Package ‘soobench’

February 9, 2020

Title Single Objective Optimization Benchmark Functions

Description Collection of different single objective test functions useful for benchmarks and algorithm development.

Author Olaf Mersmann <olafm@p-value.net>, Bernd Bischl <bernd_bischl@gmx.net>, Jakob Bossek <jakob.bossek@tu-dortmund.de> and Leonard Judt <judt@gmx.net>

Maintainer Eric Kalosa-Kenyon <eric@opsani.com>

License BSD_2_clause + file LICENSE

LazyData yes

Version 1.9.18

Suggests rgl, testthat

RoxygenNote 7.0.2

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-02-09 16:30:02 UTC

R topics documented:

counting_function — *Count number of function evaluations*

Description

Return a new function which is identical to the soo_function passed in except that all function evaluations are counted.

Usage

```
counting_function(fn)
```

Arguments

fn	[soo_function] A test function.

See Also

number_of_evaluations, reset_evaluation_counter

Examples

```
f <- counting_function(generate_double_sum_function(5))
number_of_evaluations(f)

y <- f(random_parameters(1, f))
number_of_evaluations(f)

reset_evaluation_counter(f)
number_of_evaluations(f)

y <- f(random_parameters(21, f))
number_of_evaluations(f)
```

first_hitting_times — *Return numerical vector of first hitting times, counted in function evaluations.*

Description

Return numerical vector of first hitting times, counted in function evaluations.

Usage

```
first_hitting_times(x, target_levels)
```

Arguments

x	Object from which the reached target values are to be extracted. Methods for numeric vectors and `record_target_values_function` class objects are provided.
target_levels	Numerical vector of target levels which should ideally be reached.

See Also

`recording_function`

`function_id` *Function ID*

Description

Get a short id for the function that can be used in filenames and such. It is guaranteed that the ID contains only "safe" characters in the range A-Z,a-z,0-9,_,-.

Usage

```
function_id(fn)
```

Arguments

fn	[`soo_function`] Function to name.

Value

ID of function. Guaranteed to be unique among all test functions.

`function_name` *Function name*

Description

Get a pretty function name for a benchmark function.

Usage

```
function_name(fn)
```

Arguments

fn	[`soo_function`] Function to name.

Value

Name of function.

generate_ackley_function

Ackley test function generator

Description

Generator for the Ackley test function. The Ackley function is defined as

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i^2)}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1)$$

Usage

```
generate_ackley_function(dimensions)
```

Arguments

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

Value

A `soo_function`.

References

D. H. Ackley. A connectionist machine for genetic hillclimbing. Kluwer Academic Publishers, Boston, 1987

Examples

```
f <- generate_ackley_function(2)
plot(f, rank=TRUE)
```

generate_batman_function

Generator for the Batman function.

Description

The definition used is

Usage

```
generate_batman_function(dimensions, alpha = 0.25)
```

Arguments

dimensions	[integer(1)] Size of parameter space.
alpha	[numeric(1)] Parameter for control of groove shape

Details

$$f(x) = \left[\left((x^T x)^2 - (\sum_{i=1}^{n})^2\right)^2\right]^{\alpha} + \frac{1}{N}\left(\frac{1}{2}x^T x + \sum_{i=1}^{n} x_i\right) + \frac{1}{2}$$

Value

A soo_function.

References

H.-G. Beyer and S. Finck. HappyCat - A Simple Function Class Where Well-Known Direct Search Algorithms Do Fail. In: PPSN XII (Parallel Problem Solving from Nature), 367-376, Springer, Heidelberg, 2012.

Examples

```
f <- generate_batman_function(2, 1/4)
plot(f)
```

---

generate_bbob2009_function

(Noisy) BBOB 2009 test function generator.

Description

(Noisy) BBOB 2009 test function generator.

Usage

```
generate_bbob2009_function(dimensions, fid, iid)

generate_noisy_bbob2009_function(dimensions, fid, iid, noiseSeed = 1L)
```

Arguments

dimensions	[integer(1)] Size of parameter space. Must be greater than 1 and less than or equal to 40.
fid	[integer(1)] Function ID, valid values are 1 to 24.
iid	[integer(1)] Instance ID, defaults to 1L.
noiseSeed	[integer(1)] Seed for the noise random number generator, defaults to 1L.

Value

A `soo_function`.

Note

Due to the way the instances are generated, the function value at the optimal parameter settings (as returned by `global_minimum`) may differ slightly from the optimal function value. These differences are in the order of 10^{-16}.

Also note that the random number generator used for the noisy test functions is shared by all instantiated test functions. This means that if you run multiple trials in parallel within the same interpreter, your results will not necessarily be repeatable.

Author(s)

R interface by Olaf Mersmann. Original C code graciously provided by the BBOB team (Anne Auger, Hans-Georg Beyer, Nikolaus Hansen, Steffen Finck, Raymond Ros, Marc Schoenauer, Darrell Whitley)

References

For a complete description of the 24 test functions and much more background information please see the BBOB homepage

---

generate_beale_function

Generator for the Beale test function.

Description

The function is defined as:

Usage

```
generate_beale_function()
```

Details

$$f(x) = (1.5 - x_1 * (1 - x_2))^2 + (2.25 - x_1 * (1 - x_2^2))^2 + (2.625 - x_1 * (1 - x_2^3))^2$$

Value

A `soo_function`.

Examples

```
f <- generate_beale_function()
plot(f, rank=TRUE)
```

`generate_branin_function`

Generator for the Branin test function.

Description

This function is a 2D test function. The generator does not take any parameters. It is defined as

Usage

```
generate_branin_function()
```

Details

$$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$$

Value

A `soo_function`.

References

F. H. Branin. 1972. Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. IBM J. Res. Dev. 16, 5 (September 1972), 504-522.

Examples

```
f <- generate_branin_function()
plot(f, rank=TRUE)
```

`generate_chained_cb3_ii_function`

Chained CB3 II test function generator.

Description

The chained CB3 II test function is defined as

Usage

```
generate_chained_cb3_ii_function(dimensions)
```

Arguments

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

Details

$$f(x) = \max\left\{\sum_{i=1}^{n-1}\left(x_i^4 + x_{i+1}^2\right), \sum_{i=1}^{n-1}\left((2-x_i)^2 + (2-x_{i+1})^2\right), \sum_{i=1}^{n-1}\left(2e^{-x_i+x_{i+1}}\right)\right\}$$

Value

A soo_function.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

---

generate_chained_cb3_i_function

Chained CB3 I test function generator.

Description

The chained CB3 I test function is defined as

Usage

```
generate_chained_cb3_i_function(dimensions)
```

Arguments

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

Details

$$f(x) = \sum_{i=1}^{n-1} \max\left\{x_i^4 + x_{i+1}^2, (2-x_i)^2 + (2-x_{i+1})^2, 2e^{-x_i+x_{i+1}}\right\}$$

Value

A soo_function.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

`generate_chained_crescent_II_function`
Chained Crescent II function generator.

Description

The chained Crescent II function is defined as

Usage

```
generate_chained_crescent_II_function(dimensions)
```

Arguments

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

Details

$$f(x) = \sum_{i=1}^{n-1} \max \left\{ x_i^2 + (x_{i+1}^2 - 1)^2 + x_{i+1} - 1), -x_i^2 - (x_{i+1}^2 - 1)^2 + x_{i+1} 11 \right\}$$

Value

A `soo_function`.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

`generate_chained_crescent_I_function`
Chained Crescent I function generator.

Description

The chained Crescent I function is defined as

Usage

```
generate_chained_crescent_I_function(dimensions)
```

Arguments

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

Details

$$f(x) = \max\left\{\sum_{i=1}^{n-1}\left(x_i^2 + (x_{i+1}^2 - 1)^2 + x_{i+1} - 1\right), \sum_{i=1}^{n-1}\left(-x_i^2 - (x_{i+1}^2 - 1)^2 + x_{i+1} 11\right)\right\}$$

Value

A `soo_function`.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

---

generate_chained_LQ_function

Chained LQ test function generator.

Description

The chained LQ test function is defined as

Usage

```
generate_chained_LQ_function(dimensions)
```

Arguments

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

Details

$$f(x) = \sum_{i=1}^{n-1} \max\left\{-x_i - x_{i+1}, -x_i - x_{i+1} + (x_i^2 + x_{i+1}^2 - 1)\right\}$$

Value

A `soo_function`.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

generate_chained_mifflin_function
Chained Mifflin 2 function generator.

Description

The chained Mifflin function 2 is defined as

Usage

```
generate_chained_mifflin_function(dimensions)
```

Arguments

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

Details

$$f(x) = \sum_{i=1}^{n-1} \left(-x_i + 2(x_i^2 + x_{i+1}^2 - 1) + 1.75|x_i^2 + x_{i+1}^2 - 1|\right)$$

Value

A soo_function.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

generate_discus_function
Discus test function generator.

Description

The discus test function is similar to a high condition ellipsoid. It is defined as

Usage

```
generate_discus_function(dimensions)
```

Arguments

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

Details

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

Value

A soo_function.

Note that the functions returned by this generator only differ in the elongated axis from those returned by generate_ellipsoidal_function.

---

generate_double_sum_function

Double sum test function generator.

Description

The definition used is

Usage

```
generate_double_sum_function(dimensions)
```

Arguments

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

Details

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

Value

A soo_function.

References

H.-P. Schwefel. Evolution and Optimum Seeking. John Wiley & Sons, New York, 1995.

generate_ellipsoidal_function

Generator for ellipsoidal test functions.

Description

The ellipsoidal test function is a badly conditioned variant of the sphere function. The definition used here is

Usage

```
generate_ellipsoidal_function(dimensions)
```

Arguments

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

Details

$$f(x) = \sum_{i=1}^{n} 10^{6\frac{i}{n}} x_i^2$$

Value

A `soo_function`.

Note that the functions returned by this generator only differ in the elongated axis from those returned by `generate_discus_function`.

Examples

```
f <- generate_ellipsoidal_function(2)
plot(f, rank=TRUE)
```

generate_generalized_maxq_function

Generalized MAXQ test function generator.

Description

The generalized MAXW test function is defined as

Usage

```
generate_generalized_maxq_function(dimensions)
```

Arguments

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

Details

$$f(x) = \max_{1 \leq i \neq n} x_i^2$$

Value

A soo_function.

References

Haarala, M. and Miettinen, K. and Maelelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

---

generate_generalized_mxhilb_function

Generalized MXHILB test function generator.

Description

The generalized MXHILB test function is defined as

Usage

```
generate_generalized_mxhilb_function(dimensions)
```

Arguments

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

Details

$$f(x) = \max_{1 \leq i \neq n} \left| \sum_{j=1}^{n} \frac{x_j}{i + j - 1} \right|$$

Value

A soo_function.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

`generate_goldstein_price_function`
Generator for the Goldstein-Price function.

Description

The definition used is

Usage

```
generate_goldstein_price_function()
```

Details

$$f(x) = \left(1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right) \left(30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48\right.\right.$$

Value

A `soo_function`.

References

Goldstein, A. A. and Price, I. F. On descent from local minima, Mathematics of Computation 25, 569-574, 1971.

Examples

```
f <- generate_goldstein_price_function()
plot(f, rank=TRUE)
```

`generate_griewank_function`
Griewank test function generator.

Description

The definition used is

Usage

```
generate_griewank_function(dimensions)
```

Arguments

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

Details

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

Value

A soo_function.

References

A. O. Griewank. Generalized descent for global optimization. Journal of Optimization Theory and Applications 34:11-39, 1981.

---

generate_happycat_function

Generator for the Happycat function.

Description

The definition used is

Usage

```
generate_happycat_function(dimensions, alpha = 0.125)
```

Arguments

dimensions	[integer(1)] Size of parameter space.
alpha	[numeric(1)] Parameter for control of groove shape.

Details

$$f(x) = \left((x^T x - n)^2\right)^{\alpha} + \frac{1}{N}\left(\frac{1}{2}x^T x + \sum_{i=1}^{n} x_i\right) + \frac{1}{2}$$

Value

A soo_function.

References

H.-G. Beyer and S. Finck. HappyCat - A Simple Function Class Where Well-Known Direct Search Algorithms Do Fail. In: PPSN XII (Parallel Problem Solving from Nature), 367-376, Springer, Heidelberg, 2012.

Examples

```
f <- generate_happycat_function(2, 1/8)
plot(f)
```

`generate_himmelblau_function`

Generator for the Himmelblau test function.

Description

The function is defined as:

Usage

```
generate_himmelblau_function()
```

Details

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

Value

A `soo_function`.

Examples

```
f <- generate_himmelblau_function()
plot(f, rank=TRUE)
```

`generate_kotanchek_function`

Kotanchek test function generator.

Description

The definition used is

Usage

```
generate_kotanchek_function()
```

Details

$$f(x) = -\frac{\exp\left(-(x_1 - 1)^2\right)}{1.2 + (x_1 - 2.5)^2}$$

Value

A `soo_function`.

`generate_mexican_hat_function`
Mexican hat test function generator.

Description

The Mexican hat function is defined slightly differently by different people. The definition used here is

Usage

```
generate_mexican_hat_function(dimensions)
```

Arguments

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

Details

$$f(x) = -(1 - x'x) * \exp\left(-\frac{x'x}{2}\right)$$

Note that we have flipped the sign of the function so that it is a minimization problem like all other SOO functions.

Value

A `soo_function`.

---

generate_nonsmooth_generalized_brown_2_function
Nonsmooth generalized Brown function 2 function generator.

Description

The generalized Brown function 2 is defined as

Usage

```
generate_nonsmooth_generalized_brown_2_function(dimensions)
```

Arguments

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

Details

$$f(x) = \sum_{i=1}^{n-1} \left(|x_i|^{x_{i+1}^2 + 1} + |x_{i+1}|^{x_i^2 + 1} \right)$$

Value

A soo_function.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

---

generate_number_of_active_faces_function
Number of active faces test function generator.

Description

The number of active faces function test function is defined as

Usage

```
generate_number_of_active_faces_function(dimensions)
```

Arguments

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

Details

$$f(x) = \max_{1 \leq i \leq n} \left\{ \ln \left(\left| \sum_{i=1}^{n} x_i \right| + 1 \right), \ln(|x_i| + 1) \right\}$$

Value

A `soo_function`.

References

Haarala, M. and Miettinen, K. and Maekelae, M. M., New limited memory bundle method for large-scale nonsmooth optimization.

`generate_rastrigin_function`

Rastrigin test function generator.

Description

The definition used is

Usage

```
generate_rastrigin_function(dimensions)
```

Arguments

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

Details

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

Value

A `soo_function`.

References

L. A. Rastrigin. Extremal control systems. Theoretical Foundations of Engineering Cybernetics Series. (in Russian), Nauka, Moscow, 1974.

generate_rosenbrock_function
Rosenbrock test function generator.

Description

The definition used is

Usage

```
generate_rosenbrock_function(dimensions)
```

Arguments

dimensions	[integer(1)] Size of parameter space. Must be greater than 1.

Details

$$f(x) = \sum_{i=1}^{n-1} \left(100 \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right)$$

Value

A soo_function.

References

H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. The Computer Journal, 3(3):175-184, 1960.

generate_sphere_function
Sphere test function generator.

Description

The sphere function is arguably the simplest test function. It is defined as

Usage

```
generate_sphere_function(dimensions)
```

Arguments

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

Details

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

Value

A soo_function.

References

K. D. De Jong. An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

---

generate_weierstrass_function

Generator for the Weierstrass function.

Description

The definition used is

Usage

```
generate_weierstrass_function(dimensions)
```

Arguments

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

Details

$$f(x) = \sum_{i=1}^{n} \left(\sum_{k=0}^{20} \left(0.5^k \cos\left(2\pi 3^k \left(x_i + 0.5\right)\right)\right] \right) - n \sum_{k=0}^{20} \left[a^k \cos\left(\pi b^k\right)\right)$$

Value

A soo_function.

Examples

```
f <- generate_weierstrass_function(2)
plot(f, rank=TRUE)
```

global_minimum — *Global Optimum*

Description

Retrieve the global minimum of a test function.

Usage

```
global_minimum(fn)
```

Arguments

fn	Function to query.

Value

List with two elements. `par` contains the location of the global minimum in the parameter space (possibly as a list if there are multiple global minima) and `value` the function value of the global minimum.

inner_function — *Retrieve the inner function contained in a wrapping function.*

Description

Retrieve the inner function contained in a wrapping function.

Usage

```
inner_function(fn)
```

Arguments

fn	`[function]` Function object

Details

This function is a utility function that is used internally by functions that wrap `soo_function` objects and want to return “wrapped” versions of these functions. Such a “wrapped” function should still provide all the methods that are applicable to the “inner” function. For this to work in a generic fashion, we need a method to retrieve the inner function and continue method dispatch on it. This generic implements that interface.

Value

The inner function of `fn`.

is_counting_function *Counting function*

Description

Check if a function or one of its wrapped functions is a ‘counting_function’.

Usage

```
is_counting_function(fn)
```

Arguments

fn	[function] Function to check.

is_recording_function *Recording function*

Description

Check if a function supports the ‘recording_function’ interface.

Usage

```
is_recording_function(fn)
```

Arguments

fn	Function to check.

Value

TRUE if fn is a ‘recording_function’ else FALSE.

is_soo_function — *Single Objective Optimization Function*

Description

Check if a function is a SOO function.

Usage

```
is_soo_function(fn)
```

Arguments

fn	Function to check.

Value

TRUE if fn is a proper SOO function, else FALSE.

is_soo_function_generator — *SOO function generator*

Description

Returns whether fn is a generator for a single objective optimization function.

Usage

```
is_soo_function_generator(fn)
```

Arguments

fn	Function.

Value

TRUE if fn is a function generator and FALSE otherwise.

lower_bounds — *Retrieve the lower or upper bounds of a test function.*

Description

Retrieve the lower or upper bounds of a test function.

Usage

```
lower_bounds(fn)

upper_bounds(fn)
```

Arguments

fn	[soo_function] Object of type soo_function to query.

Value

numeric Vector of lower or upper bounds of test function.

number_of_evaluations — *Retrieve evaluation counter*

Description

Return the number of times a test function has been evaluated.

Usage

```
number_of_evaluations(fn)
```

Arguments

fn	[counting_function] A counting function as returned by counting_function.

Details

The test function must be wrapped by counting_function for this to work.

Value

The current value of the evaluation counter.

number_of_parameters *Number of parameters*

Description

Return the parameter space size of a function.

Usage

```
number_of_parameters(fn)
```

Arguments

fn	[soo_function] Function.

Value

Expected length of first argument. I.e. the size of the parameter space of the function fn.

plot.soo_function *Plot a test function in 1 or 2 dimensions.*

Description

Plot a test function in 1 or 2 dimensions.

Usage

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

Arguments

x	[soo_function] Function to plot.
...	Passed to the respective plot function.

Author(s)

Olaf Mersmann <olafm@p-value.net>

Examples

```
par(mfrow=c(2, 2))
fn <- generate_sphere_function(2)
plot(fn)
plot(fn, show="contour")
plot(fn, rank=TRUE)
plot(fn, log=TRUE)
```

---

plot3d *Plot a test function in 3D.*

Description

Plot a test function in 3D.

Usage

```
plot3d(
  x,
  lower = lower_bounds(x),
  upper = upper_bounds(x),
  n = 10000L,
  main = function_name(x),
  xlab = expression(x[1]),
  ylab = expression(x[2]),
  log = FALSE,
  rank = FALSE,
  ...
)
```

Arguments

x	Object of type soo_function to plot.
lower	Lower bounds of x1 and x2.
upper	Upper bounds of x1 and x2.
n	Number of locations at which to sample the function.
main	Main title of plot.
xlab	Label of x (x1) axes.
ylab	Label of y (x2) axes.
log	If TRUE, the z axes is plotted on log scale.
rank	If TRUE, instead of the y values, their ranks are drawn.
...	Passed to persp3d.default. par(mfrow=c(1, 3)) fn <- generate_sphere_function(2) plot3d(fn) plot3d(fn, log=TRUE) plot3d(fn, rank=TRUE)

Author(s)

Olaf Mersmann <olafm@datensplitter.net>

`plot_1d_soo_function` *Plot a test function in 1D.*

Description

Plot a test function in 1D.

Usage

```
plot_1d_soo_function(
  fn,
  lower = lower_bounds(fn),
  upper = upper_bounds(fn),
  n = 1001L,
  xlab,
  ylab,
  main = function_name(fn),
  log = FALSE,
  rank = FALSE,
  ...
)
```

Arguments

`fn`	[`soo_function`] Function to plot.
`lower`	[`numeric`] Lower bounds of x1 and x2.
`upper`	[`numeric`] Upper bounds of x1 and x2.
`n`	[`integer(1)`] Number of locations at which to sample the function.
`xlab`	[`character(1)`] Label of x (x1) axes.
`ylab`	[`character(1)`] Label of y (x2) axes.
`main`	[`character(1)`] Main title of plot.
`log`	[`boolean(1)`] If `TRUE`, the z axes is plotted on log scale.
`rank`	[`boolean(1)`] If `TRUE`, instead of the y values, their ranks are drawn.
`...`	Ignored.

Author(s)

Olaf Mersmann <olafm@p-value.net>

plot_2d_soo_function *Plot a test function in 2D.*

Description

Plot a test function in 2D.

Usage

```
plot_2d_soo_function(
  fn,
  lower = lower_bounds(fn),
  upper = upper_bounds(fn),
  n = 10000L,
  main = function_name(fn),
  xlab = expression(x[1]),
  ylab = expression(x[2]),
  log = FALSE,
  rank = FALSE,
  asp = 1,
  show = c("image", "contour"),
  image_args = list(useRaster = TRUE),
  contour_args = list(),
  ...
)
```

Arguments

fn	[soo_function] Function to plot.
lower	[numeric] Lower bounds of x1 and x2.
upper	[numeric] Upper bounds of x1 and x2.
n	[integer(1)] Number of locations at which to sample the function.
main	[character(1)] Main title of plot.
xlab	[character(1)] Label of x (x1) axes.
ylab	[character(1)] Label of y (x2) axes.
log	[boolean(1)] If TRUE, the z axes is plotted on log scale.
rank	[boolean(1)] If TRUE, instead of the y values, their ranks are drawn.
asp	[numeric(1)] Aspect ratio of plot. Defaults to 1.
show	[character] A vector of parts to plot. Defaults to c("image","contour") and can be any subset.
image_args	[list] List of further arguments passed to image().
contour_args	[list] List of further arguments passed to contour().
...	Ignored.

Author(s)

Olaf Mersmann <olafm@p-value.net>

Examples

```
par(mfrow=c(2, 2))
fn <- generate_sphere_function(2)
plot(fn)
plot(fn, show="contour")
plot(fn, rank=TRUE)
plot(fn, log=TRUE)
```

---

print.soo_function *Print a SOO function.*

Description

Print a SOO function.

Usage

```
## S3 method for class 'soo_function'
print(x, ...)
```

Arguments

x	[soo_function] A soo_function object.
...	Ignored.

---

random_parameters *Random parameter generation*

Description

Generate random parameter(s) for a given function.

Usage

```
random_parameters(n, fn)

random_parameter(fn)
```

Arguments

n	[integer(1)] Number of parameters to generate.
fn	[soo_function] Test function.

Details

Given a test function fn, generate n random parameter settings for that function.

Value

For random_parameters, a matrix containing the parameter settings in the *columns* of the matrix. random_parameter returns a numeric vector with a single parameter setting for the given function.

Examples

```
fn <- generate_ackley_function(10)
X <- random_parameters(100, fn)
str(X)
y <- fn(X)
```

---

random_rotation_matrix

Generate a random d*-dimensional rotation matrix.*

Description

The algorithm used to randomly create the rotation matrix is due to R Salomon (see reference). No guarantee is given that the generated rotation matrices are uniformly distributed in any sense.

Usage

```
random_rotation_matrix(d)
```

Arguments

d	Dimension of desired rotation matrix.

Value

A random $d \times d$ rotation matrix.

References

Salomon R. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. Biosystems. 1996;39(3):263-78.

recorded_values *Return the internally recorded parameter settings and function values for a test function.*

Description

Return the internally recorded parameter settings and function values for a test function.

Usage

```
recorded_values(fn)
```

Arguments

fn	A function with recorded parameter settings and function values that should be returned.

Value

A list with elements `time`, `par` and `value` which are the point in time (in number of function evaluations), the parameter setting and the function value recorded. The parameter settings are stored in a matrix where each column is one parameter settings.

recording_function *Recording functions*

Description

Return a new function which is identical to the `soofunction` passed in except that evaluated parameter settings and function values are recorded.

Usage

```
recording_function(fn, record_pars = TRUE, record_values = TRUE, predicate)
```

Arguments

fn	A test function (class `soo_function` or `wrapped_soo_function`).
record_pars	`[boolean(1)]` If `TRUE`, parameter values (x) will be recorded.
record_values	`[boolean(1)]` If `TRUE`, function values (y) will be recorded.
predicate	`[function(par,value,time)]` Predicate function that returns `TRUE` if recording should take place for the given parameter setting `par`, function value `value` and evaluation `time`. Note that `time` is measured in function evaluations.

See Also

recorded_values to retrieve the recorded parameter and function values.

Examples

```
fn <- recording_function(generate_sphere_function(2))
X <- random_parameters(10, fn)
y <- fn(X)
rv <- recorded_values(fn)
all.equal(rv$par, X)
all.equal(rv$value, y)

## With a predicate
pv <- function(par, value, time)
  time %% 3 == 0
fn <- recording_function(generate_sphere_function(2), predicate=pv)
X <- random_parameters(10, fn)
y <- fn(X)
rv <- recorded_values(fn)
all(rv$time %% 3 == 0)
all.equal(rv$par, X[, rv$time])
all.equal(rv$value, y[rv$time])
```

---

reset_evaluation_counter

Reset evaluation counter

Description

Reset the evaluation counter of a test function.

Usage

```
reset_evaluation_counter(fn)
```

Arguments

fn	[counting_function] A counting function as returned by counting_function.

Details

The test function must be wrapped by counting_function for this to work.

Value

The current value of the evaluation counter.

rotate_parameter_space *Rotate the parameter space of a SOO function.*

Description

This function is a simple parameter space transformation. Given a function $f(x)$ it returns a new function $f_r(x) = f(Rx)$, where R is a random rotation matrix.

Usage

```
rotate_parameter_space(fn)
```

Arguments

fn	A `soo_function` object.

Details

If you want repeatable results, make sure you explicitly set a seed before calling `rotate_parameter_space`.

Value

A new `soo_function` object where the parameter space has been randomly rotated.

Examples

```
f <- generate_ackley_function(2)
f_r <- rotate_parameter_space(f)
par(mfrow=c(1, 2))
plot(f)
plot(f_r)
```

soo_function *Single Objective Optimization Function*

Description

Define a new `soo_function` object.

Usage

```
soo_function(
  name,
  id,
  fun,
  dimensions,
  lower_bounds,
  upper_bounds,
  best_value,
  best_par
)
```

Arguments

name	Name of function.
id	Short id for the function. Must be unique to the function instance and should not contain any other characters than [a-z], [0-9] and '-'.
fun	Function definition.
dimensions	Size of parameter space.
lower_bounds	Lower bounds of the parameter space.
upper_bounds	Upper bounds of the parameter space.
best_value	Best known function value.
best_par	Parameter settings that correspond to best_value. If there are multiple global minima, this should be a list with one entry for each minimum.

Value

A soo_function object.

Examples

```
## Given the following simple benchmark function:
f_my_sphere <- function(x)
  sum((x-1)*(x-1))

## we can define a corresponding 2d soo_function:
f <- soo_function("My Sphere", "my-sphere-2d", f_my_sphere, 2,
                  c(-10, -10), c(10, 10),
                  0, c(1, 1))

## And then plot it:
plot(f)
```

with_fixed_budget — *Fixed budget expression evaluation Evaluate `expr` with a fixed budget for the number of times any test function in `expr` may be evaluated.*

Description

Fixed budget expression evaluation

Evaluate `expr` with a fixed budget for the number of times any test function in `expr` may be evaluated.

Usage

```
with_fixed_budget(expr, budget)
```

Arguments

`expr`	`[expression]` Expression to evaluate
`budget`	`[integer(1)]` Maximum number of test function evaluations that may be performed by `expr`.

Details

The main use of this function is in benchmarking (optimization) algorithms. It ensures that the algorithm does not perform more than `budget` function evaluations by tracking the number of evaluations performed and raising a `condition` if the budget is reached. For this to work, the function must find one and only one `soofunction` object in `expr` which will be replaced by a modified test function that performs the tracking and signaling.

While elegant from a users perspective, this function is not fool proof. It is possible to construct situations were it will fail. For example, if the employed optimization algorithm is written in C and does not use the memory allocation routines provided by R, then this will certainly lead to memory leaks. You have been warned.

Value

A `list` with elements 'par', 'value' and 'counts' with contents that are identical to the return value of `optim`

Examples

```
fn <- generate_sphere_function(10)
res <- with_fixed_budget(optim(random_parameter(fn), fn), 25)
print(res)
```

Index