Package ‘smoof’

January 16, 2020

Type Package

Title Single and Multi-Objective Optimization Test Functions

Description Provides generators for a high number of both single- and multi-objective test functions which are frequently used for the benchmarking of (numerical) optimization algorithms. Moreover, it offers a set of convenient functions to generate, plot and work with objective functions.

Version 1.6.0

Date 2020-01-07

URL https://github.com/jakobbossek/smoof

BugReports https://github.com/jakobbossek/smoof/issues

License BSD_2_clause + file LICENSE

Depends ParamHelpers (>= 1.8), checkmate (>= 1.1)

Imports BBmisc (>= 1.6), ggplot2 (>= 2.2.1), RColorBrewer, plot3D, plotly, mco, Rcpp (>= 0.11.0), RJSONIO

Suggests testthat, rPython (>= 0.0-5)

LazyData yes

ByteCompile yes

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 6.1.1

NeedsCompilation yes

Author Jakob Bossek [aut, cre], Pascal Kerschke [ctb]

Maintainer Jakob Bossek <j.bossek@gmail.com>

Repository CRAN

Date/Publication 2020-01-16 13:40:02 UTC
R topics documented:

- smoof-package
- addCountingWrapper
- addLoggingWrapper
- autoplot.smoof_function
- computeExpectedRunningTime
- conversion
- doesCountEvaluations
- filterFunctionsByTags
- getAvailableTags
- getDescription
- getGlobalOptimum
-getID
- getLocalOptimum
- getLoggedValues
- getLowerBoxConstraints
- getMeanFunction
- getName
- getNumberOfEvaluations
- getNumberOfObjectives
- getNumberOfParameters
- getParamSet
- getRefPoint
- getTags
- getUpperBoxConstraints
- getWrappedFunction
- hasBoxConstraints
- hasConstraints
- hasGlobalOptimum
- hasLocalOptimum
- hasOtherConstraints
- hasTags
- isMultiobjective
- isNoisy
- isSingleobjective
- isSmoofFunction
- isVectorized
- isWrappedSmoofFunction
- makeAckleyFunction
- makeAdjimanFunction
- makeAlpine01Function
- makeAlpine02Function
- makeAluffiPentiniFunction
- makeBartelsConnFunction
- makeBBOBFunction
- makeBealeFunction
- makeBentCigarFunction
makeBiObjBBOBFunction .................................................. 34
makeBirdFunction ............................................................ 35
makeBiSphereFunction ....................................................... 35
makeBK1Function ............................................................... 36
makeBohachevskyN1Function .................................................. 36
makeBoothFunction ............................................................ 37
makeBrinFunction ............................................................... 37
makeBrentFunction ............................................................. 38
makeBrownFunction ............................................................ 39
makeBukinN2Function ........................................................... 39
makeBukinN4Function ........................................................... 40
makeBukinN6Function ........................................................... 41
makeCarromTableFunction ..................................................... 41
makeChichinadzeFunction ...................................................... 42
makeChungReynoldsFunction .................................................. 42
makeComplexFunction .......................................................... 43
makeCosineMixtureFunction .................................................. 43
makeCrossInTrayFunction ...................................................... 44
makeCubeFunction .............................................................. 45
makeDeckersAartsFunction .................................................... 45
makeDeflectedCorrugatedSpringFunction ....................................... 46
makeDentFunction .............................................................. 46
makeDixonPriceFunction ....................................................... 47
makeDoubleSumFunction ....................................................... 48
makeDTLZ1Function ............................................................ 48
makeDTLZ2Function ............................................................ 49
makeDTLZ3Function ............................................................ 50
makeDTLZ4Function ............................................................ 51
makeDTLZ5Function ............................................................ 52
makeDTLZ6Function ............................................................ 53
makeDTLZ7Function ............................................................ 54
makeEasomFunction ............................................................. 55
makeED1Function ............................................................... 56
makeED2Function ............................................................... 57
makeEggCrateFunction .......................................................... 58
makeEggholderFunction ........................................................ 58
makeElAttarVidyasagarDuttaFunction .......................................... 59
makeEngvallFunction ........................................................... 59
makeExponentialFunction ...................................................... 60
makeFreudensteinRothFunction ............................................... 60
makeFunctionsByName ........................................................ 61
makeGeneralizedDropWaveFunction .......................................... 62
makeGiuntaFunction ............................................................ 62
makeGoldsteinPriceFunction ................................................. 63
makeGOMOPFunction .......................................................... 63
makeGriewankFunction ........................................................ 64
makeHansenFunction ........................................................... 65
makeHartmannFunction ........................................................ 65
smoof-package

makeWFG2Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99
makeWFG3Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
makeWFG4Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101
makeWFG5Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
makeWFG6Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103
makeWFG7Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104
makeWFG8Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 105
makeWFG9Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 106
makeZDT1Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107
makeZDT2Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108
makeZDT3Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 109
makeZDT4Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110
makeZDT6Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111
makeZettlFunction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112
mnof . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112
plot.smoof_function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114
plot1DNumeric . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
plot2DNumeric . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
plot3D . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
resetEvaluationCounter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
shouldBeMinimized . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
smoof_function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
snof . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
violatesConstraints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
visualizeParetoOptimalFront . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122

Index 123

smoof-package smoof: Single and Multi-Objective Optimization test functions.

Description

The smoof R package provides generators for huge set of single- and multi-objective test functions, which are frequently used in the literature to benchmark optimization algorithms. Moreover the package provides methods to create arbitrary objective functions in an object-orientated manner, extract their parameters sets and visualize them graphically.

Some more details

Given a set of criteria \( F = \{f_1, \ldots, f_m\} \) with each \( f_i : S \subseteq \mathbb{R}^d \to \mathbb{R}, i = 1, \ldots, m \) being an objective-function, the goal in Global Optimization (GO) is to find the best solution \( x^* \in S \). The set \( S \) is termed the set of feasible solutions. In the case of only a single objective function \( f \), which we want to restrict ourself in this brief description - the goal is to minimize the objective, i.e.,

\[
\min_x f(x).
\]
Sometimes we may be interested in maximizing the objective function value, but since \( \min(f(x)) = -\min(-f(x)) \), we do not have to tackle this separately. To compare the robustness of optimization algorithms and to investigate their behaviour in different contexts, a common approach in the literature is to use artificial benchmarking functions, which are mostly deterministic, easy to evaluate and given by a closed mathematical formula. A recent survey by Jamil and Yang lists 175 single-objective benchmarking functions in total for global optimization [1]. The smoof package offers implementations of a subset of these functions beside some other functions as well as generators for large benchmarking sets like the noiseless BBOB2009 function set [2] or functions based on the multiple peaks model 2 [3].

References


addCountingWrapper

Return a function which counts its function evaluations.

Description

This is a counting wrapper for a smoof_function, i.e., the returned function first checks whether the given argument is a vector or matrix, saves the number of function evaluations of the wrapped function to compute the function values and finally passes down the argument to the wrapped smoof_function.

Usage

addCountingWrapper(fn)

Arguments

fn

[smoof_function]
Smoof function which should be wrapped.

Value

smoof_counting_function

See Also

gNumberFormatException, resetEvaluationCounter
addLoggingWrapper

Examples

```r
fn = makeBBOBFunction(dimensions = 2L, fid = 1L, iid = 1L)
fn = addCountingWrapper(fn)

# we get a value of 0 since the function has not been called yet
print(getNumberOfEvaluations(fn))

# now call the function 10 times consecutively
for (i in seq(10L)) {
  fn(runif(2))
}
print(getNumberOfEvaluations(fn))

# Here we pass a (2x5) matrix to the function with each column representing
# one input vector
x = matrix(runif(10), ncol = 5L)
fn(x)
print(getNumberOfEvaluations(fn))
```

---

addLoggingWrapper  
Return a function which internally stores x or y values.

Description

Often it is desired and useful to store the optimization path, i.e., the evaluated function values and/or the parameters. Not all optimization algorithms offer such a trace. This wrapper makes a smoof function handle x/y-values itself.

Usage

```r
addLoggingWrapper(fn, logg.x = FALSE, logg.y = TRUE)
```

Arguments

- **fn**  
  [smoof_function]  
  Smoof function.

- **logg.x**  
  [logical(1)]  
  Should x-values be logged? Default is FALSE.

- **logg.y**  
  [logical(1)]  
  Should objective values be logged? Default is TRUE.

Value

smoof_logging_function

Note

Logging values, in particular logging x-values, will substantially slow down the evaluation of the function.
Examples

```r
# We first build the smoof function and apply the logging wrapper to it
fn = makeSphereFunction(dimensions = 2L)
fn = addLoggingWrapper(fn, logg.x = TRUE)

# We now apply an optimization algorithm to it and the logging wrapper keeps
# track of the evaluated points.
res = optim(fn, par = c(1, 1), method = "Nelder-Mead")

# Extract the logged values
log.res = getLoggedValues(fn)
print(log.res$pars)
print(log.res$obj.vals)
log.res = getLoggedValues(fn, compact = TRUE)
print(log.res)
```

---

**autoplot.smoof_function**

*Generate ggplot2 object.*

**Description**

This function expects a smoof function and returns a ggplot object depicting the function landscape. The output depends highly on the decision space of the smoof function or more technically on the `ParamSet` of the function. The following distinctions regarding the parameter types are made. In case of a single numeric parameter a simple line plot is drawn. For two numeric parameters or a single numeric vector parameter of length 2 either a contour plot or a heatmap (or a combination of both depending on the choice of additional parameters) is depicted. If there are both up to two numeric and at least one discrete vector parameter, ggplot facetting is used to generate subplots of the above-mentioned types for all combinations of discrete parameters.

**Usage**

```r
## S3 method for class 'smoof_function'
autoplot(x, show.optimum = FALSE,
         main = getName(x), render.levels = FALSE, render.contours = TRUE,
         log.scale = FALSE, length.out = 50L, ...)
```

**Arguments**

- `x` [smoof_function]
  Objective function.
- `show.optimum` [logical(1)]
  If the function has a known global optimum, should its location be plotted by a point or multiple points in case of multiple global optima? Default is FALSE.
- `main` [character(1L)]
  Plot title. Default is the name of the smoof function.
autoplot.smoof_function

render.levels [logical(1)]
For 2D numeric functions only: Should an image map be plotted? Default is FALSE.

render.contours [logical(1)]
For 2D numeric functions only: Should contour lines be plotted? Default is TRUE.

log.scale [logical(1)]
Should the z-axis be plotted on log-scale? Default is FALSE.

length.out [integer(1)]
Desired length of the sequence of equidistant values generated for numeric parameters. Higher values lead to more smooth resolution in particular if render.levels is TRUE. Avoid using a very high value here especially if the function at hand has many parameters. Default is 50.

... [any]
Not used.

Value
ggplot

Note

Keep in mind, that the plots for mixed parameter spaces may be very large and computationally expensive if the number of possible discrete parameter values is large. I.e., if we have d discrete parameter with each n_1, n_2, ..., n_d possible values we end up with n_1 x n_2 x ... x n_d subplots.

Examples

library(ggplot2)

# Simple 2D contour plot with activated heatmap for the Himmelblau function
fn = makeHimmelblauFunction()
print(autoplot(fn))
print(autoplot(fn, render.levels = TRUE, render.contours = FALSE))
print(autoplot(fn, show.optimum = TRUE))

# Now we create 4D function with a mixed decision space (two numeric, one discrete, # and one logical parameter)
fn.mixed = makeSingleObjectiveFunction(
  name = "4d S00 function",
  fn = function(x) {
    if (x$disc1 == "a") {
      (x$x1^2 + x$x2^2) + 10 * as.numeric(x$logic)
    } else {
      x$x1 + x$x2 - 10 * as.numeric(x$logic)
    }},
  has.simple.signature = FALSE,
  par.set = makeParamSet(}
```r
makeNumericParam("x1", lower = -5, upper = 5),
makeNumericParam("x2", lower = -3, upper = 3),
makeDiscreteParam("disc1", values = c("a", "b")),
makeLogicalParam("logic")
)
)
pl = autoplot(fn.mixed)
print(pl)

# Since autoplot returns a ggplot object we can modify it, e.g., add a title
# or hide the legend
pl + ggtitle("My fancy function") + theme(legend.position = "none")
```

`computeExpectedRunningTime`

*Compute the Expected Running Time (ERT) performance measure.*

**Description**

The functions can be called in two different ways:

- 1. Pass a vector of function evaluations and a logical vector which indicates which runs were successful (see details).
- 2. Pass a vector of function evaluation, a vector of reached target values and a single target value. In this case the logical vector of option 1. is computed internally.

**Usage**

```r
computeExpectedRunningTime(fun.evals, fun.success.runs = NULL,
fun.reached.target.values = NULL, fun.target.value = NULL,
penalty.value = Inf)
```

**Arguments**

- `fun.evals` [numeric]: Vector containing the number of function evaluations.
- `fun.success.runs` [logical]: Boolean vector indicating which algorithm runs were successful, i.e., which runs reached the desired target value. Default is NULL.
- `fun.reached.target.values` [numeric | NULL]: Numeric vector with the objective values reached in the runs. Default is NULL.
- `fun.target.value` [numeric(1) | NULL]: Target value which shall be reached. Default is NULL.
- `penalty.value` [numeric(1)]: Penalty value which should be returned if none of the algorithm runs was successful. Default is Inf.
Details

The Expected Running Time (ERT) is one of the most popular performance measures in optimization. It is defined as the expected number of function evaluations needed to reach a given precision level, i.e., to reach a certain objective value for the first time.

Value

numeric(1) Estimated Expected Running Time.

References


Description

We can minimize $f$ by maximizing $-f$. The majority of predefined objective functions in smoof should be minimized by default. However, there is a handful of functions, e.g., Keane or Alpine02, which shall be maximized by default. For benchmarking studies it might be beneficial to inverse the direction. The functions convertToMaximization and convertToMinimization do exactly that keeping the attributes.

Usage

convertToMaximization(fn)
convertToMinimization(fn)

Arguments

fn [smoof_function]
Smoof function.

Value

smoof_function

Note

Both functions will quit with an error if multi-objective functions are passed.
Examples

# create a function which should be minimized by default
fn = makeSphereFunction(1L)
print(shouldBeMinimized(fn))
# Now invert the objective direction ...
fn2 = convertToMaximization(fn)
# and invert it again
fn3 = convertToMinimization(fn2)
# Now to convince ourselves we render some plots
opar = par(mfrow = c(1, 3))
plot(fn)
plot(fn2)
plot(fn3)
par(opar)

---

doesCountEvaluations  Check whether the function is counting its function evaluations.

Description

In this case the function is of type smoof_counting_function or it is further wrapped by another wrapper. This function then checks recursively, if there is a counting wrapper.

Usage

doesCountEvaluations(object)

Arguments

object  [any]
  Arbitrary R object.

Value

logical(1)

See Also

addCountingWrapper
**filterFunctionsByTags**

*Get a list of implemented test functions with specific tags.*

**Description**

Single objective functions can be tagged, e.g., as unimodal. Searching for all functions with a specific tag by hand is tedious. The `filterFunctionsByTags` function helps to filter all single objective smoof function.

**Usage**

```
filterFunctionsByTags(tags, or = FALSE)
```

**Arguments**

- `tags` [character]
  Character vector of tags. All available tags can be determined with a call to `getAvailableTags`.
- `or` [logical(1)]
  Should all tags be assigned to the function or are single tags allowed as well? Default is FALSE.

**Value**

character Named vector of function names with the given tags.

**Examples**

```r
# list all functions which are unimodal
filterFunctionsByTags("unimodal")
# list all functions which are both unimodal and separable
filterFunctionsByTags(c("unimodal", "separable"))
# list all functions which are unimodal or separable
filterFunctionsByTags(c("multimodal", "separable"), or = TRUE)
```

**getAvailableTags**

*Returns a character vector of possible function tags.*

**Description**

Test function are frequently distinguished by characteristic high-level properties, e.g., unimodal or multimodal, continuous or discontinuous, separable or non-separable. The smoof package offers the possibility to associate a set of properties, termed “tags” to a smoof_function. This helper function returns a character vector of all possible tags.
Usage

getAvailableTags()

Value

character

getDescription

Return the description of the function.

Description

Return the description of the function.

Usage

getDescription(fn)

Arguments

fn [smoof_function]
Objective function.

Value

character(1)

getGlobalOptimum

Returns the global optimum and its value.

Description

Returns the global optimum and its value.

Usage

globalOptimum(fn)

Arguments

fn [smoof_function]
Objective function.
**getValue**

**Value**

List containing the following entries:

- **param [list]** Named list of parameter value(s).
- **value [numeric(1)]** Optimal value.
- **is.minimum [logical(1)]** Is the global optimum a minimum or maximum?

**Note**

Keep in mind, that this method makes sense only for single-objective target function.

---

**getID**

*Return the ID / short name of the function.*

**Description**

Return the ID / short name of the function.

**Usage**

```r
getID(fn)
```

**Arguments**

- **fn [smoof_function]** Objective function.

**Value**

character(1)

---

**getLocalOptimum**

*Returns the local optima of a single objective smoof function.*

**Description**

This function returns the parameters and objective values of all local optima (including the global one).

**Usage**

```r
getLocalOptimum(fn)
```
Arguments

fn [Wrapped smoof function]
Objective function.

Value

list List containing the following entries:

• param [list] List of parameter values per local optima.
• value [list] List of objective values per local optima.
• is.minimum [logical(1)] Are the local optima minima or maxima?

Note

Keep in mind, that this method makes sense only for single-objective target functions.

getLoggedValues

Extract logged values of a function wrapped by a logging wrapper.

Description

Extract logged values of a function wrapped by a logging wrapper.

Usage

getLoggedValues(fn, compact = FALSE)

Arguments

fn [wrapped_smoof_function]
Wrapped smoof function.
compact [logical(1)]
Wrap all logged values in a single data frame? Default is FALSE.

Value

list || data.frame If compact is TRUE, a single data frame. Otherwise the function returns a list containing the following values:

pars Data frame of parameter values, i.e., x-values or the empty data frame if x-values were not logged.
obj.vals Numeric vector of objective vals in the single-objective case respectively a matrix of objective vals for multi-objective functions.

See Also

addLoggingWrapper
getLowerBoxConstraints

Return lower box constraints.

Description

Return lower box constraints.

Usage

getLowerBoxConstraints(fn)

Arguments

fn [smoof_function]
Objective function.

Value

numeric

gMeanFunction

Return the true mean function in the noisy case.

Description

Return the true mean function in the noisy case.

Usage

gMeanFunction(fn)

Arguments

fn [smoof_function]
Objective function.

Value

function
**getName**

*Return the name of the function.*

**Description**

Return the name of the function.

**Usage**

```
getName(fn)
```

**Arguments**

- `fn`  
  [smoof_function]
  Objective function.

**Value**

character(1)

---

**getNumberOfEvaluations**

*Return the number of function evaluations performed by the wrapped smoof_function.*

**Description**

Return the number of function evaluations performed by the wrapped smoof_function.

**Usage**

```
getNumberOfEvaluations(fn)
```

**Arguments**

- `fn`  
  [smoof_counting_function]
  Wrapped smoof_function.

**Value**

integer(1)
**getNumberOfObjectives**

Determine the number of objectives.

**Usage**

```plaintext
getNumberOfObjectives(fn)
```

**Arguments**

- `fn` [smoof_function]
  
  Objective function.

**Value**

integer(1)

---

**getNumberOfParameters**

Determine the number of parameters.

**Description**

Determine the number of parameters.

**Usage**

```plaintext
getNumberOfParameters(fn)
```

**Arguments**

- `fn` [smoof_function]
  
  Objective function.

**Value**

integer(1)
**getParamSet**  
*Get parameter set.*

**Description**

Each smoof function contains a parameter set of type `ParamSet` assigned to it, which describes types and bounds of the function parameters. This function returns the parameter set.

**Arguments**

- `fn` [smoof_function]  
  Objective function.

**Value**

`ParamSet`

**Examples**

```r
fn = makeSphereFunction(3L)
ps = getParamSet(fn)
print(ps)
```

---

**getRefPoint**  
*Returns the reference point of a multi-objective function.*

**Description**

Returns the reference point of a multi-objective function.

**Usage**

`getRefPoint(fn)`

**Arguments**

- `fn` [smoof_function]  
  Objective function.

**Value**

`numeric`

**Note**

Keep in mind, that this method makes sense only for multi-objective target functions.
getTags

Returns vector of associated tags.

Description

Returns vector of associated tags.

Usage

getTags(fn)

Arguments

fn [smoof_function]
Objective function.

Value

character

getUpperBoxConstraints

Return upper box constraints.

Description

Return upper box constraints.

Usage

getUpperBoxConstraints(fn)

Arguments

fn [smoof_function]
Objective function.

Value

numeric
getWrappedFunction  

Extract wrapped function.

Description

The smoof package offers means to let a function log its evaluations or even store to points and function values it has been evaluated on. This is done by wrapping the function with other functions. This helper function extract the wrapped function.

Usage

getWrappedFunction(fn, deepest = FALSE)

Arguments


deepest [logical(1)] Function may be wrapped with multiple wrappers. If deepest is set to TRUE the function unwraps recursively until the “deepest” wrapped smoof_function is reached. Default is FALSE.

Value

function

Note

If this function is applied to a simple smoof_function, the smoof_function itself is returned.

See Also

addCountingWrapper, addLoggingWrapper

hasBoxConstraints  

Checks whether the objective function has box constraints.

Description

Checks whether the objective function has box constraints.

Usage

hasBoxConstraints(fn)
**hasConstraints**

**Arguments**

- **fn** [smoof_function]
  Objective function.

**Value**

logical(1)

**Description**

Checks whether the objective function has constraints.

**Usage**

hasConstraints(fn)

**Arguments**

- **fn** [smoof_function]
  Objective function.

**Value**

logical(1)

**hasGlobalOptimum**

**Description**

Checks whether global optimum is known.

**Usage**

hasGlobalOptimum(fn)

**Arguments**

- **fn** [smoof_function]
  Objective function.

**Value**

logical(1)
hasLocalOptimum  Checks whether local optima are known.

Description

Checks whether local optima are known.

Usage

hasLocalOptimum(fn)

Arguments

fn  [smoof_function]
Objective function.

Value

logical(1)

hasOtherConstraints  Checks whether the objective function has other constraints.

Description

Checks whether the objective function has other constraints.

Usage

hasOtherConstraints(fn)

Arguments

fn  [smoof_function]
Objective function.

Value

logical(1)
hasTags

Check if function has assigned special tags.

Description
Each single-objective smoof function has tags assigned to it (see `getAvailableTags`). This little helper returns a vector of assigned tags from a smoof function.

Usage

```r
hasTags(fn, tags)
```

Arguments

- `fn` [smoof_function] Function of smoof_function, a smoof_generator or a string.
- `tags` [character] Vector of tags/properties to check fn for.

Value

`logical(1)`

---

isMultiobjective

Checks whether the given function is multi-objective.

Description
Checks whether the given function is multi-objective.

Usage

```r
isMultiobjective(fn)
```

Arguments

- `fn` [smoof_function] Objective function.

Value

`logical(1)` TRUE if function is multi-objective.
isNoisy  Checks whether the given function is noisy.

Description
Checks whether the given function is noisy.

Usage
isNoisy(fn)

Arguments
fn [smoof_function]
Objective function.

Value
logical(1)

isSingleobjective  Checks whether the given function is single-objective.

Description
Checks whether the given function is single-objective.

Usage
isSingleobjective(fn)

Arguments
fn [smoof_function]
Objective function.

Value
logical(1)  TRUE if function is single-objective.
**isSmoofFunction**

Checks whether the given object is a `smoof_function` or a `smoof_wrapped_function`.

**Description**

Checks whether the given object is a `smoof_function` or a `smoof_wrapped_function`.

**Usage**

```r
isSmoofFunction(object)
```

**Arguments**

- `object` [any]
  - Arbitrary R object.

**Value**

`logical(1)`

**See Also**

- `addCountingWrapper`, `addLoggingWrapper`

---

**isVectorized**

Checks whether the given function accept “vectorized” input.

**Description**

Checks whether the given function accept “vectorized” input.

**Usage**

```r
isVectorized(fn)
```

**Arguments**

- `fn` [smoof_function]
  - Objective function.

**Value**

`logical(1)`
**isWrappedSmoofFunction**

*Checks whether the function is of type smoof_wrapped_function.*

**Description**

Checks whether the function is of type smoof_wrapped_function.

**Usage**

```r
isWrappedSmoofFunction(object)
```

**Arguments**

- `object` [any]
  
  Arbitrary R object.

**Value**

`logical(1)`

---

**makeAckleyFunction**

*Ackley Function*

**Description**

Also known as “Ackley’s Path Function”. Multimodal test function with its global optimum in the center of the definition space. The implementation is based on the formula

\[
  f(x) = -a \cdot \exp\left(-b \cdot \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(c \cdot x_i)\right),
\]

with \(a = 20\), \(b = 0.2\) and \(c = 2\pi\). The feasible region is given by the box constraints \(x_i \in [-32.768, 32.768]\).

**Usage**

```r
makeAckleyFunction(dimensions)
```

**Arguments**

- `dimensions` [integer(1)]
  
  Size of corresponding parameter space.
makeAdjimanFunction

Value

smoof_single_objective_function

References


makeAdjimanFunction   Adjiman function

Description

This two-dimensional multimodal test function follows the formula

\[ f(x) = \cos(x_1)\sin(x_2) - \frac{x_1}{(x_2^2 + 1)} \]

with \( x_1 \in [-1, 2], x_2 \in [2, 1] \).

Usage

makeAdjimanFunction()

Value

smoof_single_objective_function

References


makeAlpine01Function   Alpine01 function

Description

Highly multimodal single-objective optimization test function. It is defined as

\[ f(x) = \sum_{i=1}^{n} \left| x_i \sin(x_i) + 0.1x_i \right| \]

with box constraints \( x_i \in [-10, 10] \) for \( i = 1, \ldots, n \).
Usage

makeAlpine02Function(dimensions)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

smoof_single_objective_function

References


makeAlpine02Function  Alpine02 function

Description

Another multimodal optimization test function. The implementation is based on the formula

\[ f(x) = \prod_{i=1}^{n} \sqrt{x_i} \sin(x_i) \]

with \( x_i \in [0, 10] \) for \( i = 1, \ldots, n \).

Usage

makeAlpine02Function(dimensions)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

smoof_single_objective_function

References

makeAluffiPentiniFunction

Description

Two-dimensional test function based on the formula

\[ f(x) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2 \]

with \( x_1, x_2 \in [-10, 10] \).

Usage

makeAluffiPentiniFunction()

Value

smoof_single_objective_function

Note

This function is also known as the Zirilli function.

References

See http://al-roomi.org/benchmarks/unconstrained/2-dimensions/26-aluffi-pentini-s-or-zirilli-s-function

makeBartelsConnFunction

Bartels Conn Function

Description

The Bartels Conn Function is defined as

\[ f(x) = |x_1^2 + x_2^2 + x_1x_2| + |\sin(x_1)| + |\cos(x)| \]

subject to \( x_i \in [-500, 500] \) for \( i = 1, 2 \).

Usage

makeBartelsConnFunction()

Value

smoof_single_objective_function
makeBBOBFunction

Generator for the noiseless function set of the real-parameter Black-Box Optimization Benchmarking (BBOB).

Description

Generator for the noiseless function set of the real-parameter Black-Box Optimization Benchmarking (BBOB).

Usage

makeBBOBFunction(dimensions, fid, iid)

Arguments

dimensions [integer(1)]
Problem dimension. Integer value between 2 and 40.

fid [integer(1)]
Function identifier. Integer value between 1 and 24.

iid [integer(1)]
Instance identifier. Integer value greater than or equal 1.

Value

smoof_single_objective_function

Note

It is possible to pass a matrix of parameters to the functions, where each column consists of one parameter setting.

References

See the BBOB website for a detailed description of the BBOB functions.

Examples

# get the first instance of the 2D Sphere function
fn = makeBBOBFunction(dimensions = 2L, fid = 1L, iid = 1L)
if (require(plot3D)) {
  plot3D(fn, contour = TRUE)
}

**makeBealeFunction**  

**Beale Function**

**Description**
Multimodal single-objective test function for optimization. It is based on the mathematic formula

\[ f(x) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2 \]

usually evaluated within the bounds \( x_i \in [-4.5, 4.5], \ i = 1, 2 \). The function has a flat but multimodal region around the single global optimum and large peaks in the edges of its definition space.

**Usage**

`makeBealeFunction()`

**Value**

`smoof_single_objective_function`

---

**makeBentCigarFunction**  

**Bent-Cigar Function**

**Description**
Scalable test function \( f \) with

\[ f(x) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2 \]

subject to \(-100 \leq x_i \leq 100\) for \( i = 1, \ldots, n \).

**Usage**

`makeBentCigarFunction(dimensions)`

**Arguments**

dimensions \[\text{[integer(1)]}\]  
Size of corresponding parameter space.

**Value**

`smoof_single_objective_function`

**References**
makeBiObjBBOBFunction  
*Generator for the function set of the real-parameter Bi-Objective Black-Box Optimization Benchmarking (BBOB).*

**Description**

Generator for the function set of the real-parameter Bi-Objective Black-Box Optimization Benchmarking (BBOB).

**Usage**

`makeBiObjBBOBFunction(dimensions, fid, iid)`

**Arguments**

- `dimensions` [integer(1)]
  Problem dimensions. Integer value between 2 and 40.
- `fid` [integer(1)]
  Function identifier. Integer value between 1 and 55.
- `iid` [integer(1)]
  Instance identifier. Integer value greater than or equal 1.

**Value**

`snowf_multi_objective_function`

**Note**

Concatenation of single-objective BBOB functions into a bi-objective problem.

**References**

See the COCO website for a detailed description of the bi-objective BBOB functions. An overview of which pair of single-objective BBOB functions creates which of the 55 bi-objective BBOB functions can be found here.

**Examples**

```r
# get the fifth instance of the concatenation of the
# 3D versions of sphere and Rosenbrock
fn = makeBiObjBBOBFunction(dimensions = 3L, fid = 4L, iid = 5L)
fn(c(3, -1, 0))
# compare to the output of its single-objective pendants
f1 = makeBBOBFunction(dimensions = 3L, fid = 1L, iid = 2L * 5L + 1L)
f2 = makeBBOBFunction(dimensions = 3L, fid = 8L, iid = 2L * 5L + 2L)
identical(fn(c(3, -1, 0)), c(f1(c(3, -1, 0)), f2(c(3, -1, 0))))
```
**makeBirdFunction**  

*Bird Function*

---------

**Description**

Multimodal single-objective test function. The implementation is based on the mathematical formulation

\[ f(x) = (x_1 - x_2)^2 + \exp((1 - \sin(x_1))^2) \cos(x_2) + \exp((1 - \cos(x_2))^2) \sin(x_1). \]

The function is restricted to two dimensions with \( x_i \in [-2\pi, 2\pi], i = 1, 2. \)

**Usage**

```
makeBirdFunction()
```

**Value**

`smaof_single_objective_function`

**References**


---------

**makeBiSphereFunction**  

*Bi-objective Sphere function*

---------

**Description**

Builds and returns the bi-objective Sphere test problem:

\[ f(x) = \left( \sum_{i=1}^{n} x_i^2, \sum_{i=1}^{n} (x_i - a)^2 \right) \]

where \( a \in \mathbb{R}^n \).

**Usage**

```
makeBiSphereFunction(dimensions, a = rep(0, dimensions))
```
makeBohachevskyN1Function

Arguments

dimensions [integer(1)]
Number of decision variables.
a [numeric(1)]
Shift parameter for the second objective. Default is (0,...,0).

Value
smoof_multi_objective_function

makeBK1Function  

BKI function generator

Description
...

Usage
makeBK1Function()

Value
smoof_multi_objective_function

References
...

makeBohachevskyN1Function

Bohachevsky function N. 1

Description
Highly multimodal single-objective test function. The mathematical formula is given by

\[ f(x) = \sum_{i=1}^{n-1} (x_i^2 + 2x_{i+1}^2 - 0.3 \cos(3\pi x_i) - 0.4 \cos(4\pi x_{i+1}) + 0.7) \]

with box-constraints \( x_i \in [-100, 100] \) for \( i = 1, \ldots, n \). The multimodality will be visible by “zooming in” in the plot.

Usage
makeBohachevskyN1Function(dimensions)
makeBoothFunction

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

smoof_single_objective_function

References


makeBoothFunction | Booth Function

Description

This function is based on the formula

\[ f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \]

subject to \( x_i \in [-10, 10], i = 1, 2 \).

Usage

makeBoothFunction()

Value

smoof_single_objective_function

makeBraninFunction | Branin RCOS function

Description

Popular 2-dimensional single-objective test function based on the formula:

\[ f(x) = a (x_2 - b x_1^2 + c x_1 - d)^2 + e (1 - f) \cos(x_1) + e, \]

where \( a = 1, b = \frac{5.1}{4\pi^2}, c = \frac{5}{\pi}, d = 6, e = 10 \) and \( f = \frac{1}{8\pi} \). The box constraints are given by \( x_1 \in [-5, 10] \) and \( x_2 \in [0, 15] \). The function has three global minima.

Usage

makeBraninFunction()
makeBrentFunction

**Value**

smoof_single_objective_function

**References**


**Examples**

```r
library(ggplot2)
fn = makeBraninFunction()
print(fn)
print(autoplot(fn, show.optimum = TRUE))
```

---

### Description

Single-objective two-dimensional test function. The formula is given as

\[
    f(x) = (x_1 + 10)^2 + (x_2 + 10)^2 + exp(-x_1^2 - x_2^2)
\]

subject to the constraints \(x_i \in [-10, 10], i = 1, 2\).

**Usage**

makeBrentFunction()

**Value**

smoof_single_objective_function

**References**

makeBrownFunction

Description
This function belongs to the unimodal single-objective test functions. The function is formulated as

\[ f(x) = \sum_{i=1}^{n} (x_i^2)(x_{i+1}+1) + (x_{i+1})(x_i+1) \]

subject to \( x_i \in [-1, 4] \) for \( i = 1, \ldots, n \).

Usage
makeBrownFunction(dimensions)

Arguments
dimensions [integer(1)]
Size of corresponding parameter space.

Value
smoof_single_objective_function

References

makeBukinN2Function

Bukin function N. 2

Description
Multimodal, non-scalable, continuous optimization test function given by:

\[ f(x) = 100(x_2 - 0.01 * x_1^2 + 1) + 0.01(x_1 + 10)^2 \]

subject to \( x_1 \in [-15, -5] \) and \( x_2 \in [-3, 3] \).

Usage
makeBukinN2Function()
**makeBukinN4Function**

**Value**

smoof_single_objective_function

**References**


**See Also**

makeBukinN2Function, makeBukinN6Function

---

**makeBukinN4Function**  
*Bukin function N. 4*

**Description**

Second continuous Bukin function test function. The formula is given by

\[
 f(x) = 100x_2^2 + 0.01 \times ||x_1 + 10||
\]

and the box constraints \(x_1 \in [-15, 5], x_2 \in [-3, 3]\).

**Usage**

makeBukinN4Function()

**Value**

smoof_single_objective_function

**References**


**See Also**

makeBukinN2Function, makeBukinN6Function
makeBukinN6Function  

Bukin function N. 6

Description
Beside Bukin N. 2 and N. 4 this is the last “Bukin family” function. It is given by the formula

\[ f(x) = 100 \sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10| \]

and the box constraints \( \mathbf{x}_1 \in [-15, 5], \mathbf{x}_2 \in [-3, 3] \).

Usage
makeBukinN6Function()

Value
smoof_single_objective_function

References

See Also
makeBukinN2Function, makeBukinN4Function

makeCarromTableFunction  

Carrom Table Function

Description
This function is defined as follows:

\[ f(x) = -\frac{1}{30} \left( \cos(x_1) \exp(|1 - ((x_1^2 + x_2^2)^{0.5}/\pi)^2|) \right). \]

The box-constraints are given by \( x_i \in [-10, 10], i = 1, 2 \).

Usage
makeCarromTableFunction()

Value
smoof_single_objective_function
References


--

makeChinchinadzeFunction

Chinchinadze Function

Description

Continuous single-objective test function \( f \) with

\[
  f(x) = x_1^2 - 12x_1 + 11 + 10 \cos(0.5\pi x_1) + 8 \sin(2.5\pi x_1) - (0.25)^{0.5} \exp(-0.5(x_2 - 0.5)^2)
\]

with \(-30 \leq x_i \leq 30, i = 1, 2\).

Usage

makeChinchinadzeFunction()

Value

smoof_single_objective_function

--

makeChungReynoldsFunction

Chung Reynolds Function

Description

The definition is given by

\[
  f(x) = \left( \sum_{i=1}^{n} x_i^2 \right)^2
\]

with box-constraings \( x_i \in [-100, 100], i = 1, \ldots, n \).

Usage

makeChungReynoldsFunction(dimensions)

Arguments

dimensions [integer(1)]
    Size of corresponding parameter space.
**makeComplexFunction**

**Value**

smoof_single_objective_function

**References**


---

**Description**

Two-dimensional test function based on the formula

\[ f(\mathbf{x}) = (x_1^3 - 3x_1x_2^2 - 1)^2 + (3x_2x_1^2 - x_3^2)^2 \]

with \(x_1, x_2 \in [-2, 2]\).

**Usage**

makeComplexFunction()

**Value**

smoof_single_objective_function

**References**


---

**makeCosineMixtureFunction**

**Cosine Mixture Function**

**Description**

Single-objective test function based on the formula

\[ f(\mathbf{x}) = -0.1 \sum_{i=1}^{n} \cos(5\pi x_i) - \sum_{i=1}^{n} x_i^2 \]

subject to \(x_i \in [-1, 1]\) for \(i = 1, \ldots, n\).
Usage

makeCrossInTrayFunction(dimensions)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

smoof_single_objective_function

References


makeCrossInTrayFunction

Cross-In-Tray Function

Description

Non-scalable, two-dimensional test function for numerical optimization with

\[
f(x) = -0.0001 \left( |\sin(x_1 x_2 \exp(|100 - [(x_1^2 + x_2^2)^{0.5}/\pi]|) + 1)^{0.1}
\right.
\]

subject to \( x_i \in [-15, 15] \) for \( i = 1, 2 \).

Usage

makeCrossInTrayFunction()

Value

smoof_single_objective_function

References

**makeCubeFunction**  

---

**Cube Function**

**Description**

The Cube Function is defined as follows:

\[ f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \]

The box-constraints are given by \( x_i \in [-10, 10], i = 1, 2. \)

**Usage**

makeCubeFunction()

**Value**

smoof_single_objective_function

**References**

A. Lavi, T. P. Vogel (eds), Recent Advances in Optimization Techniques, John Wiley & Sons, 1966.

---

**makeDeckkersAartsFunction**  

---

**Deckkers-Aarts Function**

**Description**

This continuous single-objective test function is defined by the formula

\[ f(x) = 10^5x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4 \]

with the bounding box \(-20 \leq x_i \leq 20\) for \( i = 1, 2. \).

**Usage**

makeDeckkersAartsFunction()

**Value**

smoof_single_objective_function

**References**

**makeDeflectedCorrugatedSpringFunction**

*Deflected Corrugated Spring function*

**Description**

Scalable single-objective test function based on the formula

\[ f(x) = 0.1 \sum_{i=1}^{n} (x_i - \alpha)^2 - \cos \left( K \sqrt{\sum_{i=1}^{n} (x_i - \alpha)^2} \right) \]

with \( x_i \in [0, 2\alpha], i = 1, \ldots, n \) and \( \alpha = K = 5 \) by default.

**Usage**

`makeDeflectedCorrugatedSpringFunction(dimensions, K = 5, alpha = 5)`

**Arguments**

- `dimensions` [integer(1)]
  Size of corresponding parameter space.
- `K` [numeric(1)]
  Parameter. Default is 5.
- `alpha` [numeric(1)]
  Parameter. Default is 5.

**Value**

`smoof_single_objective_function`

**References**


**makeDentFunction**

*Dent Function*

**Description**

Builds and returns the bi-objective Dent test problem, which is defined as follows:

\[
    f(x) = (f_1(x_1), f_2(x))
\]

with

\[
    f_1(x_1) = 0.5 \left( \sqrt{1 + (x_1 + x_2)^2} + \sqrt{1 + (x_1 - x_2)^2} + x_1 - x_2 \right) + d
\]

and

\[
    f_1(x_1) = 0.5 \left( \sqrt{1 + (x_1 + x_2)^2} + \sqrt{1 + (x_1 - x_2)^2} - x_1 + x_2 \right) + d
\]

where \( d = \lambda * \exp(-(x_1 - x_2)^2) \) and \( x_i \in [-1.5, 1.5], i = 1, 2 \).
**Description**

Dixon and Price defined the function

\[
  f(x) = (x_1 - 1)^2 + \sum_{i=1}^{n} i(2x_i^2 - x_{i-1})
\]

subject to \( x_i \in [-10, 10] \) for \( i = 1, \ldots, n \).

**Usage**

\[
  \text{makeDixonPriceFunction}(\text{dimensions})
\]

**Arguments**

- **dimensions**
  - \([\text{integer}(1)]\)
  - Size of corresponding parameter space.

**Value**

\[
  \text{smoof_single_objective_function}
\]

**References**

**makeDoubleSumFunction**  
*Double-Sum Function*

**Description**

Also known as the rotated hyper-ellipsoid function. The formula is given by

\[ f(x) = \sum_{i=1}^{n} \left( \sum_{j=1}^{i} x_j \right)^2 \]

with \( x_i \in [-65.536, 65.536], i = 1, \ldots, n. \)

**Usage**

```makeDoubleSumFunction(dimensions)```

**Arguments**

- `dimensions` [integer(1)]  
  Size of corresponding parameter space.

**Value**

`smoof_single_objective_function`

**References**


---

**makeDTLZ1Function**  
*DTLZ1 Function (family)*

**Description**

Builds and returns the multi-objective DTLZ1 test problem.

The DTLZ1 test problem is defined as follows:

Minimize \( f_1(x) = \frac{1}{2} x_1 x_2 \cdots x_{M-1}(1 + g(x_M)), \)

Minimize \( f_2(x) = \frac{1}{2} x_1 x_2 \cdots (1 - x_{M-1})(1 + g(x_M)), \)

\vdots\]

Minimize \( f_{M-1}(x) = \frac{1}{2} x_1 (1 - x_2)(1 + g(x_M)), \)

Minimize \( f_M(x) = \frac{1}{2}(1 - x_1)(1 + g(x_M)), \)

with \( 0 \leq x_i \leq 1, \) for \( i = 1, 2, \ldots, n, \)

where \( g(x_M) = 100 \left( |x_M| + \sum_{x_i \in x_M} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)) \right) \)
**Usage**

```r
makeDTLZ2Function(dimensions, n.objectives)
```

**Arguments**

- `dimensions` [integer(1)]: Number of decision variables.
- `n.objectives` [integer(1)]: Number of objectives.

**Value**

`smaof_multi_objective_function`

**References**


---

**Description**

Builds and returns the multi-objective DTLZ2 test problem.

The DTLZ2 test problem is defined as follows:

- Minimize $f_1(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \cos(x_2 \pi/2) \cdots \cos(x_{M-2} \pi/2) \cos(x_{M-1} \pi/2)$,
- Minimize $f_2(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \cos(x_2 \pi/2) \cdots \cos(x_{M-2} \pi/2) \sin(x_{M-1} \pi/2)$,
- Minimize $f_3(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \cos(x_2 \pi/2) \cdots \sin(x_{M-2} \pi/2)$,

... 

- Minimize $f_{M-1}(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \sin(x_2 \pi/2)$,
- Minimize $f_M(x) = (1 + g(x_M)) \sin(x_1 \pi/2)$,

where $0 \leq x_i \leq 1$, for $i = 1, 2, \ldots, n$.

with $g(x_M) = \sum_{x_i \in x_M} (x_i - 0.5)^2$

**Usage**

```r
makeDTLZ2Function(dimensions, n.objectives)
```
**Arguments**

- **dimensions** [integer(1)]
  Number of decision variables.
- **n.objectives** [integer(1)]
  Number of objectives.

**Value**

`smoof_multi_objective_function`

**References**


---

**Description**

Builds and returns the multi-objective DTLZ3 test problem. The formula is very similar to the formula of DTLZ2, but it uses the $g$ function of DTLZ1, which introduces a lot of local Pareto-optimal fronts. Thus, this problems is well suited to check the ability of an optimizer to converge to the global Pareto-optimal front.

The DTLZ3 test problem is defined as follows:

\[
\text{Minimize } f_1(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \cos(x_2 \pi/2) \cdots \cos(x_{M-2} \pi/2) \cos(x_{M-1} \pi/2),
\]

\[
\text{Minimize } f_2(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \cos(x_2 \pi/2) \cdots \cos(x_{M-2} \pi/2) \sin(x_{M-1} \pi/2),
\]

\[
\text{Minimize } f_3(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \cos(x_2 \pi/2) \cdots \sin(x_{M-2} \pi/2),
\]

\[\vdots\]

\[
\text{Minimize } f_{M-1}(x) = (1 + g(x_M)) \cos(x_1 \pi/2) \sin(x_2 \pi/2),
\]

\[
\text{Minimize } f_M(x) = (1 + g(x_M)) \sin(x_1 \pi/2),
\]

with $0 \leq x_i \leq 1$, for $i = 1, 2, \ldots, n$,

where $g(x_M) = 100 \left[ |x_M| + \sum_{i \in x_M} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)) \right]$

**Usage**

`makeDTLZ3Function(dimensions, n.objectives)`
makeDTLZ4Function

Arguments

dimensions [integer(1)]
Number of decision variables.
n.objectives [integer(1)]
Number of objectives.

Value

smoof_multi_objective_function

References


makeDTLZ4Function

DTLZ4 Function (family)

Description

Builds and returns the multi-objective DTLZ4 test problem. It is a slight modification of the DTLZ2 problems by introducing the parameter $\alpha$. The parameter is used to map $x_i \rightarrow x_i^\alpha$.

The DTLZ4 test problem is defined as follows:

Minimize $f_1(x) = (1 + g(x_M)) \cos(x_1^\alpha \pi/2) \cos(x_2^\alpha \pi/2) \cdots \cos(x_{M-2}^\alpha \pi/2) \cos(x_{M-1}^\alpha \pi/2)$,
Minimize $f_2(x) = (1 + g(x_M)) \cos(x_1^\alpha \pi/2) \cos(x_2^\alpha \pi/2) \cdots \cos(x_{M-2}^\alpha \pi/2) \sin(x_{M-1}^\alpha \pi/2)$,
Minimize $f_3(x) = (1 + g(x_M)) \cos(x_1^\alpha \pi/2) \cos(x_2^\alpha \pi/2) \cdots \sin(x_{M-2}^\alpha \pi/2)$,

\[ \vdots \]

Minimize $f_{M-1}(x) = (1 + g(x_M)) \cos(x_1^\alpha \pi/2) \sin(x_2^\alpha \pi/2)$,
Minimize $f_M(x) = (1 + g(x_M)) \sin(x_1^\alpha \pi/2)$,

with $0 \leq x_i \leq 1$, for $i = 1, 2, \ldots, n$,
where $g(x_M) = \sum_{x_i \in x_M} (x_i - 0.5)^2$

Usage

makeDTLZ4Function(dimensions, n.objectives, alpha = 100)
makeDTLZ5Function

Arguments

- **dimensions** [integer(1)]
  Number of decision variables.

- **n.objectives** [integer(1)]
  Number of objectives.

- **alpha** [numeric(1)]
  Optional parameter. Default is 100, which is recommended by Deb et al.

Value

smoof_multi_objective_function

References


Description

Builds and returns the multi-objective DTLZ5 test problem. This problem can be characterized by a disconnected Pareto-optimal front in the search space. This introduces a new challenge to evolutionary multi-objective optimizers, i.e., to maintain different subpopulations within the search space to cover the entire Pareto-optimal front.

The DTLZ5 test problem is defined as follows:

Minimize \( f_1(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) \cdots \cos(\theta_{M-2} \pi/2) \cos(\theta_{M-1} \pi/2), \)

Minimize \( f_2(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) \cdots \cos(\theta_{M-2} \pi/2) \sin(\theta_{M-1} \pi/2), \)

Minimize \( f_3(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) \cdots \sin(\theta_{M-2} \pi/2), \)

\vdots

Minimize \( f_{M-1}(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \sin(\theta_2 \pi/2), \)

Minimize \( f_M((1 + g(x_M)) \sin(\theta_1 \pi/2), \)

with \( 0 \leq x_i \leq 1, \) for \( i = 1, 2, \ldots, n, \)

where \( \theta_i = \frac{\pi}{4(1+g(x_M))} \left(1 + 2g(x_M)x_i\right), \) for \( i = 2, 3, \ldots, (M-1) \)

and \( g(x_M) = \sum_{x_i \in x_M} (x_i - 0.5)^2 \)

Usage

makeDTLZ5Function(dimensions, n.objectives)
makeDTLZ6Function

Arguments

- dimensions [integer(1)]
  Number of decision variables.
- n.objectives [integer(1)]
  Number of objectives.

Value

smoof_multi_objective_function

References


Description

Builds and returns the multi-objective DTLZ6 test problem. This problem can be characterized by a disconnected Pareto-optimal front in the search space. This introduces a new challenge to evolutionary multi-objective optimizers, i.e., to maintain different subpopulations within the search space to cover the entire Pareto-optimal front.

The DTLZ6 test problem is defined as follows:

Minimize $f_1(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) \cdots \cos(\theta_{M-2} \pi/2) \cos(\theta_{M-1} \pi/2)$,

Minimize $f_2(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) \cdots \cos(\theta_{M-2} \pi/2) \sin(\theta_{M-1} \pi/2)$,

Minimize $f_3(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) \cdots \sin(\theta_{M-2} \pi/2)$,

... :

Minimize $f_{M-1}(x) = (1 + g(x_M)) \cos(\theta_1 \pi/2) \sin(\theta_2 \pi/2)$,

Minimize $f_M((1 + g(x_M)) \sin(\theta_1 \pi/2)$,

with $0 \leq x_i \leq 1$, for $i = 1, 2, \ldots, n$,

where $\theta_i = \frac{\pi}{2(1 + g(x_M))} (1 + 2g(x_M)x_i)$, for $i = 2, 3, \ldots, (M - 1)$

and $g(x_M) = \sum_{x_i \in x_M} x_i^{0.1}$

Usage

makeDTLZ6Function(dimensions, n.objectives)
makeDTLZ7Function

Arguments

- **dimensions** [integer(1)]
  Number of decision variables.

- **n.objectives** [integer(1)]
  Number of objectives.

Value

**smoof_multi_objective_function**

References


makeDTLZ7Function  
**DTLZ7 Function (family)**

Description

Builds and returns the multi-objective DTLZ7 test problem. This problem can be characterized by a disconnected Pareto-optimal front in the search space. This introduces a new challenge to evolutionary multi-objective optimizers, i.e., to maintain different subpopulations within the search space to cover the entire Pareto-optimal front.

The DTLZ7 test problem is defined as follows:

Minimize \( f_1(x) = x_1 \),

Minimize \( f_2(x) = x_2 \),

\[ \vdots \]

Minimize \( f_{M-1}(x) = x_{M-1} \),

Minimize \( f_M(x) = (1 + g(x_M))h(f_1, f_2, \cdots, f_{M-1}, g) \),

with \( 0 \leq x_i \leq 1 \), for \( i = 1, 2, \ldots, n \),

where \( g(x_M) = 1 + \frac{9}{|x_M|} \sum_{x_i \in x_M} x_i \)

and \( h(f_1, f_2, \cdots, f_{M-1}, g) = M - \sum_{i=1}^{M-1} \left[ \frac{f_i}{1 + (3\pi f_i)} (1 + \sin(3\pi f_i)) \right] \)

Usage

`makeDTLZ7Function(dimensions, n.objectives)`
**Arguments**

- **dimensions**  \([\text{integer}(1)]\)
  
  Number of decision variables.

- **n.objectives**  \([\text{integer}(1)]\)
  
  Number of objectives.

**Value**

- **smoof_multi_objective_function**

**References**


---

**Description**

Unimodal function with its global optimum in the center of the search space. The attraction area of the global optimum is very small in relation to the search space:

\[
f(x) = -\cos(x_1) \cos(x_2) \exp \left(-\left((x_1 - \pi)^2 + (x_2 - \pi)^2\right)\right)
\]

with \(x_i \in [-100, 100], i = 1, 2\).

**Usage**

- **makeEasomFunction()**

**Value**

- **smoof_single_objective_function**

**References**

Description

Builds and returns the multi-objective ED1 test problem.

The ED1 test problem is defined as follows:

Minimize $f_j(x) = \frac{1}{r(x)+1} \cdot \tilde{p}(\Theta(x))$, for $j = 1, \ldots, m$,

with $x = (x_1, \ldots, x_n)^T$, where $0 \leq x_i \leq 1$, and $\Theta = (\theta_1, \ldots, \theta_{m-1})$, where $0 \leq \theta_j \leq \frac{\pi}{2}$, for $i = 1, \ldots, n$, and $j = 1, \ldots, m - 1$.

Moreover $r(X) = \sqrt{x_m^2 + \ldots + x_n^2}$,

$\tilde{p}_1(\Theta) = \cos(\theta_1)^{2/\gamma}$,

$\tilde{p}_j(\Theta) = (\sin(\theta_1) \cdot \ldots \cdot \sin(\theta_{j-1}) \cdot \cos(\theta_j))^{2/\gamma}$, for $2 \leq j \leq m - 1$,

and $\tilde{p}_m(\Theta) = (\sin(\theta_1) \cdot \ldots \cdot \sin(\theta_{m-1}))^{2/\gamma}$.

Usage

`makeED1Function(dimensions, n.objectives, gamma = 2, theta)`

Arguments

- `dimensions` [integer(1)]
  Number of decision variables.

- `n.objectives` [integer(1)]
  Number of objectives.

- `gamma` [numeric(1)]
  Optional parameter. Default is 2, which is recommended by Emmerich and Deutz.

- `theta` [numeric(dimensions)]
  Parameter vector, whose components have to be between 0 and 0.5*\pi. The default is theta = (pi/2) * x (with x being the point from the decision space) as recommended by Emmerich and Deutz.

Value

`smoof_multi_objective_function`

References

**makeED2Function**

**Description**

Builds and returns the multi-objective ED2 test problem.

The ED2 test problem is defined as follows:

Minimize \( f_j(x) = \frac{1}{F_{natmin}(x) + 1} \cdot \tilde{p}(\Theta(X)) \), for \( j = 1, \ldots, m \),

with \( x = (x_1, \ldots, x_n)^T \), where \( 0 \leq x_i \leq 1 \), and \( \Theta = (\theta_1, \ldots, \theta_{m-1}) \), where \( 0 \leq \theta_j \leq \frac{\pi}{2} \), for \( i = 1, \ldots, n \), and \( j = 1, \ldots, m - 1 \).

Moreover \( F_{natmin}(x) = b + (r(x) - a) + 0.5 + 0.5 \cdot (2\pi \cdot (r(x) - a) + \pi) \)

with \( a \approx 0.051373 \), \( b \approx 0.0253235 \), and \( r(X) = \sqrt{x_1^2 + \ldots + x_n^2} \), as well as

\( \tilde{p}_1(\Theta) = \cos(\theta_1)^{2/\gamma} \),

\( \tilde{p}_j(\Theta) = (\sin(\theta_1) \cdot \ldots \cdot \sin(\theta_{j-1}) \cdot \cos(\theta_j))^{2/\gamma} \), for \( 2 \leq j \leq m - 1 \),

and \( \tilde{p}_m(\Theta) = (\sin(\theta_1) \cdot \ldots \cdot \sin(\theta_{m-1}))^{2/\gamma} \).

**Usage**

```r
makeED2Function(dimensions, n.objectives, gamma = 2, theta)
```

**Arguments**

- `dimensions` [integer(1)]
  Number of decision variables.
- `n.objectives` [integer(1)]
  Number of objectives.
- `gamma` [numeric(1)]
  Optional parameter. Default is 2, which is recommended by Emmerich and Deutz.
- `theta` [numeric(dimensions)]
  Parameter vector, whose components have to be between 0 and \( 0.5\pi \). The default is \( \theta = (\pi/2) \times x \) (with \( x \) being the point from the decision space) as recommended by Emmerich and Deutz.

**Value**

`smoof_multi_objective_function`

**References**

makeEggCrateFunction  *Egg Crate Function*

**Description**

This single-objective function follows the definition

\[
f(x) = x_1^2 + x_2^2 + 25(\sin^2(x_1) + \sin^2(x_2))
\]

with \( x_i \in [-5, 5] \) for \( i = 1, 2 \).

**Usage**

makeEggCrateFunction()

**Value**

smoof_single_objective_function

---

makeEggholderFunction  *Egg Holder function*

**Description**

The Egg Holder function is a difficult to optimize function based on the definition

\[
f(x) = \sum_{i=1}^{n-1} \left[ -(x_{i+1} + 47) \sin \sqrt{|x_{i+1} + 0.5x_i + 47|} - x_i \sin (\sqrt{|x_i - (x_{i+1} - 47)|}) \right]
\]

subject to \(-512 \leq x_i \leq 512\) for \( i = 1, \ldots, n \).

**Usage**

makeEggholderFunction()

**Value**

smoof_single_objective_function
Description

This function is based on the formula

\[ f(x) = (x_1^2 + x_2^2 - 10)^2 + (x_1 + x_2^2 - 7)^2 + (x_1^2 + x_2^3 - 1)^2 \]

subject to \( x_i \in [-500, 500], i = 1, 2. \)

Usage

makeElAttarVidyasagarDuttaFunction()

Value

smoof_single_objective_function

References

R. A. El-Attar, M. Vidyasagar, S. R. K. Dutta, An Algorithm for II-norm Minimiza-

Description

Two-dimensional test function based on the formula

\[ f(x) = (x_1^4 + x_2^3 + 2x_1^2x_2^2 - 4x_1 + 3 \]

with \( x_1, x_2 \in [-2000, 2000]. \)

Usage

makeEngvallFunction()

Value

smoof_single_objective_function

References

**makeExponentialFunction**

*Exponential Function*

**Description**

This scalable test function is based on the definition

\[ f(x) = -\exp\left(-0.5 \sum_{i=1}^{n} x_i^2\right) \]

with the box-constraints \( x_i \in [-1, 1], i = 1, \ldots, n \).

**Usage**

`makeExponentialFunction(dimensions)`

**Arguments**

- `dimensions` [integer(1)]
  
  Size of corresponding parameter space.

**Value**

`smoof_single_objective_function`

**References**


---

**makeFreudensteinRothFunction**

*Freudenstein Roth Function*

**Description**

This test function is based on the formula

\[ f(x) = (x_1 - 13 + ((5 - x_2)x_2 - 2)x_2)^2 + (x_1 - 29 + ((x_2 + 1)x_2 - 14)x_2)^2 \]

subject to \( x_i \in [-10, 10], i = 1, 2 \).

**Usage**

`makeFreudensteinRothFunction()`
makeFunctionsByName

Value

smoof_single_objective_function

References


Description

This function is especially useful in combination with filterFunctionsByTags to generate a test set of functions with certain properties, e.g., multimodality.

Usage

makeFunctionsByName(fun.names, ...)

Arguments

fun.names [character]
Non empty character vector of generator function names.

... [any]
Further arguments passed to generator.

Value

smoof_function

See Also

filterFunctionsByTags

Examples

# generate a testset of multimodal 2D functions
### Not run:
test.set = makeFunctionsByName(filterFunctionsByTags("multimodal"), dimensions = 2L, m = 5L)

### End(Not run)
**Generalized Drop-Wave Function**

**Description**

Multimodal single-objective function following the formula:

\[ x = -\frac{1 + \cos\left(\sqrt{\sum_{i=1}^{n} x_i^2}\right)}{2 + 0.5 \sum_{i=1}^{n} x_i^2} \]

with \( x_i \in [-5.12, 5.12], i = 1, \ldots, n \).

**Usage**

```r
makeGeneralizedDropWaveFunction(dimensions = 2L)
```

**Arguments**

- `dimensions` [integer(1)]
  Size of corresponding parameter space.

**Value**

`smoof_single_objective_function`

---

**Giunta Function**

**Description**

Multimodal test function based on the definition

\[ f(x) = 0.6 + \sum_{i=1}^{n} \left[ \sin\left(\frac{16}{15} x_i - 1\right) + \sin^2\left(\frac{16}{15} x_i - 1\right) + \frac{1}{50} \sin\left(4\left(\frac{16}{15} x_i - 1\right)\right) \right] \]

with box-constraints \( x_i \in [-1, 1] \) for \( i = 1, \ldots, n \).

**Usage**

```r
makeGiuntaFunction()
```

**Value**

`smoof_single_objective_function`

**References**

**Description**

Two-dimensional test function for global optimization. The implementation follows the formula:

\[ f(x) = (1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot (30 + (2x_1 - 3x_2)^2 \cdot (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)) \]

with \( x_i \in [-2, 2], i = 1, 2 \).

**Usage**

`makeGoldsteinPriceFunction()`

**Value**

`smoof_single_objective_function`

**References**


---

**makeGOMOPFunction**  
**GOMOP function generator.**

**Description**

Construct a multi-objective function by putting together multiple single-objective smoof functions.

**Usage**

`makeGOMOPFunction(dimensions = 2L, funs = list())`

**Arguments**

- **dimensions**  
  [integer(1)]  
  Size of corresponding parameter space.

- **funs**  
  [list]  
  List of single-objective smoof functions.
makeGriewankFunction

Details
The decision space of the resulting function is restricted to $[0, 1]^d$. Each parameter $x$ is stretched for each objective function. I.e., if $f_1, \ldots, f_n$ are the single objective smoof functions with box constraints $[l_i, u_i], i = 1, \ldots, n$, then

$$f(x) = (f_1(l_1 + x * (u_1 - l_1)), \ldots, f_1(l_1 + x * (u_1 - l_1)))$$

for $x \in [0, 1]^d$ where the additions and multiplication are performed component-wise.

Value
smoof_multi_objective_function

makeGriewankFunction  Griewank Function

Description
Highly multimodal function with a lot of regularly distributed local minima. The corresponding formula is:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1$$

subject to $x_i \in [-100, 100], i = 1, \ldots, n$.

Usage
makeGriewankFunction(dimensions)

Arguments
dimensions  [integer(1)]
Size of corresponding parameter space.

Value
smoof_single_objective_function

References
makeHansenFunction  

**Hansen Function**

**Description**

Test function with multiple global optima based on the definition

\[ f(x) = \sum_{i=1}^{4} (i + 1) \cos(i x_1 + i - 1) \sum_{j=1}^{4} (j + 1) \cos((j + 2)x_2 + j + 1) \]

subject to \( x_i \in [-10, 10], i = 1, 2. \)

**Usage**

makeHansenFunction()

**Value**

smoof_single_objective_function

**References**


makeHartmannFunction  

**Hartmann Function**

**Description**

Unimodal single-objective test function with six local minima. The implementation is based on the mathematical formulation

\[ f(x) = -\sum_{i=1}^{4} \alpha_i \exp \left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2 \right) \]

, where

\[ \alpha = (1.0, 1.2, 3.0, 3.2)^T, A = \begin{pmatrix} 10 & 3 & 17 & 3.50 & 17 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, P = 10^{-4}. \]

The function is restricted to six dimensions with \( x_i \in [0, 1], i = 1, \ldots, 6. \) The function is not normalized in contrast to some benchmark applications in the literature.
Usage

makeHimmelblauFunction(dimensions)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

global_optimum = c(3, 2)

References


---

makeHimmelblauFunction

Himmelblau Function

Description

Two-dimensional test function based on the function definition

\[ f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \]

with box-constraing \( x_i \in [-5, 5], i = 1, 2. \)

Usage

makeHimmelblauFunction()

Value

smoof_single_objective_function

References

**makeHolderTableN1Function**

*Holder Table function N. 1*

**Description**

This multimodal function is defined as

\[ f(x) = -|\cos(x_1) \cos(x_2) \exp(1 - \sqrt{x_1 + x_2}/\pi)| \]

with box-constraints \( x_i \in [-10, 10], i = 1, 2 \).

**Usage**

`makeHolderTableN1Function()`

**Value**

`smoof_single_objective_function`

**References**


**See Also**

`makeHolderTableN2Function`

---

**makeHolderTableN2Function**

*Holder Table function N. 2*

**Description**

This multimodal function is defined as

\[ f(x) = -|\sin(x_1) \cos(x_2) \exp(1 - \sqrt{x_1 + x_2}/\pi)| \]

with box-constraints \( x_i \in [-10, 10], i = 1, 2 \).

**Usage**

`makeHolderTableN2Function()`
makeHosakiFunction

Value

smoof_single_objective_function

References


See Also

makeHolderTableN1Function

---

makeHosakiFunction  Hosaki Function

Description

Two-dimensional test function $f$ with

$$f(x) = (1 - 8x_1 + 7x_1^2 - 7/3x_1^3 + 1/4x_1^4)x_2^2e^{-x_2}$$

subject to $0 \leq x_1 \leq 5$ and $0 \leq x_2 \leq 6$.

Usage

makeHosakiFunction()

Value

smoof_single_objective_function

References

makeHyperEllipsoidFunction

Hyper-Ellipsoid function

Description
Unimodal, convex test function similar to the Sphere function (see makeSphereFunction). Calculated via the formula:

\[ f(x) = \sum_{i=1}^{n} i \cdot x_i. \]

Usage
makeHyperEllipsoidFunction(dimensions)

Arguments
dimensions [integer(1)]
Size of corresponding parameter space.

Value
smoof_single_objective_function

makeInvertedVincentFunction

Inverted Vincent Function

Description
Single-objective test function based on the formula

\[ f(x) = \frac{1}{n} \sum_{i=1}^{n} \sin(10 \log(x_i)) \]

subject to \( x_i \in [0.25, 10] \) for \( i = 1, \ldots, n \).

Usage
makeInvertedVincentFunction(dimensions)

Arguments
dimensions [integer(1)]
Size of corresponding parameter space.
makeJennrichSampsonFunction

Value

smoof_single_objective_function

References


---

Description

Two-dimensional test function based on the formula

\[ f(x) = \sum_{i=1}^{10} \left[ 2 + 2i - (e^{ix_1} + e^{ix_2}) \right]^2 \]

with \( x_1, x_2 \in [-1, 1] \).

Usage

makeJennrichSampsonFunction()

Value

smoof_single_objective_function

References

**makeJudgeFunction**  
*Judge function.*

**Description**

Two-dimensional test function based on the formula

\[
    f(x) = \sum_{i=1}^{20} \left( (x_1 + B_i x_2 + C_i x_2^2) - A_i \right)^2
\]

with \(x_1, x_2 \in [-10, 10]\). For details on \(A, B, C\) see the referenced website.

**Usage**

`makeJudgeFunction()`

**Value**

`smoof_single_objective_function`

**References**


---

**makeKeaneFunction**  
*Keane Function*

**Description**

Two-dimensional test function based on the definition

\[
    f(x) = \frac{\sin^2(x_1 - x_2) \sin^2(x_1 + x_2)}{\sqrt{x_1^2 + x_2^2}}.
\]

The domain of definition is bounded by the box constraints \(x_i \in [0, 10], i = 1, 2\).

**Usage**

`makeKeaneFunction()`

**Value**

`smoof_single_objective_function`
**makeKearfottFunction**  
*Kearfott function.*

### Description

Two-dimensional test function based on the formula

\[
f(x) = (x_1^2 + x_2^2 - 2)^2 + (x_1^2 - x_2^2 - 1)^2
\]

with \(x_1, x_2 \in [-3, 4]\).

### Usage

```r
makeKearfottFunction()
```

### Value

`smaof_single_objective_function`

### References


---

**makeKursaweFunction**  
*Kursawe Function*

### Description

Builds and returns the bi-objective Kursawe test problem.  
The Kursawe test problem is defined as follows:

Minimize \( f_1(x) = \sum_{i=1}^{n-1} (-10 \cdot \exp(-0.2 \cdot \sqrt{x_i^2 + x_{i+1}^2})) \),

Minimize \( f_2(x) = \sum_{i=1}^{n} (|x_i|^{0.8} + 5 \cdot \sin^3(x_i)) \),

with \(-5 \leq x_i \leq 5\), for \(i = 1, 2, \ldots, n\).

### Usage

```r
makeKursaweFunction(dimensions)
```

### Arguments

- **dimensions**  
  [integer(1)]
  Number of decision variables.
**Leon Function**

### Description

The function is based on the definition

\[
f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2
\]

Box-constraints: \(x_i \in [-1.2, 1.2]\) for \(i = 1, 2\).

### Usage

```r
makeLeonFunction()
```

**Matyas Function**

### Description

Two-dimensional, unimodal test function

\[
f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2
\]

subject to \(x_i \in [-10, 10], i = 1, 2\).

### Usage

```r
makeMatyasFunction()
```
Value
smoof_single_objective_function

References
A.-R. Hedar, Global Optimization Test Problems.

makeMcCormickFunction  McCormick Function

Description
Two-dimensional, multimodal test function. The definition is given by
\[ f(x) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1 \]
subject to \( x_1 \in [-1.5, 4], x_2 \in [-3, 3] \).

Usage
makeMcCormickFunction()

Value
smoof_single_objective_function

References

makeMichalewiczFunction  Michalewicz Function

Description
Highly multimodal single-objective test function with \( n! \) local minima with the formula:
\[ f(x) = -\sum_{i=1}^{n} \sin(x_i) \cdot \left( \sin \left( \frac{i \cdot x_i}{\pi} \right) \right)^{2m} \]
The recommended value \( m = 10 \), which is used as a default in the implementation.

Usage
makeMichalewiczFunction(dimensions, m = 10)
Arguments

- **dimensions**: [integer(1)]
  - Size of corresponding parameter space.

- **m**: [integer(1)]
  - “Steepness” parameter.

Value

smoof_single_objective_function

Note

The location of the global optimum is varying based on both the dimension and m parameter and is thus not provided in the implementation.

References


Description

A modified version of the Rastrigin function following the formula:

$$f(x) = \sum_{i=1}^{n} 10(1 + \cos(2\pi k_i x_i)) + 2k_i x_i^2.$$  

The box-constraints are given by \(x_i \in [0, 1]\) for \(i = 1, \ldots, n\) and \(k\) is a numerical vector. Deb et al. (see references) use, e.g., \(k = (2, 2, 3, 4)\) for \(n = 4\). See the reference for details.

Usage

makeModifiedRastriginFunction(dimensions, k = rep(1, dimensions))

Arguments

- **dimensions**: [integer(1)]
  - Size of corresponding parameter space.

- **k**: [numeric]
  - Vector of numerical values of length dimensions. Default is rep(1, dimensions)

Value

smoof_single_objective_function
References


makeMOP1Function  
MOP1 function generator.

Description

MOP1 function from Van Valedhuizen’s test suite.

Usage

makeMOP1Function()

Value

smoof_multi_objective_function

References


makeMOP2Function  
MOP2 function generator.

Description

MOP2 function from Van Valedhuizen’s test suite due to Fonseca and Fleming.

Usage

makeMOP2Function(dimensions = 2L)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

smoof_multi_objective_function
References


makeMOP3Function

Description

MOP3 function from Van Valedhuizen's test suite.

Usage

makeMOP3Function(dimensions = 2L)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

smoof_multi_objective_function

References


makeMOP4Function

Description

MOP4 function from Van Valedhuizen's test suite based on Kursawe.

Usage

makeMOP4Function()

Value

smoof_multi_objective_function
References


makeMOP5Function

Description

MOP5 function from Van Valedhuizen’s test suite.

Usage

makeMOP5Function()

Value

smoof_multi_objective_function

Note

Original box constraints where $[-3, 3]$.

References


makeMOP6Function

Description

MOP6 function from Van Valedhuizen’s test suite.

Usage

makeMOP6Function()

Value

smoof_multi_objective_function
**makeMOP7Function**

*MOP7 function generator.*

**Description**

MOP7 function from Van Valedhuizen’s test suite.

**Usage**

```r
makeMOP7Function()
```

**Value**

`smaof_multi_objective_function`

**References**


---

**makeMPM2Function**

*Generator for function with multiple peaks following the multiple peaks model 2.*

**Description**

Generator for function with multiple peaks following the multiple peaks model 2.

**Usage**

```r
makeMPM2Function(n.peaks, dimensions, topology, seed, rotated = TRUE, peak.shape = "ellipse")
```

**Arguments**

- `n.peaks` [integer(1)] Desired number of peaks, i.e., number of (local) optima.
- `dimensions` [integer(1)] Size of corresponding parameter space.
- `topology` [character(1)] Type of topology. Possible values are “random” and “funnel”.
- `seed` [integer(1)] Seed for the random numbers generator.
**makeMultiObjectiveFunction**

Generator for multi-objective target functions.

**Description**

Generator for multi-objective target functions.

```r
rotated [logical(1)]
Should the peak shapes be rotated? This parameter is only relevant in case of elliptically shaped peaks.

peak.shape [character(1)]
Shape of peak(s). Possible values are “ellipse” and “sphere”.
```

**Value**

`smoof_single_objective_function`

**Author(s)**

R interface by Jakob Bossek. Original python code provided by the Simon Wessing.

**References**

See the technical report of multiple peaks model 2 for an in-depth description of the underlying algorithm.

**Examples**

```r
## Not run:
fn = makeMPM2Function(n.peaks = 10L, dimensions = 2L,
  topology = “funnel”, seed = 123, rotated = TRUE, peak.shape = “ellipse”)
if (require(plot3D)) {
  plot3D(fn)
}
## End(Not run)

## Not run:
fn = makeMPM2Function(n.peaks = 5L, dimensions = 2L,
  topology = “random”, seed = 134, rotated = FALSE)
plot(fn, render.levels = TRUE)
## End(Not run)
```
makeMultiObjectiveFunction

Usage

makeMultiObjectiveFunction(name = NULL, id = NULL,
                          description = NULL, fn, has.simple.signature = TRUE, par.set,
                          n.objectives = NULL, noisy = FALSE, fn.mean = NULL,
                          minimize = NULL, vectorized = FALSE, constraint.fn = NULL,
                          ref.point = NULL)

Arguments

name [character(1)]
  Function name. Used for the title of plots for example.
id [character(1) | NULL]
  Optional short function identifier. If provided, this should be a short name without
  whitespaces and now special characters beside the underscore. Default is
  NULL, which means no ID at all.
description [character(1) | NULL]
  Optional function description.
fn [function]
  Objective function.
has.simple.signature [logical(1)]
  Set this to TRUE if the objective function expects a vector as input and FALSE if
  it expects a named list of values. The latter is needed if the function depends on
  mixed parameters. Default is TRUE.
par.set [ParamSet]
  Parameter set describing different aspects of the objective function parameters,
  i.e., names, lower and/or upper bounds, types and so on. See makeParamSet
  for further information.
n.objectives [integer(1)]
  Number of objectives of the multi-objective function.
noisy [logical(1)]
  Is the function noisy? Defaults to FALSE.
fn.mean [function]
  Optional true mean function in case of a noisy objective function. This functions
  should have the same mean as fn.
minimize [logical]
  Logical vector of length n.objectives indicating if the corresponding objectives
  shall be minimized or maximized. Default is the vector with all components set to TRUE.
vectorized [logical(1)]
  Can the objective function handle "vector" input, i.e., does it accept matrix of
  parameters? Default is FALSE.
constraint.fn [function | NULL]
  Function which returns a logical vector indicating whether certain conditions are
  met or not. Default is NULL, which means, that there are no constraints beside
  possible box constraints defined via the par.set argument.
**makePeriodicFunction**

**Description**

Single-objective two-dimensional test function. The formula is given as

\[
f(x) = 1 + \sin^2(x_1) + \sin^2(x_2) - 0.1e^{-(x_1^2 + x_2^2)}
\]

subject to the constraints \( x_i \in [-10, 10], i = 1, 2 \).

**Usage**

makePeriodicFunction()

**Value**

smoof_single_objective_function

**References**

**makePowellSumFunction**  

**Powell-Sum Function**

**Description**

The formula that underlies the implementation is given by

\[ f(x) = \sum_{i=1}^{n} |x_i|^{i+1} \]

with \( x_i \in [-1, 1], i = 1, \ldots, n \).

**Usage**

`makePowellSumFunction(dimensions)`

**Arguments**

- `dimensions`  
  [integer(1)]  
  Size of corresponding parameter space.

**Value**

`smoof_single_objective_function`

**References**


---

**makePriceN1Function**  

**Price Function N. 1**

**Description**

Second function by Price. The implementation is based on the definition

\[ f(x) = (|x_1| - 5)^2 + (|x_2| - 5)^2 \]

subject to \( x_i \in [-500, 500], i = 1, 2 \).

**Usage**

`makePriceN1Function()`
makePriceN2Function

Value

smoof_single_objective_function

References


See Also

makePriceN1Function, makePriceN4Function

---

makePriceN2Function  Price Function N. 2

Description

Second function by Price. The implementation is based on the definition

\[ f(x) = 1 + \sin^2(x_1) + \sin^2(x_2) - 0.1 \exp(-x^2 - x_2^2) \]

subject to \( x_i \in [-10, 10], i = 1, 2 \).

Usage

makePriceN2Function()

Value

smoof_single_objective_function

References


See Also

makePriceN1Function, makePriceN4Function
makePriceN4Function  Price Function N. 4

Description

Fourth function by Price. The implementation is based on the definition

\[ f(x) = (2x_1^3x_2 - x_2^3)^2 + (6x_1 - x_2^2 + x_2)^2 \]

subject to \( x_i \in [-500, 500] \).

Usage

makePriceN4Function()

Value

smoof_single_objective_function

References


See Also

makePriceN1Function, makePriceN2Function

makeRastriginFunction  Rastrigin Function

Description

One of the most popular single-objective test functions consists of many local optima and is thus highly multimodal with a global structure. The implementation follows the formula

\[ f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10 \cos(2\pi x_i)) \cdot \]

The box-constraints are given by \( x_i \in [-5.12, 5.12] \) for \( i = 1, \ldots, n \).

Usage

makeRastriginFunction(dimensions)
makeRosenbrockFunction

Arguments

- dimensions [integer(1)]
  - Size of corresponding parameter space.

Value

smoof_single_objective_function

References


Description

Also known as the “De Jong’s function 2” or the “(Rosenbrock) banana/valley function” due to its shape. The global optimum is located within a large flat valley and thus it is hard for optimization algorithms to find it. The following formula underlies the implementation:

\[
   f(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2.
\]

The domain is given by the constraints \(x_i \in [-30, 30], i = 1, \ldots, n\).

Usage

makeRosenbrockFunction(dimensions)

Arguments

- dimensions [integer(1)]
  - Size of corresponding parameter space.

Value

smoof_single_objective_function

References

**makeSchafferN2Function**

*Modified Schaffer Function N. 2*

**Description**

Second function by Schaffer. The definition is given by the formula

\[ f(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \]

subject to \(x_i \in [-100, 100], i = 1, 2\).

**Usage**

```
makeSchafferN2Function()
```

**Value**

`smoof_single_objective_function`

**References**

S. K. Mishra, Some New Test Functions For Global Optimization And Performance of Repulsive Particle Swarm Method.

---

**makeSchafferN4Function**

*Schaffer Function N. 4*

**Description**

Second function by Schaffer. The definition is given by the formula

\[ f(x) = 0.5 + \frac{\cos^2(sin(|x_1^2 - x_2^2|)) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \]

subject to \(x_i \in [-100, 100], i = 1, 2\).

**Usage**

```
makeSchafferN4Function()
```

**Value**

`smoof_single_objective_function`
References

S. K. Mishra, Some New Test Functions For Global Optimization And Performance of Repulsive Particle Swarm Method.

makeSchwefelFunction  Schwefel function

Description
Highly multimodal test function. The crucial thing about this function is, that the global optimum is far away from the next best local optimum. The function is computed via:

\[ f(x) = \sum_{i=1}^{n} -x_i \sin \left( \sqrt{|x_i|} \right) \]

with \( x_i \in [-500, 500], i = 1, \ldots, n. \)

Usage
makeSchwefelFunction(dimensions)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value
smoof_single_objective_function

References

makeShekelFunction  Shekel functions

Description
Single-objective test function based on the formula

\[ f(x) = -\sum_{i=1}^{m} \left( \sum_{j=1}^{4} (x_j - C_{ji})^2 + \beta_i \right)^{-1} \]

Here, \( m \in \{5, 7, 10\} \) defines the number of local optima, \( C \) is a 4x10 matrix and \( \beta = \frac{1}{10}(1, 1, 2, 2, 4, 4, 6, 3, 7, 5, 5) \) is a vector. See https://www.sfu.ca/~ssurjano/shekel.html for a definition of \( C \).
makeShubertFunction

Usage

makeShubertFunction(m)

Arguments

m [numeric(1)]
Integer parameter (defines the number of local optima). Possible values are 5, 7 or 10.

Value

smoof_single_objective_function

Description

The definition of this two-dimensional function is given by

\[ f(x) = \prod_{i=1}^{2} \left( \sum_{j=1}^{5} \cos((j+1)x_i + j) \right) \]

subject to \( x_i \in [-10, 10], i = 1, 2. \)

Usage

makeShubertFunction()

Value

smoof_single_objective_function

References

makeSingleObjectiveFunction

Generator for single-objective target functions.

Description

Generator for single-objective target functions.

Usage

makeSingleObjectiveFunction(name = NULL, id = NULL,
                              description = NULL, fn, has.simple.signature = TRUE,
                              vectorized = FALSE, par.set, noisy = FALSE, fn.mean = NULL,
                              minimize = TRUE, constraint.fn = NULL, tags = character(0),
                              global.opt.params = NULL, global.opt.value = NULL,
                              local.opt.params = NULL, local.opt.values = NULL)

Arguments

name  [character(1)]
      Function name. Used for the title of plots for example.

id    [character(1)] | NULL
      Optional short function identifier. If provided, this should be a short name without whitespaces and now special characters beside the underscore. Default is NULL, which means no ID at all.

description [character(1)] | NULL
      Optional function description.

fn     [function]
      Objective function.

has.simple.signature [logical(1)]
      Set this to TRUE if the objective function expects a vector as input and FALSE if it expects a named list of values. The latter is needed if the function depends on mixed parameters. Default is TRUE.

vectorized [logical(1)]
      Can the objective function handle “vector” input, i.e., does it accept matrix of parameters? Default is FALSE.

par.set   [ParamSet]
      Parameter set describing different aspects of the objective function parameters, i.e., names, lower and/or upper bounds, types and so on. See makeParamSet for further information.

noisy    [logical(1)]
      Is the function noisy? Defaults to FALSE.
fn.mean [function]
Optional true mean function in case of a noisy objective function. This function should have the same mean as fn.

minimize [logical(1)]
Set this to TRUE if the function should be minimized and to FALSE otherwise. The default is TRUE.

constraint.fn [function | NULL]
Function which returns a logical vector indicating whether certain conditions are met or not. Default is NULL, which means that there are no constraints beside possible box constraints defined via the par.set argument.

tags [character]
Optional character vector of tags or keywords which characterize the function, e.g. “unimodal”, “separable”. See getAvailableTags for a character vector of allowed tags.

global.opt.params [list | numeric | data.frame | matrix | NULL]
Default is NULL which means unknown. Passing a numeric vector will be the most frequent case (numeric only functions). In this case there is only a single global optimum. If there are multiple global optima, passing a numeric matrix is the best choice. Passing a list or a data.frame is necessary if your function is mixed, e.g., it expects both numeric and discrete parameters. Internally, however, each representation is casted to a data.frame for reasons of consistency.

global.opt.value [numeric(1) | NULL]
Global optimum value if known. Default is NULL, which means unknown. If only the global.opt.params are passed, the value is computed automatically.

local.opt.params [list | numeric | data.frame | matrix | NULL]
Default is NULL, which means the function has no local optima or they are unknown. For details see the description of global.opt.params.

local.opt.values [numeric | NULL]
Value(s) of local optima. Default is NULL, which means unknown. If only the local.opt.params are passed, the values are computed automatically.

**Value**

function Objective function with additional stuff attached as attributes.

**Examples**

```r
library(ggplot2)

fn = makeSingleObjectiveFunction(
  name = "Sphere Function",
  fn = function(x) sum(x^2),
  par.set = makeNumericParamSet("x", len = 1L, lower = -5L, upper = 5L),
  global.opt.params = list(x = 0)
```
makeSixHumpCamelFunction

Three-Hump Camel Function

Description

Two dimensional single-objective test function with six local minima oh which two are global. The surface is similar to the back of a camel. That is why it is called Camel function. The implementation is based on the formula:

\[ f(x) = (4 - 2.1x_1^2 + x_1^{0.75})x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \]

with box constraints \( x_1 \in [-3, 3] \) and \( x_2 \in [-2, 2] \).

Usage

makeSixHumpCamelFunction()

Value

smoof_single_objective_function
References


makeSphereFunction

Sphere Function

Description

Also known as the “De Jong function 1”. Convex, continuous function calculated via the formula

\[ f(x) = \sum_{i=1}^{n} x_i^2 \]

with box-constraints \( x_i \in [-5.12, 5.12], i = 1, \ldots, n. \)

Usage

makeSphereFunction(dimensions)

Arguments

dimensions [integer(1)]
Size of corresponding parameter space.

Value

smoof_single_objective_function

References


makeStyblinskiTangFunction

Styblinski-Tang function

Description

This function is based on the definition

\[ f(x) = \frac{1}{2} \sum_{i=1}^{2} (x_i^4 - 16x_i^2 + 5x_i) \]

with box-constraints given by \( x_i \in [-5, 5], i = 1, 2. \)
makeSumOfDifferentSquaresFunction

**Usage**

makeStyblinskiTangFunction()

**Value**

smoof_single_objective_function

**References**


makeSumOfDifferentSquaresFunction

*Sum of Different Squares Function*

**Description**

Simple unimodal test function similar to the Sphere and Hyper-Ellipsoidal functions. Formula:

\[ f(x) = \sum_{i=1}^{n} |x_i|^{i+1}. \]

**Usage**

makeSumOfDifferentSquaresFunction(dimensions)

**Arguments**

- `dimensions` [integer(1)]
  Size of corresponding parameter space.

**Value**

smoof_single_objective_function
makeSwiler2014Function

Swiler2014 function.

Description

Mixed parameter space with one discrete parameter $x_1 \in \{1, 2, 3, 4, 5\}$ and two numerical parameters $x_1, x_2 \in [0, 1]$. The function is defined as follows:

$$f(x) = \sin(2\pi x_3 - \pi) + 7 \sin^2(2\pi x_2 - \pi)$$ if $x_1 = 1$

$$f(x) = \sin(2\pi x_3 - \pi) + 7 \sin^2(2\pi x_2 - \pi) + 12 \sin(2\pi x_3 - \pi)$$ if $x_1 = 2$

$$f(x) = \sin(2\pi x_3 - \pi) + 7 \sin^2(2\pi x_2 - \pi) + 12 \sin(2\pi x_3 - \pi)$$ if $x_1 = 3$

$$f(x) = \sin(2\pi x_3 - \pi) + 7 \sin^2(2\pi x_2 - \pi) + 3.5 \sin(2\pi x_3 - \pi)$$ if $x_1 = 4$

$$f(x) = \sin(2\pi x_3 - \pi) + 7 \sin^2(2\pi x_2 - \pi)$$ if $x_1 = 5$.

Usage

makeSwiler2014Function()

Value

smoof_single_objective_function

makeThreeHumpCamelFunction

Three-Hump Camel Function

Description

This two-dimensional function is based on the definition

$$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$$

subject to $-5 \leq x_i \leq 5$.

Usage

makeThreeHumpCamelFunction()

Value

smoof_single_objective_function

References

**makeTrecanniFunction  Trecanni Function**

**Description**

The Trecanni function belongs to the unimodal test functions. It is based on the formula

\[ f(x) = (x_1^4 - 4x_1^3 + 4x_1 + x_2^2). \]

The box-constraints \( x_i \in [-5, 5], i = 1, 2 \) define the domain of definition.

**Usage**

```r
makeTrecanniFunction()
```

**Value**

`smoof_single_objective_function`

**References**


---

**makeUFFunction  Generator for the functions UF1, ..., UF10 of the CEC 2009.**

**Description**

Generator for the functions UF1, ..., UF10 of the CEC 2009.

**Usage**

```r
makeUFFunction(dimensions, id)
```

**Arguments**

- `dimensions` [integer(1)]
  Size of corresponding parameter space.
- `id` [integer(1)]
  Instance identifier. Integer value between 1 and 10.

**Value**

`smoof_single_objective_function`
Note
The implementation is based on the original CPP implementation by Qingfu Zhang, Aimin Zhou, Shizheng Zhaoy, Ponnuthurai Nagaratnam Suganthany, Wudong Liu and Santosh Tiwar.

Author(s)
Jakob Bossek <j.bossek@gmail.com>

Description
The Viennet test problem VNT is designed for three objectives only. It has a discrete set of Pareto fronts. It is defined by the following formulae.

\[ f(x) = (f_1(x), f_2(x), f_3(x)) \]
with

\[ f_1(x) = 0.5(x_1^2 + x_2^2) + \sin(x_1^2 + x_2^2) \]
\[ f_2(x) = \frac{(3x_1 + 2x_2 + 4)^2}{8} + \frac{(x_1 - x_2 + 1)^2}{27} + 15 \]
\[ f_3(x) = \frac{1}{x_1^2 + x_2^2 + 1} - 1.1 \exp(-(x_1^2 + x_2^2)) \]
with box constraints \(-3 \leq x_1, x_2 \leq 3\).

Usage
makeViennetFunction()

Value
smoof_multi_objective_function

References
### Description

First test problem from the "Walking Fish Group" problem generator toolkit.

### Usage

```r
makeWFG1Function(n.objectives, k, l)
```

### Arguments

- **n.objectives** `[integer(1)]`
  Number of objectives.

- **k** `[integer(1)]`
  Number of position-related parameters. These will automatically be the first k elements from the input vector. This value has to be a multiple of `n.objectives - 1`.

- **l** `[integer(1)]`
  Number of distance-related parameters. These will automatically be the last l elements from the input vector.

### Details

Huband et al. recommend a value of `k = 4L` position-related parameters for bi-objective problems and `k = 2L * (n.objectives - 1L)` for many-objective problems. Furthermore the authors recommend a value of `l = 20` distance-related parameters. Therefore, if k and/or l are not explicitly defined by the user, their values will be set to the recommended values per default.

### Value

`smoof_multi_objective_function`

### References

**makeWFG2Function**

**WFG2 Function**

**Description**

Second test problem from the "Walking Fish Group" problem generator toolkit.

**Usage**

```r
makeWFG2Function(n.objectives, k, l)
```

**Arguments**

- **n.objectives**  
  [integer(1)]  
  Number of objectives.
- **k**  
  [integer(1)]  
  Number of position-related parameters. These will automatically be the first k elements from the input vector. This value has to be a multiple of n.objectives - 1.
- **l**  
  [integer(1)]  
  Number of distance-related parameters. These will automatically be the last l elements from the input vector. This value has to be a multiple of 2.

**Details**

Huband et al. recommend a value of k = 4L position-related parameters for bi-objective problems and k = 2L * (n.objectives -1L) for many-objective problems. Furthermore the authors recommend a value of l = 20 distance-related parameters. Therefore, if k and/or l are not explicitly defined by the user, their values will be set to the recommended values per default.

**Value**

`smoof_multi_objective_function`

**References**

makeWFG3Function  \hspace{0.5em} WFG3 Function

\subsection*{Description}
Third test problem from the "Walking Fish Group" problem generator toolkit.

\subsection*{Usage}
\begin{verbatim}
makeWFG3Function(n.objectives, k, l)
\end{verbatim}

\subsection*{Arguments}
\begin{itemize}
  \item \texttt{n.objectives} \hspace{0.5em} [integer(1)]
    Number of objectives.
  \item \texttt{k} \hspace{0.5em} [integer(1)]
    Number of position-related parameters. These will automatically be the first \(k\) elements from the input vector. This value has to be a multiple of \(n.objectives - 1\).
  \item \texttt{l} \hspace{0.5em} [integer(1)]
    Number of distance-related parameters. These will automatically be the last \(l\) elements from the input vector. This value has to be a multiple of 2.
\end{itemize}

\subsection*{Details}
Huband et al. recommend a value of \(k = 4L\) position-related parameters for bi-objective problems and \(k = 2L \times (n.objectives - 1L)\) for many-objective problems. Furthermore, the authors recommend a value of \(l = 20\) distance-related parameters. Therefore, if \(k\) and/or \(l\) are not explicitly defined by the user, their values will be set to the recommended values per default.

\subsection*{Value}
\texttt{smoof_multi_objective_function}

\subsection*{References}
Description
Fourth test problem from the "Walking Fish Group" problem generator toolkit.

Usage
makeWFG4Function(n.objectives, k, l)

Arguments
n.objectives [integer(1)]
Number of objectives.

k [integer(1)]
Number of position-related parameters. These will automatically be the first k elements from the input vector. This value has to be a multiple of n.objectives -1.

l [integer(1)]
Number of distance-related parameters. These will automatically be the last l elements from the input vector.

Details
Huband et al. recommend a value of k = 4L position-related parameters for bi-objective problems and k = 2L * (n.objectives -1L) for many-objective problems. Furthermore the authors recommend a value of l = 20 distance-related parameters. Therefore, if k and/or l are not explicitly defined by the user, their values will be set to the recommended values per default.

Value
smoof_multi_objective_function

References
makeWFG5Function  \hline

\textit{WFG5 Function}  

\textbf{Description} 

Fifth test problem from the "Walking Fish Group" problem generator toolkit.

\textbf{Usage} 

\begin{verbatim}
makeWFG5Function(n.objectives, k, l)
\end{verbatim}

\textbf{Arguments} 

\begin{verbatim}
n.objectives  [integer(1)]
    Number of objectives.
k  [integer(1)]
    Number of position-related parameters. These will automatically be the first \( k \) elements from the input vector. This value has to be a multiple of \( n\text{.objectives} -1 \).
l  [integer(1)]
    Number of distance-related parameters. These will automatically be the last \( l \) elements from the input vector.
\end{verbatim}

\textbf{Details} 

Huband et al. recommend a value of \( k = 4L \) position-related parameters for bi-objective problems and \( k = 2L \times (n\text{.objectives} -1L) \) for many-objective problems. Furthermore the authors recommend a value of \( l = 20 \) distance-related parameters. Therefore, if \( k \) and/or \( l \) are not explicitly defined by the user, their values will be set to the recommended values per default.

\textbf{Value} 

\begin{verbatim}
smoof\_multi\_objective\_function
\end{verbatim}

\textbf{References} 

Description

Sixth test problem from the "Walking Fish Group" problem generator toolkit.

Usage

makeWFG6Function(n.objectives, k, l)

Arguments

n.objectives [integer(1)]
Number of objectives.

k [integer(1)]
Number of position-related parameters. These will automatically be the first k elements from the input vector. This value has to be a multiple of n.objectives -1.

l [integer(1)]
Number of distance-related parameters. These will automatically be the last l elements from the input vector.

Details

Huband et al. recommend a value of \( k = 4L \) position-related parameters for bi-objective problems and \( k = 2L \times (n.objectives -1L) \) for many-objective problems. Furthermore the authors recommend a value of \( l = 20 \) distance-related parameters. Therefore, if \( k \) and/or \( l \) are not explicitly defined by the user, their values will be set to the recommended values per default.

Value

smoof_multi_objective_function

References

Description

Seventh test problem from the "Walking Fish Group" problem generator toolkit.

Usage

```r
makeWFG7Function(n.objectives, k, l)
```

Arguments

- `n.objectives` [integer(1)]
  Number of objectives.
- `k` [integer(1)]
  Number of position-related parameters. These will automatically be the first `k` elements from the input vector. This value has to be a multiple of `n.objectives - 1`.
- `l` [integer(1)]
  Number of distance-related parameters. These will automatically be the last `l` elements from the input vector.

Details

Huband et al. recommend a value of `k = 4L` position-related parameters for bi-objective problems and `k = 2L * (n.objectives -1L)` for many-objective problems. Furthermore the authors recommend a value of `l = 20` distance-related parameters. Therefore, if `k` and/or `l` are not explicitly defined by the user, their values will be set to the recommended values per default.

Value

`smoof_multi_objective_function`

References

**Description**

Eighth test problem from the "Walking Fish Group" problem generator toolkit.

**Usage**

```r
makeWFG8Function(n.objectives, k, l)
```

**Arguments**

- `n.objectives` [integer(1)]
  Number of objectives.

- `k` [integer(1)]
  Number of position-related parameters. These will automatically be the first `k` elements from the input vector. This value has to be a multiple of `n.objectives - 1`.

- `l` [integer(1)]
  Number of distance-related parameters. These will automatically be the last `l` elements from the input vector.

**Details**

Huband et al. recommend a value of `k = 4L` position-related parameters for bi-objective problems and `k = 2L * (n.objectives - 1L)` for many-objective problems. Furthermore the authors recommend a value of `l = 20` distance-related parameters. Therefore, if `k` and/or `l` are not explicitly defined by the user, their values will be set to the recommended values per default.

**Value**

`smoof_multi_objective_function`

**References**

makeWFG9Function  

**WFG9 Function**

**Description**

Ninth test problem from the "Walking Fish Group" problem generator toolkit.

**Usage**

```r
makeWFG9Function(n.objectives, k, l)
```

**Arguments**

- `n.objectives`  
  `[integer(1)]`  
  Number of objectives.

- `k`  
  `[integer(1)]`  
  Number of position-related parameters. These will automatically be the first `k` elements from the input vector. This value has to be a multiple of `n.objectives - 1`.

- `l`  
  `[integer(1)]`  
  Number of distance-related parameters. These will automatically be the last `l` elements from the input vector.

**Details**

Huband et al. recommend a value of `k = 4L` position-related parameters for bi-objective problems and `k = 2L * (n.objectives - 1L)` for many-objective problems. Furthermore, the authors recommend a value of `l = 20` distance-related parameters. Therefore, if `k` and/or `l` are not explicitly defined by the user, their values will be set to the recommended values per default.

**Value**

`snoof_multi_objective_function`

**References**

**Description**

Builds and returns the two-objective ZDT1 test problem. For \( m \) objective it is defined as follows:

\[
f(x) = (f_1(x_1), f_2(x))
\]

with

\[
f_1(x_1) = x_1, \quad f_2(x) = g(x)h(f_1(x_1), g(x))
\]

where

\[
g(x) = 1 + \frac{9}{m-1} \sum_{i=2}^{m} x_i, \quad h(f_1, g) = 1 - \sqrt[3]{\frac{f_1}{g}}
\]

and \( x_i \in [0, 1], i = 1, \ldots, m \)

**Usage**

`makeZDT1Function(dimensions)`

**Arguments**

- **dimensions** [integer(1)]
  Number of decision variables.

**Value**

`smoof_multi_objective_function`

**References**

Description

Builds and returns the two-objective ZDT2 test problem. The function is nonconvex and resembles the ZDT1 function. For $m$ objective it is defined as follows

$$f(x) = (f_1(x_1), f_2(x))$$

with

$$f_1(x_1) = x_1, f_2(x) = g(x)h(f_1(x_1), g(x))$$

where

$$g(x) = 1 + \frac{9}{m-1} \sum_{i=2}^{m} x_i, h(f_1, g) = 1 - \left(\frac{f_1}{g}\right)^2$$

and $x_i \in [0, 1], i = 1, \ldots, m$

Usage

makeZDT2Function(dimensions)

Arguments

dimensions [integer(1)]

Number of decision variables.

Value

smoof_multi_objective_function

References

Description

Builds and returns the two-objective ZDT3 test problem. For $m$ objective it is defined as follows

$$f(x) = (f_1(x_1), f_2(x))$$

with

$$f_1(x_1) = x_1, f_2(x) = g(x) h(f_1(x_1), g(x))$$

where

$$g(x) = 1 + \frac{9}{m-1} \sum_{i=2}^{m} x_i, h(f_1, g) = 1 - \sqrt{\frac{f_1(x)}{g(x)}} - \left( \frac{f_1(x)}{g(x)} \right) \sin(10\pi f_1(x))$$

and $x_i \in [0, 1], i = 1, \ldots, m$. This function has some discontinuities in the Pareto-optimal front introduced by the sine term in the $h$ function (see above). The front consists of multiple convex parts.

Usage

```r
makeZDT3Function(dimensions)
```

Arguments

- `dimensions` [integer(1)]
  Number of decision variables.

Value

`smoof_multi_objective_function`

References

**Description**

Builds and returns the two-objective ZDT4 test problem. For \( m \) objective it is defined as follows:

\[
f(x) = (f_1(x_1), f_2(x))
\]

with

\[
f_1(x_1) = x_1, f_2(x) = g(x)h(f_1(x_1), g(x))
\]

where

\[
g(x) = 1 + 10(m - 1) + \sum_{i=2}^{m} (x_i^2 - 10 \cos(4\pi x_i)), h(f_1, g) = 1 - \sqrt{\frac{f_1(x)}{g(x)}}
\]

and \( x_i \in [0, 1], i = 1, \ldots, m \). This function has many Pareto-optimal fronts and is thus suited to test the algorithms ability to tackle multimodal problems.

**Usage**

```r
makeZDT4Function(dimensions)
```

**Arguments**

- `dimensions` [integer(1)]
  
  Number of decision variables.

**Value**

`smoof_multi_objective_function`

**References**

Description

Builds and returns the two-objective ZDT6 test problem. For $m$ objective it is defined as follows

$$f(x) = (f_1(x), f_2(x))$$

with

$$f_1(x) = 1 - \exp(-4x_1) \sin^6(6\pi x_1), f_2(x) = g(x)h(f_1(x_1), g(x))$$

where

$$g(x) = 1 + 9 \left( \sum_{i=2}^{m} \frac{x_i}{m-1} \right)^{0.25}, h(f_1, g) = 1 - \left( \frac{f_1(x)}{g(x)} \right)^2$$

and $x_i \in [0, 1], i = 1, \ldots, m$. This function introduced two difficulties (see reference): 1. the density of solutions decreases with the closeness to the Pareto-optimal front and 2. the Pareto-optimal solutions are nonuniformly distributed along the front.

Usage

```r
makeZDT6Function(dimensions)
```

Arguments

dimensions [integer(1)]
Number of decision variables.

Value

`smoof_multi_objective_function`

References

makeZettlFunction  

**Zettl Function**

**Description**

The unimodal Zettl Function is based on the definition

\[ f(x) = (x_1^2 + x_2^2 - 2x_1)^2 + 0.25x_1 \]

with box-constraints \( x_i \in [-5, 10], i = 1, 2 \).

**Usage**

makeZettlFunction()

**Value**

smoof_single_objective_function

**References**


---

mnof  

**Helper function to create numeric multi-objective optimization test function.**

**Description**

This is a simplifying wrapper around makeMultiObjectiveFunction. It can be used if the function to generate is purely numeric to save some lines of code.

**Usage**

```r
mnof(name = NULL, id = NULL, par.len = NULL, par.id = "x",
     par.lower = NULL, par.upper = NULL, n.objectives,
     description = NULL, fn, vectorized = FALSE, noisy = FALSE,
     fn.mean = NULL, minimize = rep(TRUE, n.objectives),
     constraint.fn = NULL, ref.point = NULL)
```
Arguments

name [character(1)]
Function name. Used for the title of plots for example.

id [character(1) | NULL]
Optional short function identifier. If provided, this should be a short name without whitespaces and no special characters beside the underscore. Default is NULL, which means no ID at all.

par.len [integer(1)]
Length of parameter vector.

par.id [character(1)]
Optional name of parameter vector. Default is “x”.

par.lower [numeric]
Vector of lower bounds. A single value of length 1 is automatically replicated to n.pars. Default is -Inf.

par.upper [numeric]
Vector of upper bounds. A single value of length 1 is automatically replicated to n.pars. Default is Inf.

n.objectives [integer(1)]
Number of objectives of the multi-objective function.

description [character(1) | NULL]
Optional function description.

fn [function]
Objective function.

vectorized [logical(1)]
Can the objective function handle “vector” input, i.e., does it accept matrix of parameters? Default is FALSE.

noisy [logical(1)]
Is the function noisy? Defaults to FALSE.

fn.mean [function]
Optional true mean function in case of a noisy objective function. This function should have the same mean as fn.

minimize [logical]
Logical vector of length n.objectives indicating if the corresponding objectives shall be minimized or maximized. Default is the vector with all components set to TRUE.

constraint.fn [function | NULL]
Function which returns a logical vector indicating whether certain conditions are met or not. Default is NULL, which means that there are no constraints beside possible box constraints defined via the par.set argument.

ref.point [numeric]
Optional reference point in the objective space, e.g., for hypervolume computation.
Examples

```r
# first we generate the 10d sphere function the long way
fn = makeMultiObjectiveFunction(
  name = "Testfun",
  fn = function(x) c(sum(x^2), exp(sum(x^2))),
  par.set = makeNumericParamSet(
    len = 10L, id = "a",
    lower = rep(-1.5, 10L), upper = rep(1.5, 10L)
  ),
  n.objectives = 2L
)

# ... and now the short way
fn = mnof(
  name = "Testfun",
  fn = function(x) c(sum(x^2), exp(sum(x^2))),
  par.len = 10L, par.id = "a", par.lower = -1.5, par.upper = 1.5,
  n.objectives = 2L
)
```

Description

Generate ggplot2 object.

Usage

```r
## S3 method for class 'smoof_function'
plot(x, ...)
```

Arguments

- `x` [smoof_function]
  - Function.

- `...` [any]
  - Further parameters passed to the corresponding plot functions.

Value

- Nothing
plot1DNumeric  

Plot an one-dimensional function.

Description
Plot an one-dimensional function.

Usage
plot1DNumeric(x, show.optimum = FALSE, main = getName(x),
               n.samples = 500L, ...)

Arguments
x             [smoof_function]
Function.
show.optimum  [logical(1)]
If the function has a known global optimum, should its location be plotted by a
point or multiple points in case of multiple global optima? Default is FALSE.
main          [character(1L)]
Plot title. Default is the name of the smoof function.
n.samples     [integer(1)]
Number of locations to be sampled. Default is 500.
...
[any]         Further parameters passed to plot function.

Value
Nothing

plot2DNumeric  

Plot a two-dimensional numeric function.

Description
Either a contour-plot or a level-plot.

Usage
plot2DNumeric(x, show.optimum = FALSE, main = getName(x),
              render.levels = FALSE, render.contours = TRUE, n.samples = 100L,
              ...)

...
plot3D

Surface plot of two-dimensional test function.

Description

Surface plot of two-dimensional test function.

Usage

plot3D(x, length.out = 100L, package = "plot3D", ...)

Arguments

x [smoof_function]
Two-dimensional smoof function.

length.out [integer(1)]
Determines the “smoothness” of the grid. The higher the value, the smoother the function landscape looks like. However, you should avoid setting this parameter to high, since with the contour option set to TRUE the drawing can take quite a lot of time. Default is 100.
package [character(1)]
String describing the package to use for 3D visualization. At the moment “plot3D” (package \texttt{plot3D}) and “plotly” (package \texttt{plotly}) are supported. The latter opens a highly interactive plot in a web browser and is thus suited well to explore a function by hand. Default is “plot3D”.

... [any]
Further parameters passed to method used for visualization (which is determined by the package argument).

Examples

library(plot3D)
fn = makeRastriginFunction(dimensions = 2L)
## Not run:
# use the plot3D::persp3D method (default behaviour)
plot3D(fn)
plot3D(fn, contour = TRUE)
plot3D(fn, image = TRUE, phi = 30)

# use plotly::plot_ly for interactive plot
plot3D(fn, package = "plotly")

## End(Not run)
shouldBeMinimized  

Check if function should be minimized.

Description

Functions can have an associated global optimum. In this case one needs to know whether the optimum is a minimum or a maximum.

Usage

shouldBeMinimized(fn)

Arguments

fn [smoof_function]
Objective function.

Value

logical  Each component indicates whether the corresponding objective should be minimized.

smoof_function  Smoof function

Description

Regular R function with additional classes smoof_function and one of smoof_single_objective_function or codesmoof_multi_objective_function. Both single- and multi-objective functions share the following attributes.

name [character(1)] Optional function name.
id [character(1)] Short identifier.
description [character(1)] Optional function description.
has.simple.signature  TRUE if the target function expects a vector as input and FALSE if it expects a named list of values.
par.set [ParamSet] Parameter set describing different aspects of the target function parameters, i.e., names, lower and/or upper bounds, types and so on.
n.objectives [integer(1)] Number of objectives.
o. noisy [logical(1)] Boolean indicating whether the function is noisy or not.
fn.mean [function] Optional true mean function in case of a noisy objective function.
minimize [logical(1)] Logical vector of length n.objectives indicating which objectives shall be minimized/maximized.
vectorized [logical(1)] Can the handle “vector” input, i.e., does it accept matrix of parameters?
constraint.fn [function ] Optional function which returns a logical vector with each component indicating whether the corresponding constraint is violated.

Furthermore, single-objective function may contain additional parameters with information on local and/or global optima as well as characterizing tags.

tags [character ] Optional character vector of tags or keywords.

global.opt.params [data.frame ] Data frame of parameter values of global optima.

global.opt.value [numeric(1) ] Function value of global optima.

local.opt.params [data.frame ] Data frame of parameter values of local optima.

global.opt.value [numeric ] Function values of local optima.

Currently tagging is not possible for multi-objective functions. The only additional attribute may be a reference point:

ref.point [numeric ] Optional reference point of length n.objectives.

---

snof

Helper function to create numeric single-objective optimization test function.

Description

This is a simplifying wrapper around makeSingleObjectiveFunction. It can be used if the function to generate is purely numeric to save some lines of code.

Usage

snof(name = NULL, id = NULL, par.len = NULL, par.id = "x", par.lower = NULL, par.upper = NULL, description = NULL, fn, vectorized = FALSE, noisy = FALSE, fn.mean = NULL, minimize = TRUE, constraint.fn = NULL, tags = character(0), global.opt.params = NULL, global.opt.value = NULL, local.opt.params = NULL, local.opt.values = NULL)

Arguments

name [character(1)]
Function name. Used for the title of plots for example.

id [character(1) | NULL]
Optional short function identifier. If provided, this should be a short name without whitespaces and now special characters beside the underscore. Default is NULL, which means no ID at all.

par.len [integer(1)]
Length of parameter vector.

par.id [character(1)]
Optional name of parameter vector. Default is “x”.

---
par.lower [numeric]
Vector of lower bounds. A single value of length 1 is automatically replicated to n.pars. Default is -Inf.

par.upper [numeric]
Vector of upper bounds. A single value of length 1 is automatically replicated to n.pars. Default is Inf.

description [character(1) | NULL]
Optional function description.

fn [function]
Objective function.

vectorized [logical(1)]
Can the objective function handle “vector” input, i.e., does it accept matrix of parameters? Default is FALSE.

noisy [logical(1)]
Is the function noisy? Defaults to FALSE.

fn.mean [function]
Optional true mean function in case of a noisy objective function. This function should have the same mean as fn.

minimize [logical(1)]
Set this to TRUE if the function should be minimized and to FALSE otherwise. The default is TRUE.

constraint.fn [function | NULL]
Function which returns a logical vector indicating whether certain conditions are met or not. Default is NULL, which means, that there are no constraints beside possible box constraints defined via the par.set argument.

tags [character]
Optional character vector of tags or keywords which characterize the function, e.g., “unimodal”, “separable”. See getAvailableTags for a character vector of allowed tags.

global.opt.params [list | numeric | data.frame | matrix | NULL]
Default is NULL which means unknown. Passing a numeric vector will be the most frequent case (numeric only functions). In this case there is only a single global optimum. If there are multiple global optima, passing a numeric matrix is the best choice. Passing a list or a data.frame is necessary if your function is mixed, e.g., it expects both numeric and discrete parameters. Internally, however, each representation is casted to a data.frame for reasons of consistency.

global.opt.value [numeric(1) | NULL]
Global optimum value if known. Default is NULL, which means unknown. If only the global.opt.params are passed, the value is computed automatically.

local.opt.params [list | numeric | data.frame | matrix | NULL]
Default is NULL, which means the function has no local optima or they are unknown. For details see the description of global.opt.params.
local.opt.values
   [numeric | NULL]
Value(s) of local optima. Default is NULL, which means unknown. If only the
local.opt.params are passed, the values are computed automatically.

Examples

# first we generate the 10d sphere function the long way
fn = makeSingleObjectiveFunction(
   name = "Testfun",
   fn = function(x) sum(x^2),
   par.set = makeNumericParamSet(
      len = 10L, id = "a",
      lower = rep(-1.5, 10L), upper = rep(1.5, 10L)
   )
)

# ... and now the short way
fn = snof(
   name = "Testfun",
   fn = function(x) sum(x^2),
   par.len = 10L, par.id = "a", par.lower = -1.5, par.upper = 1.5
)

violatesConstraints Checks whether constraints are violated.

Description

Checks whether constraints are violated.

Usage

violatesConstraints(fn, values)

Arguments

fn [smoof_function]
   Objective function.

values [numeric]
   List of values.

Value

logical(1)
visualizeParetoOptimalFront

Pareto-optimal front visualization.

Description

Quickly visualize the Pareto-optimal front of a bi-criteria objective function by calling the EMOA `nsga2` and extracting the approximated Pareto-optimal front.

Usage

`visualizeParetoOptimalFront(fn, ...)`

Arguments

- `fn` [smoof_multi_objective_function]
  Multi-objective smoof function.

- `...` [any]
  Arguments passed to `nsga2`.

Value

`ggplot`

Examples

```r
# Here we visualize the Pareto-optimal front of the bi-objective ZDT3 function
fn = makeZDT3Function(dimensions = 3L)
vis = visualizeParetoOptimalFront(fn)

# Alternatively we can pass some more algorithm parameters to the NSGA2 algorithm
vis = visualizeParetoOptimalFront(fn, popsize = 1000L)
```
Index

addCountingWrapper, 6, 12, 22, 27
addLoggingWrapper, 7, 16, 22, 27
autoplot.smoof_function, 8
computeExpectedRunningTime, 10
conversion, 11
convertToMaximization (conversion), 11
convertToMinimization (conversion), 11
doesCountEvaluations, 12
filterFunctionsByTags, 13, 61
getAvailableTags, 13, 13, 25, 91, 120
getDescription, 14
getGlobalOptimum, 14
getID, 15
getLocalOptimum, 15
getLoggedValues, 16
getLowerBoxConstraints, 17
getMeanFunction, 17
getsName, 18
getNumberOfEvaluations, 6, 18
getNumberOfObjectives, 19
getNumberOfParameters, 19
ggetParamSet, 20
getRefPoint, 20
getTags, 21
getUpperBoxConstraints, 21
getWrappedFunction, 22
ggplot, 9, 122
hasBoxConstraints, 22
hasConstraints, 23
hasGlobalOptimum, 23
hasLocalOptimum, 24
hasOtherConstraints, 24
hasTags, 25
isMultiobjective, 25
isNoisy, 26
isSingleobjective, 26
isSmooffunction, 27
isVectorized, 27
isWrappedSmooffunction, 28
makeAckleyFunction, 28
makeAdjimanFunction, 29
makeAlpine01Function, 29
makeAlpine02Function, 30
makeAluffiPentiniFunction, 31
makeBartelsConnFunction, 31
makeBBOBFunction, 32
makeBealeFunction, 33
makeBentCigarFunction, 33
makeBiobjBBOBFunction, 34
makeBirdFunction, 35
makeBiSphereFunction, 35
makeBK1Function, 36
makeBohachevskyyN1Function, 36
makeBoothFunction, 37
makeBraninFunction, 37
makeBrentFunction, 38
makeBrownFunction, 39
makeBukinN2Function, 39, 40, 41
makeBukinN4Function, 40, 40, 41
makeBukinN6Function, 40, 41
makeCarromTableFunction, 41
makeChichinadzeFunction, 42
makeChungReynoldsFunction, 42
makeComplexFunction, 43
makeCosineMixtureFunction, 43
makeCrossInTrayFunction, 44
makeCubeFunction, 45
makeDeckkersAartsFunction, 45
makeDeflectedCorrugatedSpringFunction, 46
makeDentFunction, 46
makeDixonPriceFunction, 47
makeDoubleSumFunction, 48
makeDTLZ1Function, 48
makeDTLZ2Function, 49
makeDTLZ3Function, 50
makeDTLZ4Function, 51
makeDTLZ5Function, 52
makeDTLZ6Function, 53
makeDTLZ7Function, 54
makeEasomFunction, 55
makeE1Function, 56
makeE2Function, 57
makeEggCrateFunction, 58
makeEggHolderFunction, 58
makeElAttarVidyasagarDuttaFunction, 59
makeEngvallFunction, 59
makeExponentialFunction, 60
makeFreudensteinRothFunction, 60
makeFunctionsByName, 61
makeGeneralizedDropWaveFunction, 62
makeGiuntaFunction, 62
makeGoldsteinPriceFunction, 63
makeGOMOPFunction, 63
makeGriewankFunction, 64
makeHansenFunction, 65
makeHartmannFunction, 65
makeHimmelblauFunction, 66
makeHolderTableN1Function, 67, 68
makeHolderTableN2Function, 67, 67
makeHosakiFunction, 68
makeHyperEllipsoidFunction, 69
makeInvertedVincentFunction, 69
makeJennrichSampsonFunction, 70
makeJudgeFunction, 71
makeKeaneFunction, 71
makeKearfottFunction, 72
makeKursaweFunction, 72
makeLeonFunction, 73
makeMatyasFunction, 73
makeMcCormickFunction, 74
makeMichalewiczFunction, 74
makeModifiedRastriginFunction, 75
makeMOP1Function, 76
makeMOP2Function, 76
makeMOP3Function, 77
makeMOP4Function, 77
makeMOP5Function, 78
makeMOP6Function, 78
makeMOP7Function, 79
makeMOPM2Function, 79
makeMultiObjectiveFunction, 80, 112
makeParamSet, 81, 90
makePeriodicFunction, 82
makePowellSumFunction, 83
makePriceN1Function, 83, 84, 85
makePriceN2Function, 84, 84, 85
makePriceN4Function, 84, 85
makeRastriginFunction, 85
makeRosenbrockFunction, 86
makeSchafferN2Function, 87
makeSchafferN4Function, 87
makeSchwefelFunction, 88
makeShekelFunction, 88
makeShubertFunction, 89
makeSingleObjectiveFunction, 90, 119
makeSixHumpCamelFunction, 92
makeSphereFunction, 69, 93
makeStyblinskiTangFunction, 93
makeSumOfDifferentSquaresFunction, 94
makeSwiler2014Function, 95
makeThreeHumpCamelFunction, 95
makeTrecanniFunction, 96
makeUFunc, 96
makeViennetFunction, 97
makeWFG1Function, 98
makeWFG2Function, 99
makeWFG3Function, 100
makeWFG4Function, 101
makeWFG5Function, 102
makeWFG6Function, 103
makeWFG7Function, 104
makeWFG8Function, 105
makeWFG9Function, 106
makeZDT1Function, 107
makeZDT2Function, 108
makeZDT3Function, 109
makeZDT4Function, 110
makeZDT6Function, 111
makeZettlFunction, 112
mnof, 112
nsga2, 122
ParamSet, 8, 20, 81, 90, 118
plot.smoof_function, 114
plot1DNumeric, 115
plot2DNumeric, 115
plot3D, 116
resetEvaluationCounter, 6, 117
INDEX

shouldBeMinimized, 118
smoof-package, 5
smoof_function, 118
smoof_multi_objective_function
  (smoof_function), 118
smoof_single_objective_function
  (smoof_function), 118
snof, 119
violatesConstraints, 121
visualizeParetoOptimalFront, 122