringsInteger()

getTypeStringsCharacter()

getTypeStringsDiscrete(include.logical = TRUE)

getTypeStringsLogical()
```

Arguments

include.int (logical(1))
Are integers also considered to be numeric? Default is TRUE.

include.logical (logical(1))
Are logicals also considered to be discrete? Default is TRUE.

Value

character.

---

hasExpression *Check if parameter values contain expressions.*

Description

Checks if a parameter, parameter set or list of parameters contain expressions.

Usage

```
hasExpression(obj)
```

Arguments

obj (Param() | ParamHelpers::ParamSet() | list)
Parameter, parameter set or list of parameters.

Value

logical(1).

Examples

```
ps1 = makeParamSet(
  makeNumericParam("x", lower = 1, upper = 2),
  makeNumericParam("y", lower = 1, upper = 10)
)

ps2 = makeParamSet(
  makeNumericLearnerParam("x", lower = 1, upper = 2),
  makeNumericLearnerParam("y", lower = 1, upper = expression(p))
)

hasExpression(ps1)
hasExpression(ps2)
```

hasFiniteBoxConstraints

Checks if a parameter or each parameter of a parameter set has ONLY finite lower and upper bounds.

Description

Checks if a parameter or each parameter of a parameter set has ONLY finite lower and upper bounds.

Usage

```
hasFiniteBoxConstraints(par, dict = NULL)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.
dict	(environment | list | NULL) Environment or list which will be used for evaluating the variables of expressions within a parameter, parameter set or list of parameters. The default is NULL.

Value

logical(1)

hasForbidden

Check parameter set for forbidden region.

Description

Check parameter set for forbidden region.

Usage

```
hasForbidden(par.set)
```

Arguments

par.set	ParamSet Parameter set.

Value

logical(1).

hasRequires *Check parameter / parameter set for requirements / dependencies.*

Description

TRUE iff the parameter has any requirements or any parameter in the set has requirements.

Usage

```
hasRequires(par)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.

Value

logical(1).

hasTrafo *Check parameter / parameter set for trafos.*

Description

TRUE iff the parameter has any trafos or any parameter in the set has trafos.

Usage

```
hasTrafo(par)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.

Value

logical(1).

hasType — *Check whether parameter set contains a certain type.*

Description

TRUE if the parameter set contains at least one parameter of the mentioned type x. Type x always subsumes x and x-vector.

Usage

```
hasDiscrete(par.set, include.logical = TRUE)

hasInteger(par.set)

hasLogical(par.set)

hasCharacter(par.set)

hasNumeric(par.set, include.int = TRUE)
```

Arguments

par.set	ParamSet Parameter set.
include.logical	(logical(1)) Are logicals also considered to be discrete? Default is TRUE.
include.int	(logical(1)) Are integers also considered to be numeric? Default is TRUE.

Value

logical(1)

isEmpty — *Check whether parameter set is empty.*

Description

Check whether parameter set is empty.

Usage

```
isEmpty(par.set)
```

Arguments

par.set	(ParamSet()]) Parameter set.

Value

logical(1).

---

isFeasible *Check if parameter value is valid.*

Description

Check if a parameter value satisfies the constraints of the parameter description. This includes the requires expressions and the forbidden expression, if par is a ParamSet(). If requires is not satisfied, the parameter value must be set to scalar NA to be still feasible, a single scalar even in a case of a vector parameter. If the result is FALSE the attribute "warning" is attached which gives the reason for the negative result.

If the parameter has cnames, these are also checked.

Usage

```
isFeasible(par, x, use.defaults = FALSE, filter = FALSE)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.
x	(any) Single value to check against the Param or ParamSet. For a ParamSet x must be a list. x has to contain the untransformed values. If the list is named, it is possible to only pass a subset of parameters defined in the ParamSet() par. In that case, only conditions regarding the passed parameters are checked. (Note that this might not work if one of the passed params has a requires setting which refers to an unpassed param.)
use.defaults	(logical(1)) Whether defaults of the Param()/ParamSet() should be used if no values are supplied. If the defaults have requirements that are not met by x it will be feasible nonetheless. Default is FALSE.
filter	(logical(1)) Whether the ParamSet() should be reduced to the space of the given Param Values. Note that in case of use.defaults = TRUE the filtering will be conducted after the insertion of the default values. Default is FALSE.

Value

logical(1).

Examples

```
p = makeNumericParam("x", lower = -1, upper = 1)
isFeasible(p, 0) # True
isFeasible(p, 2) # False, out of bounds
isFeasible(p, "a") # False, wrong type
# now for parameter sets
ps = makeParamSet(
  makeNumericParam("x", lower = -1, upper = 1),
  makeDiscreteParam("y", values = c("a", "b"))
)
isFeasible(ps, list(0, "a")) # True
isFeasible(ps, list("a", 0)) # False, wrong order
```

---

isForbidden *Check whether parameter setting lies in forbidden region of parameter set.*

Description

Parameter sets without a forbidden region always return FALSE.

Usage

```
isForbidden(par.set, x)
```

Arguments

par.set	ParamSet Parameter set.
x	(named list) Parameter setting to check.

Value

logical(1).

---

isRequiresOk *Check if parameter requirements are met.*

Description

Check if a parameter value satisfies the requirements of the parameter description. This only checks the requires expressions.

Usage

```
isRequiresOk(par.set, par.vals, ids = names(par.vals), use.defaults = TRUE)
```

Arguments

par.set	ParamSet Parameter set.
par.vals	(list()) List of parameter settings.
ids	(character()) ids of the param.vals to check. Default is names(par.vals).
use.defaults	(logical()) Some requirements relay on default values of the par.set. Default is TRUE, which means that if the value is not present in par.vals the default value will be considered.

Value

logical(1)

---

isSpecialValue *Is a given value in the list of special values for a param?*

Description

See title.

Usage

```
isSpecialValue(par, x)
```

Arguments

par	Param Parameter.
x	(any) Single value to check.

Value

logical(1).

isType — *Check parameter / parameter set contain ONLY a certain type.*

Description

An empty param set is considered to be of all types.

Usage

```
isNumeric(par, include.int = TRUE)

isDiscrete(par, include.logical = TRUE)

isInteger(par)

isLogical(par)

isCharacter(par)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.
include.int	(logical(1)) Are integers also considered to be numeric? Default is TRUE.
include.logical	(logical(1)) Are logicals also considered to be discrete? Default is TRUE.

Value

(logical(1))

isTypeString — *Check if type string is of certain type.*

Description

TRUE iff the type string is a certain type, e.g. isIntegerTypeString checks if we have "integer" or "integervector", and isVectorTypeString check if we have "*vector".

Usage

```
isNumericTypeString(type, include.int = TRUE)

isIntegerTypeString(type)

isCharacterTypeString(type)

isDiscreteTypeString(type, include.logical = TRUE)

isLogicalTypeString(type)

isVectorTypeString(type)
```

Arguments

type	(character(1)) Type string.
include.int	(logical(1)) Are integers also considered to be numeric? Default is TRUE.
include.logical	(logical(1)) Are logicals also considered to be discrete? Default is TRUE.

Value

(logical(1))

---

isVector *Check parameter / parameter set for vector params.*

Description

TRUE if the parameter is a vector parameter or all parameters in the set are vector parameters.

Usage

```
isVector(par)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.

Value

logical(1).

LearnerParam *Create a description object for a parameter of a machine learning algorithm.*

Description

This specializes `Param()` by adding a few more attributes, like a default value, whether it refers to a training or a predict function, etc. Note that you can set `length` to `NA`

The S3 class is a `Param()` which additionally stores these elements:

when `character(1)` See argument of same name.

See the note in `Param()` about being able to pass expressions to certain arguments.

Usage

```
makeNumericLearnerParam(
  id,
  lower = -Inf,
  upper = Inf,
  allow.inf = FALSE,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeNumericVectorLearnerParam(
  id,
  len = as.integer(NA),
  lower = -Inf,
  upper = Inf,
  allow.inf = FALSE,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeIntegerLearnerParam(
  id,
  lower = -Inf,
  upper = Inf,
  default,
  when = "train",
```

```
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeIntegerVectorLearnerParam(
  id,
  len = as.integer(NA),
  lower = -Inf,
  upper = Inf,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeDiscreteLearnerParam(
  id,
  values,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeDiscreteVectorLearnerParam(
  id,
  len = as.integer(NA),
  values,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeLogicalLearnerParam(
  id,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeLogicalVectorLearnerParam(
```

```
  id,
  len = as.integer(NA),
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeUntypedLearnerParam(
  id,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeFunctionLearnerParam(
  id,
  default,
  when = "train",
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)
```

Arguments

id	(character(1)) Name of parameter.
lower	(numeric | expression) Lower bounds. A singe value of length 1 is automatically replicated to len for vector parameters. If len = NA you can only pass length-1 scalars. Default is -Inf.
upper	(numeric | expression) Upper bounds. A singe value of length 1 is automatically replicated to len for vector parameters. If len = NA you can only pass length-1 scalars. Default is Inf.
allow.inf	(logical(1)) Allow infinite values for numeric and numericvector params to be feasible settings. Default is FALSE.
default	(any concrete value | expression) Default value used in learner. Note: When this is a discrete parameter make sure to use a VALUE here, not the NAME of the value. If this argument is missing, it means no default value is available.
when	(character(1))

Specifies when parameter is used in the learner: "train", "predict" or "both". Default is "train".

requires (NULL | call | expression)
States requirements on other parameters' values, so that setting this parameter only makes sense if its requirements are satisfied (dependent parameter). Can be an object created either with expression or quote, the former type is auto-converted into the later. Only really useful if the parameter is included in a (ParamSet()). Default is NULL which means no requirements.

tunable (logical(1))
Is this parameter tunable? Defining a parameter to be not-tunable allows to mark arguments like, e.g., "verbose" or other purely technical stuff. Note that this flag is most likely not respected by optimizing procedures unless stated otherwise. Default is TRUE (except for untyped, function, character and characterVector) which means it is tunable.

special.vals (list())
A list of special values the parameter can except which are outside of the defined range. Default is an empty list.

len (integer(1))
Length of vector parameter. Can be set to NA to define a vector with unspecified length.

values (vector | list | expression)
Possible discrete values. Instead of using a vector of atomic values, you are also allowed to pass a list of quite "complex" R objects, which are used as discrete choices. If you do the latter, the elements must be uniquely named, so that the names can be used as internal representations for the choice.

Value

LearnerParam().

---

listToDfOneRow *Convert a list to a data.frame with one row*

Description

Convert a list of vectors or scalars to a data.frame with only one row. Names of the columns correspond to the names of elements in the list. If a vector is one list element it is spread over multiple columns and named sequentially, e.g. a = c(5,7) becomes data.frame(a1 = 5, a2 = 7).

Usage

```
listToDfOneRow(l)
```

Arguments

l (list)
of atomic values of vectors.

Value

(data.frame) with only one row, containing the list elements.

---

makeParamSet *Construct a parameter set.*

Description

makeParamSet: Construct from a bunch of parameters.

Multiple sets can be concatenated with c.

The constructed S3 class is simply a list that contains the element pars. pars is a list of the passed parameters, named by their ids.

If keys are provided it will automatically be checked whether all expressions within the provided parameters only contain arguments that are a subset of keys.

Usage

```
makeParamSet(..., params = NULL, forbidden = NULL, keys = NULL)

makeNumericParamSet(id = "x", len, lower = -Inf, upper = Inf, vector = TRUE)
```

Arguments

...	(Param()) Parameters.
params	(list of Param()) List of parameters, alternative way instead of using ....
forbidden	(NULL | R expression) States forbidden region of parameter set via an expression. Every setting which satisfies this expression is considered to be infeasible. This makes it possible to exclude more complex region of the parameter space than through simple constraints or requires-conditions (although these should be always used when possible). If parameters have associated trafos, the forbidden region must always be specified on the original scale and not the transformed one. Default is NULL which means no forbidden region.
keys	character Character vector with keys (names) of feasible variable names which will be provided via a dictionary/hash later. Default is NULL.
id	(character(1)) Name of parameter.
len	(integer(1)) Length of vector.
lower	(numeric) Lower bound. Default is -Inf.

upper — numeric
Upper bound. Default is Inf.

vector — (logical(1))
Should a NumericVectorParam be used instead of n NumericParam objects? Default is TRUE.

Value

ParamSet() | LearnerParamSet. If all parameters of the ParamSet are learner parameters, the output will inherit the class LearnerParamSet.

Examples

```
makeParamSet(
  makeNumericParam("u", lower = 1),
  makeIntegerParam("v", lower = 1, upper = 2),
  makeDiscreteParam("w", values = 1:2),
  makeLogicalParam("x"),
  makeDiscreteVectorParam("y", len = 2, values = c("a", "b"))
)
makeParamSet(
  makeNumericParam("u", lower = expression(ceiling(n))),
  makeIntegerParam("v", lower = expression(floor(n)), upper = 2),
  keys = c("p", "n")
)
makeParamSet(
  makeNumericParam("min", lower = 0, upper = 0.8),
  makeNumericParam("max", lower = 0.2, upper = 1),
  forbidden = expression(min > max)
)
```

OptPath — *Create optimization path.*

Description

Optimizers can iteratively log their evaluated points into this object. Can be converted into a data.frame with as.data.frame(x, discretes.as.factor = TRUE / FALSE).

A optimization path has a number of path elements, where each element consists of: the value of the decision variables at this point, the values of the performance measures at this point, the date-of-birth (dob) of this point, the end-of-life (eol) of this point and possibly an error message. See also addOptPathEl().

For discrete parameters always the name of the value is stored as a character. When you retrieve an element with getOptPathEl(), this name is converted to the actual discrete value.

If parameters have associated transformation you are free to decide whether you want to add x values before or after transformation, see argument add.transformed.x and trafoOptPath().

The S3 class is a list which stores at least these elements:

par.set ParamSet() See argument of same name.

y.names character See argument of same name.

minimize logical See argument of same name.

add.transformed.x logical(1) See argument of same name.

env environment Environment which stores the optimization path. Contents depend on implementation.

Usage

```
makeOptPathDF(
  par.set,
  y.names,
  minimize,
  add.transformed.x = FALSE,
  include.error.message = FALSE,
  include.exec.time = FALSE,
  include.extra = FALSE
)
```

Arguments

par.set	ParamSet Parameter set.
y.names	(character) Names of performance measures that are optimized or logged.
minimize	(logical) Which of the performance measures in y.names should be minimized? Vector of booleans in the same order as y.names.
add.transformed.x	(logical(1)) If some parameters have associated transformations, are you going to add x values after they have been transformed? Default is FALSE.
include.error.message	(logical(1)) Should it be possible to include an error message string (or NA if no error occurred) into the path for each evaluation? This is useful if you have complex, long running objective evaluations that might fail. Default is FALSE.
include.exec.time	(logical(1)) Should it be possible to include execution time of evaluations into the path for each evaluation? Note that execution time could also be entered in y.names as a direct performance measure. If you use this option here, time is regarded as an extra measurement you might be curious about. Default is FALSE.
include.extra	(logical(1)) Should it be possible to include extra info into the path for each evaluation? Default is FALSE.

See Also

Other optpath: addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB(), setOptPathElEOL()

Param *Create a description object for a parameter.*

Description

For each parameter type a special constructor function is available, see below.

For the following arguments you can also pass an `expression` instead of a concrete value: `default`, `len`, `lower`, `upper`, `values`. These expressions can depend on arbitrary symbols, which are later filled in / substituted from a dictionary, in order to produce a concrete valu, see `evaluateParamExpressions()`. So this enables data / context dependent settings, which is sometimes useful.

The S3 class is a list which stores these elements:

id (`character(1)`) See argument of same name.

type (`character(1)`) Data type of parameter. For all type string see (getTypeStringsAll())

len (`integer(1)` | `expression`) See argument of same name.

lower (`numeric` | `expression`) See argument of same name. Length of this vector is `len`.

upper (`numeric` | `expression`) See argument of same name. Length of this vector is `len`.

values (`list` | `expression`) Discrete values, always stored as a named list.

cnames (`character` See argument of same name.

allow.inf (`logical(1)`) See argument of same name.

trafo (`NULL` | `function(x)`) See argument of same name.

requires (`NULL` | `expression`) See argument of same name.

default (**any concrete value** | `expression`) See argument of same name.

has.default (`logical(1)`) Extra flag to really be able to check whether the user passed a default, to avoid troubles with `NULL` and `NA`.

tunable (`logical(1)`) See argument of same name.

special.vals (`list`) See argument of same name.

Usage

```
makeNumericParam(
  id,
  lower = -Inf,
  upper = Inf,
  allow.inf = FALSE,
  default,
```

```
  trafo = NULL,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeNumericVectorParam(
  id,
  len,
  lower = -Inf,
  upper = Inf,
  cnames = NULL,
  allow.inf = FALSE,
  default,
  trafo = NULL,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeIntegerParam(
  id,
  lower = -Inf,
  upper = Inf,
  default,
  trafo = NULL,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeIntegerVectorParam(
  id,
  len,
  lower = -Inf,
  upper = Inf,
  cnames = NULL,
  default,
  trafo = NULL,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeLogicalParam(
  id,
  default,
  requires = NULL,
```

```
  tunable = TRUE,
  special.vals = list()
)

makeLogicalVectorParam(
  id,
  len,
  cnames = NULL,
  default,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeDiscreteParam(
  id,
  values,
  trafo = NULL,
  default,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeDiscreteVectorParam(
  id,
  len,
  values,
  trafo = NULL,
  default,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
)

makeFunctionParam(
  id,
  default = default,
  requires = NULL,
  special.vals = list()
)

makeUntypedParam(
  id,
  default,
  requires = NULL,
  tunable = TRUE,
  special.vals = list()
```

```
)

makeCharacterParam(id, default, requires = NULL, special.vals = list())

makeCharacterVectorParam(
  id,
  len,
  cnames = NULL,
  default,
  requires = NULL,
  special.vals = list()
)
```

Arguments

id	(character(1)) Name of parameter.
lower	(numeric | expression) Lower bounds. A singe value of length 1 is automatically replicated to len for vector parameters. If len = NA you can only pass length-1 scalars. Default is -Inf.
upper	(numeric | expression) Upper bounds. A singe value of length 1 is automatically replicated to len for vector parameters. If len = NA you can only pass length-1 scalars. Default is Inf.
allow.inf	(logical(1)) Allow infinite values for numeric and numericvector params to be feasible settings. Default is FALSE.
default	(any concrete value | expression) Default value used in learner. Note: When this is a discrete parameter make sure to use a VALUE here, not the NAME of the value. If this argument is missing, it means no default value is available.
trafo	(NULL | function(x)) Function to transform parameter. It should be applied to the parameter value before it is, e.g., passed to a corresponding objective function. Function must accept a parameter value as the first argument and return a transformed one. Default is NULL which means no transformation.
requires	(NULL | call | expression) States requirements on other parameters' values, so that setting this parameter only makes sense if its requirements are satisfied (dependent parameter). Can be an object created either with expression or quote, the former type is auto-converted into the later. Only really useful if the parameter is included in a (ParamSet()). Default is NULL which means no requirements.
tunable	(logical(1)) Is this parameter tunable? Defining a parameter to be not-tunable allows to mark arguments like, e.g., "verbose" or other purely technical stuff. Note that this flag is most likely not respected by optimizing procedures unless stated

otherwise. Default is TRUE (except for untyped, function, character and characterVector) which means it is tunable.

special.vals (list())
A list of special values the parameter can except which are outside of the defined range. Default is an empty list.

len (integer(1) | expression)
Length of vector parameter.

cnames (character)
Component names for vector params (except discrete). Every function in this package that creates vector values for such a param, will name that vector with cnames.

values (vector | list | expression)
Possible discrete values. Instead of using a vector of atomic values, you are also allowed to pass a list of quite "complex" R objects, which are used as discrete choices. If you do the latter, the elements must be uniquely named, so that the names can be used as internal representations for the choice.

Value

Param() .

Examples

```
makeNumericParam("x", lower = -1, upper = 1)
makeNumericVectorParam("x", len = 2)
makeDiscreteParam("y", values = c("a", "b"))
makeCharacterParam("z")
```

---

paramValueToString *Convert a value to a string.*

Description

Useful helper for logging. For discrete parameter values always the name of the discrete value is used.

Usage

```
paramValueToString(par, x, show.missing.values = FALSE, num.format = "%.3g")
```

Arguments

par (Param | ParamSet)
Parameter or parameter set.

x (any)
Value for parameter or value for parameter set. In the latter case it must be named list. For discrete parameters their values must be used, not their names.

show.missing.values
: (logical(1))
Display "NA" for parameters, which have no setting, because their requirements are not satisfied (dependent parameters), instead of displaying nothing? Default is FALSE.

num.format
: (character(1))
Number format for output of numeric parameters. See the details section of the manual for base::sprintf() for details.

Value

character(1).

Examples

```
p = makeNumericParam("x")
paramValueToString(p, 1)
paramValueToString(p, 1.2345)
paramValueToString(p, 0.000039)
paramValueToString(p, 8.13402, num.format = "%.2f")

p = makeIntegerVectorParam("x", len = 2)
paramValueToString(p, c(1L, 2L))

p = makeLogicalParam("x")
paramValueToString(p, TRUE)

p = makeDiscreteParam("x", values = list(a = NULL, b = 2))
paramValueToString(p, NULL)

ps = makeParamSet(
  makeNumericVectorParam("x", len = 2L),
  makeDiscreteParam("y", values = list(a = NULL, b = 2))
)
paramValueToString(ps, list(x = c(1, 2), y = NULL))
```

---

plotEAF *Plots attainment functions for data stored in multiple OptPaths.*

Description

Can be used to plot OptPaths where information for bi-objective evaluation was logged for repeated runs of different algorithmic runs. Pretty directly calls eaf::eafplot().

Usage

```
plotEAF(opt.paths, xlim = NULL, ylim = NULL, ...)
```

Arguments

opt.paths	(list) List of list of OptPath objects. First index is the algorithm / major variation in the experiment, second index is the index of the replicated run.
xlim	(numeric(2)) The x limits (x1, x2) of the plot.
ylim	(numeric(2)) The y limits (y1, y2) of the plot.
...	(any) Passed on to eaf::eafplot().

Value

data.frame Invisibly returns the data passed to eaf::eafplot().

Note

We changed the defaults of eaf::eafplot() in the following way: The axis are labeled by y.names, colors are set to our favorite grey values and linetypes changed, too. With our colors / linetypes default it is possible to distinguish 6 different algorithms. But this can again be overwritten by the user.

---

plotOptPath *Plot method for optimization paths.*

Description

Plot method for every type of optimization path, containing any numbers and types of variables. For every iteration up to 4 types of plots can be generated: One plot for the distribution of points in X and Y space respectively and plots for the trend of specified X variables, Y variables and extra measures over the time.

Usage

```
plotOptPath(
  op,
  iters,
  pause = TRUE,
  xlim = list(),
  ylim = list(),
  title = "Optimization Path Plots",
  ...
)
```

Arguments

op	(OptPath) Optimization path.
iters	(integer | NULL) Vector of iterations which should be plotted one after another. If NULL, which is the default, only the last iteration is plotted. Iteration 0 plots all elements with dob = 0. Note that the plots for iteration i contains all observations alive in iteration i.
pause	(logical(1)) Should the process be paused after each iteration? Default is TRUE.
xlim	list X axis limits for the plots. Must be a named list, so you can specify the axis limits for every plot. Every element of the list must be a numeric vector of length 2. Available names for elements are: XSpace - limits for the X-Space plot YSpace - limits for the Y-Space plot Default is an empty list - in this case limits are automatically set. Note: For some plots it is not meaningful to set limits, in this case the set limits are ignored. Note: We do not support setting lims for the over.time.plots. We think, in nearly every case the ggplot defaults are fine, and the very rare case you have to set them, you can you can extract the plots and add your own limits.
ylim	list Y axis limits for the plots. Must be a named list, so you can specify the axis limits for every plot. Every element of the list must be a numeric vector of length 2. Available names for elements are: XSpace - limits for the X-Space plot YSpace - limits for the Y-Space plot Default is an empty list - in this case limits are automatically set. Note: For some plots it is not meaningful to set limits, in this case the set limits are ignored. Note: We do not support setting lims for the over.time.plots. We think, in nearly every case the ggplot defaults are fine, and the very rare case you have to set them, you can you can extract the plots and add your own limits.
title	(character(1)) Main title for the arranged plots, default is Optimization Path Plots.
...	Additional parameters for renderOptPathPlot().

---

plotYTraces *Plots Y traces of multiple optimization paths*

Description

Plot function for renderYTraces()

Usage

```
plotYTraces(opt.paths, over.time = "dob")
```

Arguments

opt.paths	list List of OptPath objects
over.time	character Should the traces be plotted versus the iteration number or the cumulated execution time? For the later, the opt.path has to contain a extra column names exec.time. Possible values are dob and exec.time, default is dob.

Value

NULL

---

removeMissingValues *Removes all scalar NAs from a parameter setting list.*

Description

Removes all scalar NAs from a parameter setting list.

Usage

```
removeMissingValues(x)
```

Arguments

x	list List of parameter values.

Value

list.

---

renderOptPathPlot *Function for plotting optimization paths.*

Description

Same as plotOptPath(), but renders the plots for just 1 iteration and returns a list of plots instead of printing the plot. Useful, if you want to extract single plots or to edit the ggplots by yourself.

Usage

```
renderOptPathPlot(
  op,
  iter,
  x.over.time,
  y.over.time,
  contour.name = NULL,
  xlim = list(),
  ylim = list(),
  alpha = TRUE,
  log = NULL,
  colours = c("red", "blue", "green", "orange"),
  size.points = 3,
  size.lines = 1.5,
  impute.scale = 1,
  impute.value = "missing",
  scale = "std",
  ggplot.theme = ggplot2::theme(legend.position = "top"),
  marked = NULL,
  subset.obs,
  subset.vars,
  subset.targets,
  short.x.names,
  short.y.names,
  short.rest.names
)
```

Arguments

op	OptPath Optimization path.
iter	(integer(1)) Selected iteration of x to render plots for.
x.over.time	(list | NULL) List of vectors of x-variables, either specified via name or id. If specified via names, also extra measurements from the opt.path can be selected. Maximum length for each vector is 5. For each list-element a line-plot iteration versus variable is generated. If the vector has length > 2 only mean values per iteration are plotted as lines, if vector has length 1 every point is plotted. Default is to plot all variables into as few plots as possible. Note that discrete variables are converted to numeric, if specified in the same vector with numerics. Moreover, if more than 1 point per iteration exists, mean values are calculated. This is also done for factor variables! We recommend you to specify this argument in a useful way.
y.over.time	(list | NULL) List of vectors of y-variables, either specified via name or id. If specified via names, also extra measurements from the opt.path can be selected. Maximum

length for each vector is 5. For each list-element a line-plot iteration versus variable is generated. If the vector has length > 2 only mean values per iteration are plotted as lines, if vector has length 1 every point is plotted. Default is to plot all variables into as few plots as possible. Note that discrete variables are converted to numeric, if specified in the same vector with numerics. Moreover, if more than 1 point per iteration exists, mean values are calculated. This is also done for factor variables! We recommend you to specify this argument in a useful way.

`contour.name` (`character(1)` | NULL)
It is possible to overlay the XSpace plot with an contour plot. This is only possible, if the XSpace has exact 2 numeric and 0 discrete variable. Consider subsetting your variables to use this feature! contour.name is the name of the target variable that will be used for the contour lines. Default is to use the first target variable, if it is possible to add contour lines.

`xlim` list
X axis limits for the plots. Must be a named list, so you can specify the axis limits for every plot. Every element of the list must be a numeric vector of length 2. Available names for elements are: XSpace - limits for the X-Space plot YSpace - limits for the Y-Space plot Default is an empty list - in this case limits are automatically set. Note: For some plots it is not meaningful to set limits, in this case the set limits are ignored. Note: We do not support setting lims for the over.time.plots. We think, in nearly every case the ggplot defaults are fine, and the very rare case you have to set them, you can you can extract the plots and add your own limits.

`ylim` list
Y axis limits for the plots. Must be a named list, so you can specify the axis limits for every plot. Every element of the list must be a numeric vector of length 2. Available names for elements are: XSpace - limits for the X-Space plot YSpace - limits for the Y-Space plot Default is an empty list - in this case limits are automatically set. Note: For some plots it is not meaningful to set limits, in this case the set limits are ignored. Note: We do not support setting lims for the over.time.plots. We think, in nearly every case the ggplot defaults are fine, and the very rare case you have to set them, you can you can extract the plots and add your own limits.

`alpha` (`logical(1)`)
Activates or deactivates the alpha fading for the plots. Default is `TRUE`.

`log` character
Vector of variable names. All of this variable logarithmized in every plot. Default is NULL - no logarithm is applied. Note that, if an variable has only negative value, it is multiplied with -1. For variables with both positive and negative values you have to do your own data preprocessing.

`colours` (`character(4)`)
Colours of the points/lines for the four point types init, seq, prob and marked. Default is red for init, blue for seq, green for prob and orange for marked.

`size.points` (`numeric(4)` | NULL)
Size of points in the plot, default is 3.

size.lines (numeric(4) | NULL)
Size of lines in the plots, default is 1.5.

impute.scale (numeric(1))
Numeric missing values will be replaced by max + impute.scale * (max - min). Default is 1.

impute.value (character(1))
Factor missing values will be replaced by impute.value. Default is missing.

scale (character(1))
Parameter scale from the function GGally::ggparcoord() which is used for the multiD-case. Default is std.

ggplot.theme Theme for the ggplots. Can be generated by ggplot2::theme(). Default is ggplot2::theme(legend.position = "top").

marked (integer | character(1) | NULL)
"best" or indices of points that should be marked in the plots. If marked = "best" the best point for single crit optimization respectively the pareto front for multi crit optimization is marked. Default is NULL (no points are marked).

subset.obs integer
Vector of indices to subset of observations to be plotted, default is all observations. All indices must be available in the opt.path. But, to enable subsetting over multiple iterations, not all indices must be available in the current iteration. Indices not available in the current iteration will be ignored. Default is all observations.

subset.vars (integer | character)
Subset of variables (x-variables) to be plotted. Either vector of indices or names. Default is all variables.

subset.targets (integer | character)
Subset of target variables (y-variables) to be plotted. Either vector of indices or names. Default is all variables

short.x.names character
Short names for x variables that are used as axis labels. Note you can only give shortnames for variables you are using in subset.vars

short.y.names character
Short names for y variables that are used as axis labels. Note you can only give shortnames for variables you are using in subset.targets

short.rest.names
character
Short names for rest variables that are used as axis labels. Note you can only give shortnames for variables you are used in x.over.time or y.over.time.

Value

List of plots. List has up to elements: plot.x: Plot for XSpace. If both X and Y are 1D, Plot for both plot.y: Plot for YSpace. If both X and Y are 1D, NULL. plot.x.over.time: List of plots for x over time. Can also be NULL. plot.y.over.time: List of plots for y over time. Can also be NULL.

renderYTraces *Plots Y traces of multiple optimization paths*

Description

Can be used for only single-objective optimization paths. Useful to compare runs of different algorithms on the same optimization problem. You can add your own ggplot layers to the resulting plot object.

Usage

```
renderYTraces(opt.paths, over.time = "dob")
```

Arguments

opt.paths	[OptPath() | list of OptPath()] Object(s) to plot.
over.time	character Should the traces be plotted versus the iteration number or the cumulated execution time? For the later, the opt.path has to contain a extra column names exec.time. Possible values are dob and exec.time, default is dob.

Value

ggplot2 plot object

repairPoint *Repairs values of numeric and integer parameters out side of constraints.*

Description

Clips values outside of box constraints to bounds.

Usage

```
repairPoint(par.set, x, warn = FALSE)
```

Arguments

par.set	ParamSet Parameter set.
x	list List of parameter values. Must be in correct order. Values corresponding to non-numeric/integer types are left unchanged.
warn	(logical(1)) Boolean indicating whether a warning should be printed each time a value is repaired. Default is FALSE.

Value

list: List of repaired points.

`sampleValue` *Sample a random value from a parameter or a parameter set uniformly.*

Description

Sample a random value from a parameter or a parameter set uniformly.

Dependent parameters whose requirements are not satisfied are represented by a scalar NA in the output.

Usage

```
sampleValue(par, discrete.names = FALSE, trafo = FALSE)
```

Arguments

`par`	(Param | ParamSet) Parameter or parameter set.
`discrete.names`	(`logical(1)`) Should names be sampled for discrete parameters or values instead? Default is code FALSE.
`trafo`	(`logical(1)`) Transform all parameters by using theirs respective transformation functions. Default is `FALSE`.

Value

The return type is determined by the type of the parameter. For a set a named list of such values in the correct order is returned.

Examples

```
# bounds are necessary here, can't sample with Inf bounds:
u = makeNumericParam("x", lower = 0, upper = 1)
# returns a random number between 0 and 1:
sampleValue(u)

p = makeDiscreteParam("x", values = c("a", "b", "c"))
# can be either "a", "b" or "c"
sampleValue(p)

p = makeIntegerVectorParam("x", len = 2, lower = 1, upper = 5)
# vector of two random integers between 1 and 5:
sampleValue(p)
```

```
ps = makeParamSet(
  makeNumericParam("x", lower = 1, upper = 10),
  makeIntegerParam("y", lower = 1, upper = 10),
  makeDiscreteParam("z", values = 1:2)
)
sampleValue(ps)
```

`sampleValues` *Sample n random values from a parameter or a parameter set uniformly.*

Description

Sample n random values from a parameter or a parameter set uniformly.

Dependent parameters whose requirements are not satisfied are represented by a scalar NA in the output.

Usage

```
sampleValues(par, n, discrete.names = FALSE, trafo = FALSE)
```

Arguments

`par`	(Param | ParamSet) Parameter or parameter set.
`n`	(`integer(1)`) Number of values.
`discrete.names`	(`logical(1)`) Should names be sampled for discrete parameters or values instead? Default is code FALSE.
`trafo`	(`logical(1)`) Transform all parameters by using theirs respective transformation functions. Default is `FALSE`.

Value

`list`. For consistency always a list is returned.

Examples

```
p = makeIntegerParam("x", lower = -10, upper = 10)
sampleValues(p, 4)

p = makeNumericParam("x", lower = -10, upper = 10)
sampleValues(p, 4)
```

```
p = makeLogicalParam("x")
sampleValues(p, 4)

ps = makeParamSet(
  makeNumericParam("u", lower = 1, upper = 10),
  makeIntegerParam("v", lower = 1, upper = 10),
  makeDiscreteParam("w", values = 1:2)
)
sampleValues(ps, 2)
```

`setOptPathElDOB` *Set the dates of birth of parameter values, in-place.*

Description

Set the dates of birth of parameter values, in-place.

Usage

```
setOptPathElDOB(op, index, dob)
```

Arguments

`op`	OptPath Optimization path.
`index`	integer Vector of indices of elements.
`dob`	integer Dates of birth, single value or same length of `index`.

Value

Nothing.

See Also

Other optpath: `OptPath`, `addOptPathEl()`, `getOptPathBestIndex()`, `getOptPathCols()`, `getOptPathCol()`, `getOptPathDOB()`, `getOptPathEOL()`, `getOptPathEl()`, `getOptPathErrorMessages()`, `getOptPathExecTimes()`, `getOptPathLength()`, `getOptPathParetoFront()`, `getOptPathX()`, `getOptPathY()`, `setOptPathElEOL()`

setOptPathElEOL *Set the end of life dates of parameter values, in-place.*

Description

Set the end of life dates of parameter values, in-place.

Usage

```
setOptPathElEOL(op, index, eol)
```

Arguments

op	OptPath Optimization path.
index	integer Vector of indices of elements.
eol	integer EOL dates, single value or same length of index.

Value

Nothing.

See Also

Other optpath: OptPath, addOptPathEl(), getOptPathBestIndex(), getOptPathCols(), getOptPathCol(), getOptPathDOB(), getOptPathEOL(), getOptPathEl(), getOptPathErrorMessages(), getOptPathExecTimes(), getOptPathLength(), getOptPathParetoFront(), getOptPathX(), getOptPathY(), setOptPathElDOB()

setValueCNames *Set components names for vector names*

Description

If param has cnames set component names in a value. Otherwise x is left unchanged.

Usage

```
setValueCNames(par, x)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.
x	(any) Param value(s). For a parameter set this must be a list in the correct order.

Value

x with changed names.

`trafoOptPath` *Transform optimization path.*

Description

Transform optimization path with associated transformation functions of parameters. Can only be done when x values where added "untransformed".

Usage

```
trafoOptPath(opt.path)
```

Arguments

`opt.path`	[`OptPath()`] Optimization path.

Value

`OptPath()` .

Examples

```
ps = makeParamSet(
  makeIntegerParam("u", trafo = function(x) 2 * x),
  makeNumericVectorParam("v", len = 2, trafo = function(x) x / sum(x)),
  makeDiscreteParam("w", values = c("a", "b"))
)
op = makeOptPathDF(ps, y.names = "y", minimize = TRUE)
addOptPathEl(op, x = list(3, c(2, 4), "a"), y = 0, dob = 1, eol = 1)
addOptPathEl(op, x = list(4, c(5, 3), "b"), y = 2, dob = 5, eol = 7)

as.data.frame(op)
op = trafoOptPath(op)
as.data.frame(op)
```

trafoValue *Transform a value.*

Description

Transform a value with associated transformation function(s).

Usage

```
trafoValue(par, x)
```

Arguments

par	(Param | ParamSet) Parameter or parameter set.
x	(any) Single value to check. For a parameter set this must be a list. If the list is unnamed (not recommended) it must be in the same order as the param set. If it is named, its names must match the parameter names in the param set.

Value

Transformed value.

Examples

```
# transform simple parameter:
p = makeNumericParam(id = "x", trafo = function(x) x^2)
trafoValue(p, 2)
# for a parameter set different transformation functions are possible:
ps = makeParamSet(
  makeIntegerParam("u", trafo = function(x) 2 * x),
  makeNumericVectorParam("v", len = 2, trafo = function(x) x / sum(x)),
  makeDiscreteParam("w", values = c("a", "b"))
)
# now the values of "u" and "v" are transformed:
trafoValue(ps, list(3, c(2, 4), "a"))
```

updateParVals *Insert par.vals to old ones with meeting requirements*

Description

Update the values of a given parameter setting with a new parameter setting. Settings that do not meet the requirements anymore will be deleted from the first given parameter setting. Default values of the Param Set are respected to check if the new param settings meet the requirements.

Usage

```
updateParVals(par.set, old.par.vals, new.par.vals, warn = FALSE)
```

Arguments

`par.set`	ParamSet Parameter set.
`old.par.vals`	list Param Values to be updated.
`new.par.vals`	list New Param Values to update the `old.par.vals`.
`warn`	logical Whether a warning should be shown, if a param setting from `old.par.vals` is dropped. Default is `FALSE`.

Value

`list`.

Index

∗ **optpath**
addOptPathEl, 3
getOptPathBestIndex, 21
getOptPathCol, 22
getOptPathCols, 23
getOptPathDOB, 24
getOptPathEl, 24
getOptPathEOL, 25
getOptPathErrorMessages, 26
getOptPathExecTimes, 27
getOptPathLength, 27
getOptPathParetoFront, 28
getOptPathX, 29
getOptPathY, 30
OptPath, 50
setOptPathElDOB, 67
setOptPathElEOL, 68

addOptPathEl, 3, *21–30*, *52*, *67, 68*
addOptPathEl(), *50*
any, *9*
as.data.frame.OptPathDF, 5

base::sprintf(), *57*
BBmisc::convertDataFrameCols(), *5*, *8*, *13*, *16, 17*

character, *6*, *9*, *23*, *26*, *28*, *30, 31*, *34*, *36*, *49*, *51*, *60*, *62–64*
checkParamSet, 6
convertParamSetToIrace, 7

data.frame, *6*, *14*, *16–18*, *23*, *29*, *58*
dfRowsToList, 7
dfRowToList *(*dfRowsToList*)*, 7
discreteNameToValue, 8
discreteValueToName, 9
dropParams, 10

eaf::eafplot(), *57, 58*
environment, *51*
evaluateParamExpressions, 10
evaluateParamExpressions(), *52*

filterParams, 11
filterParamsDiscrete *(*filterParams*)*, 11
filterParamsNumeric *(*filterParams*)*, 11

generateDesign, 13
generateDesign(), *15*
generateDesignOfDefaults, 15
generateGridDesign, 16
generateGridDesign(), *15*
generateRandomDesign, 17
generateRandomDesign(), *15*
getDefaults, 18
getLower, 19
getOptPathBestIndex, *4*, 21, *22–30*, *52*, *67, 68*
getOptPathCol, *4*, *21*, 22, *23–30*, *52*, *67, 68*
getOptPathCols, *4*, *21, 22*, 23, *24–30*, *52*, *67, 68*
getOptPathDOB, *4*, *21–23*, 24, *25–30*, *52*, *67, 68*
getOptPathEl, *4*, *21–24*, 24, *26–30*, *52*, *67, 68*
getOptPathEl(), *50*
getOptPathEOL, *4*, *21–25*, 25, *26–30*, *52*, *67, 68*
getOptPathErrorMessages, *4*, *21–26*, 26, *27–30*, *52*, *67, 68*
getOptPathExecTimes, *4*, *21–26*, 27, *28–30*, *52*, *67, 68*
getOptPathLength, *4*, *21–27*, 27, *29, 30*, *52*, *67, 68*
getOptPathParetoFront, *4*, *21–28*, 28, *30*, *52*, *67, 68*
getOptPathX, *4*, *21–29*, 29, *30*, *52*, *67, 68*
getOptPathY, *4*, *21–30*, 30, *52*, *67, 68*
getParamIds, 31
getParamLengths, 31
getParamNr, 32

getParamSet, 33
getParamTypeCounts, 33
getParamTypes, 34
getRequirements, 35
getTypeStrings, 35
getTypeStringsAll (getTypeStrings), 35
getTypeStringsCharacter (getTypeStrings), 35
getTypeStringsDiscrete (getTypeStrings), 35
getTypeStringsInteger (getTypeStrings), 35
getTypeStringsLogical (getTypeStrings), 35
getTypeStringsNumeric (getTypeStrings), 35
getTypeStringsNumericStrict (getTypeStrings), 35
getUpper (getLower), 19
getValues (getLower), 19
GGally::ggparcoord(), *63*
ggplot2::theme(), *63*

hasCharacter (hasType), 39
hasDiscrete (hasType), 39
hasExpression, 36
hasFiniteBoxConstraints, 37
hasForbidden, 37
hasInteger (hasType), 39
hasLogical (hasType), 39
hasNumeric (hasType), 39
hasRequires, 38
hasTrafo, 38
hasType, 39

integer, *6*, *21–30*, *32*, *63*, *67*, *68*
irace::readParameters(), *7*
isCharacter (isType), 43
isCharacterTypeString (isTypeString), 43
isDiscrete (isType), 43
isDiscreteTypeString (isTypeString), 43
isEmpty, 39
isFeasible, 40
isForbidden, 41
isInteger (isType), 43
isIntegerTypeString (isTypeString), 43
isLogical (isType), 43
isLogicalTypeString (isTypeString), 43
isNumeric (isType), 43
isNumericTypeString (isTypeString), 43
isRequiresOk, 41
isSpecialValue, 42
isType, 43
isTypeString, 43
isVector, 44
isVectorTypeString (isTypeString), 43

LearnerParam, 45
LearnerParam(), *48*
lhs::geneticLHS(), *14*
lhs::improvedLHS(), *14*
lhs::maximinLHS(), *14*
lhs::optAugmentLHS(), *14*
lhs::optimumLHS(), *14*
lhs::randomLHS(), *13*, *14*
list, *6*, *8*, *10*, *19*, *20*, *32*, *33*, *37*, *59*, *60*, *62*, *64*, *65*, *71*
list(), *7*
listToDfOneRow, 48
logical, *51*, *71*

makeCharacterParam (Param), 52
makeCharacterVectorParam (Param), 52
makeDiscreteLearnerParam (LearnerParam), 45
makeDiscreteParam (Param), 52
makeDiscreteVectorLearnerParam (LearnerParam), 45
makeDiscreteVectorParam (Param), 52
makeFunctionLearnerParam (LearnerParam), 45
makeFunctionParam (Param), 52
makeIntegerLearnerParam (LearnerParam), 45
makeIntegerParam (Param), 52
makeIntegerVectorLearnerParam (LearnerParam), 45
makeIntegerVectorParam (Param), 52
makeLogicalLearnerParam (LearnerParam), 45
makeLogicalParam (Param), 52
makeLogicalVectorLearnerParam (LearnerParam), 45
makeLogicalVectorParam (Param), 52
makeNumericLearnerParam (LearnerParam), 45
makeNumericParam (Param), 52
makeNumericParamSet (makeParamSet), 49

makeNumericVectorLearnerParam (LearnerParam), 45
makeNumericVectorParam (Param), 52
makeOptPathDF (OptPath), 50
makeParamSet, 49
makeUntypedLearnerParam (LearnerParam), 45
makeUntypedParam (Param), 52

NULL, *60*
numeric, *27*, *50*

OptPath, *4*, *21–30*, 50, *61*, *67, 68*
OptPath(), *6*, *25*, *64*, *69*
OptPathDF (OptPath), 50
OptPathDF(), *8*

Param, *9*, *31, 32*, *37, 38*, *40*, *42–44*, 52, *56*, *65, 66*, *68*, *70*
Param(), *10, 11*, *18*, *20*, *36*, *40*, *45*, *49*, *56*
ParamHelpers::ParamSet(), *10, 11*, *33*, *36*
ParamSet, *6–8*, *10*, *12*, *14, 15*, *17, 18*, *31–35*, *37–44*, *51*, *56*, *64–66*, *68*, *70, 71*
ParamSet (makeParamSet), 49
ParamSet(), *10*, *12*, *18–20*, *32*, *40*, *50, 51*
paramValueToString, 56
plotEAF, 57
plotOptPath, 58
plotOptPath(), *60*
plotYTraces, 59

readParameters, *7*
removeMissingValues, 60
renderOptPathPlot, 60
renderOptPathPlot(), *59*
renderYTraces, 64
renderYTraces(), *59*
repairPoint, 64

sampleValue, 65
sampleValues, 66
sampleValues(), *17*
setOptPathElDOB, *4*, *21–30*, *52*, 67, *68*
setOptPathElEOL, *4*, *21–30*, *52*, *67*, 68
setValueCNames, 68

trafoOptPath, 69
trafoOptPath(), *50*
trafoValue, 70
TRUE, *7*

updateParVals, 70