Package ‘EmiR’

April 20, 2022

Type Package

Title Evolutionary Minimizer for R

Version 1.0.3

Author Davide Pagano [aut],
Lorenzo Sostero [aut, cre]

Maintainer Lorenzo Sostero <l.sostero@studenti.unibs.it>

Description A C++ implementation of the following evolutionary algorithms: Bat Algorithm (Yang, 2010 <doi:10.1007/978-3-642-12538-6_6>), Cuckoo Search (Yang, 2009 <doi:10.1109/nabic.2009.5393690>), Genetic Algorithms (Holland, 1992, ISBN:978-0262581110), Gravitational Search Algorithm (Rashedi et al., 2009 <doi:10.1016/j.ins.2009.03.004>), Grey Wolf Optimization (Mirjalili et al., 2014 <doi:10.1016/j.advengsoft.2013.12.007>), Harmony Search (Geem et al., 2001 <doi:10.1177/003754970107600201>), Improved Harmony Search (Mahdavi et al., 2007 <doi:10.1016/j.amc.2006.11.033>), Moth-flame Optimization (Mirjalili, 2015 <doi:10.1016/j.knosys.2015.07.006>), Particle Swarm Optimization (Kennedy et al., 2001 ISBN:1558605959), Simulated Annealing (Kirkpatrick et al., 1983 <doi:10.1126/science.220.4598.671>), Whale Optimization Algorithm (Mirjalili and Lewis, 2016 <doi:10.1016/j.advengsoft.2016.01.008>). 'EmiR' can be used not only for unconstrained optimization problems, but also in presence of inequality constrains, and variables restricted to be integers.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports Rcpp (>= 1.0.5), methods, Rdpack, tictoc, ggplot2, tibble, tidyr, dplyr, gganimate, mathjaxr, data.table, plot3D, graphics

LinkingTo Rcpp, RcppProgress, testthat

RoxygenNote 7.1.2

Roxygen list(markdown = TRUE)

RdMacros Rdpack, mathjaxr

SystemRequirements C++11

Suggests xml2, testthat

R topics documented:

ackley_func — *Ackley Function*

Description

Implementation of n-dimensional Ackley function, with $a = 20$, $b = 0.2$ and $c = 2\pi$ (see definition below).

Usage

```
ackley_func(x)
```

Arguments

`x`	numeric or complex vector.

Details

On an n-dimensional domain it is defined by

$$f(\vec{x}) = -a \exp\left(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + \exp(1),$$

and is usually evaluated on $x_i \in [-32.768, 32.768]$, for all $i = 1, ..., n$. The function has one global minimum at $f(\vec{x}) = 0$ for $x_i = 0$ for all $i = 1, ..., n$.

Value

The value of the function.

References

Ackley DH (1987). *A Connectionist Machine for Genetic Hillclimbing*. Springer US. doi: 10.1007/9781461319979.

animate_population — *Animation of population motion*

Description

Create an animation of the population motion for the minimization of 1D and 2D functions. The animation can be produced only if `save_pop_history` is `TRUE` in the options of the minimizer (see MinimizerOpts).

Usage

```
animate_population(minimizer_result, n_points = 100)
```

Arguments

minimizer_result
: an object of class `OptimizationResults` (see OptimizationResults).

n_points
: number of points per dimension used to draw the objective function. Default is `100`.

---

bohachevsky_func *Bohachevsky Function*

Description

Implementation of 2-dimensional Bohachevsky function.

Usage

```
bohachevsky_func(x)
```

Arguments

x
: numeric or complex vector.

Details

On an 2-dimensional domain it is defined by

$$f(\vec{x}) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

and is usually evaluated on $x_i \in [-100, 100]$, for all $i = 1, 2$. The function has one global minimum at $f(\vec{x}) = 0$ for $\vec{x} = [0, 0]$.

Value

The value of the function.

References

Bohachevsky IO, Johnson ME, Stein ML (1986). "Generalized simulated annealing for function optimization." *Technometrics*, **28**(3), 209–217.

colville_func *Colville Function*

Description

Implementation of 4-dimensional Colville function.

Usage

```
colville_func(x)
```

Arguments

x	numeric or complex vector.

Details

On an 4-dimensional domain it is defined by

$$f(\vec{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1),$$

and is usually evaluated on $x_i \in [-10, 10]$, for all $i = 1, ..., 4$. The function has one global minimum at $f(\vec{x}) = 0$ for $\vec{x} = [1, 1, 1, 1]$.

Value

The value of the function.

References

Grippo L, Lampariello F, Lucidi S (1989). "A truncated Newton method with nonmonotone line search for unconstrained optimization." *Journal of Optimization Theory and Applications*, **60**(3), 401–419. doi: 10.1007/bf00940345.

config_abc *Configuration object for the Artificial Bee Colony Algorithm*

Description

Create a configuration object for the Artificial Bee Colony Algorithm (ABC). At minimum the number of iterations (parameter `iterations`) and the number of bees (parameter `population_size`) have to be provided.

Usage

```
config_abc(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  employed_frac = 0.5,
  n_scout = 1
)
```

Arguments

`iterations`	maximum number of iterations.
`population_size`	number of bees.
`iterations_same_cost`	maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.
`absolute_tol`	absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.
`employed_frac`	fraction employed bees. Default is `0.5`.
`n_scout`	number of scout bees. Default is `1`.

Value

`config_abc` returns an object of class `ABCConfig`.

References

Karaboga D, Basturk B (2007). "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm." *Journal of Global Optimization*, **39**(3), 459–471. doi: 10.1007/s108980079149x.

Examples

```
conf <- config_abc(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL, employed_frac = 0.5, n_scout = 1)
```

`config_algo` *Configuration object for algorithms*

Description

Create a configuration object for one of the algorithms available in EmiR. At minimum the id of the algorithm (parameter `algorithm_id`), the number of iterations (parameter `iterations`) and the number of individuals in the population (parameter `population_size`) have to be provided.

Usage

```
config_algo(
  algorithm_id,
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  ...
)
```

Arguments

`algorithm_id`	id of the algorithm to be used. See list_of_algorithms for the list of the available algorithms.
`iterations`	maximum number of iterations.
`population_size`	number of individuals in the population.
`iterations_same_cost`	maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.
`absolute_tol`	absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.
`...`	algorithm specific parameters (see specific configuration functions for more details).

Value

`config_algo` returns a configuration object specific for the specified algorithm.

Examples

```
conf <- config_algo(algorithm_id = "PS", population_size = 200, iterations = 10000)
```

config_bat *Configuration object for the Bat Algorithm*

Description

Create a configuration object for the Bat Algorithm (BAT). At minimum the number of iterations (parameter `iterations`) and the number of bats (parameter `population_size`) have to be provided.

Usage

```
config_bat(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  initial_loudness = 1.5,
  alpha = 0.9,
  initial_pulse_rate = 0.5,
  gamma = 0.9,
  freq_min = 0,
  freq_max = 2
)
```

Arguments

`iterations` maximum number of iterations.

`population_size` number of bats.

`iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.

`absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.

`initial_loudness` initial loudness of emitted pulses. Typical values are in the range [1, 2]. Default is `1.5`.

`alpha` parameter to control the linearly decreasing loudness with the iterations. It should be between 0 and 1. Default is `0.9`.

`initial_pulse_rate` initial rate at which pulses are emitted. It should be between 0 and 1. Default is `0.5`.

`gamma` parameter to control the exponentially decreasing pulse rate with the iterations. Defatul is `0.9`.

freq_min minimum frequency value of pulses. Default is 0.

freq_max maximum frequency value of pulses. Default is 2.0.

Value

config_bat returns an object of class BATConfig.

References

Yang X (2010). "A new metaheuristic bat-inspired algorithm." In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, 65–74. Springer.

Examples

```
conf <- config_bat(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL, initial_loudness = 1.5, alpha = 0.9,
initial_pulse_rate = 0.5, gamma = 0.9,
freq_min = 0., freq_max = 2.)
```

---

config_cs *Configuration object for the Cuckoo Search Algorithm*

Description

Create a configuration object for the Cuckoo Search Algorithm (CS). At minimum the number of iterations (parameter iterations) and the number of host nests (parameter population_size) have to be provided.

Usage

```
config_cs(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  discovery_rate = 0.25,
  step_size = 1
)
```

Arguments

iterations maximum number of iterations.

population_size number of host nests.

iterations_same_cost
: maximum number of consecutive iterations with the *same* (see the parameter absolute_tol) best cost before ending the minimization. If NULL the minimization continues for the number of iterations specified by the parameter iterations. Default is NULL.

absolute_tol
: absolute tolerance when comparing best costs from consecutive iterations. If NULL the machine epsilon is used. Default is NULL.

discovery_rate
: probability for the egg laid by a cuckoo to be discovered by the host bird. It should be between 0 and 1. Default is 0.25.

step_size
: step size of the Levy flight. Default is 1.0.

Value

config_cs returns an object of class CSConfig.

References

Yang X, Deb S (2009). "Cuckoo Search via Lèvy flights." In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*. doi: 10.1109/nabic.2009.5393690.

Examples

```
conf <- config_cs(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL, discovery_rate = 0.25, step_size = 1.0)
```

---

config_ga *Configuration object for the Genetic Algorithm*

Description

Create a configuration object for the Genetic Algorithm (GA). At minimum the number of iterations (parameter iterations) and the number of chromosomes (parameter population_size) have to be provided.

Usage

```
config_ga(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  keep_fraction = 0.4,
  mutation_rate = 0.1
)
```

Arguments

- `iterations` maximum number of iterations.
- `population_size` number of chromosomes.
- `iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.
- `absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.
- `keep_fraction` fraction of the population that survives for the next step of mating. Default is `0.4`.
- `mutation_rate` probability of mutation. Default is `0.1`.

Value

`config_ga` returns an object of class `GAConfig`.

References

Holland JH (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA. ISBN 0262082136.

Examples

```
conf <- config_ga(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL,keep_fraction = 0.4, mutation_rate = 0.1)
```

`config_gsa` *Configuration object for the Gravitational Search Algorithm*

Description

Create a configuration object for the Gravitational Search Algorithm (GSA). At minimum the number of iterations (parameter `iterations`) and the number of planets (parameter `population_size`) have to be provided.

Usage

```
config_gsa(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  grav = 1000,
  grav_evolution = 20
)
```

Arguments

- `iterations` maximum number of iterations.
- `population_size` number of planets.
- `iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.
- `absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.
- `grav` gravitational constant, involved in the acceleration of planets. Default is `100`.
- `grav_evolution` parameter to control the exponentially decreasing gravitational constant with the iterations. Default is `20.0`.

Value

`config_gsa` returns an object of class `GSAConfig`.

References

Rashedi E, Nezamabadi-pour H, Saryazdi S (2009). "GSA: A Gravitational Search Algorithm." *Information Sciences*, **179**(13), 2232–2248. doi: 10.1016/j.ins.2009.03.004.

Examples

```
conf <- config_gsa(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL, grav = 1000, grav_evolution = 20.)
```

config_gwo — *Configuration object for the Grey Wolf Optimizer Algorithm*

Description

Create a configuration object for the Grey Wolf Optimizer Algorithm (GWO). At minimum the number of iterations (parameter `iterations`) and the number of wolves (parameter `population_size`) have to be provided.

Usage

```
config_gwo(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL
)
```

Arguments

`iterations` maximum number of iterations.

`population_size` number of wolves.

`iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.

`absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.

Value

`config_gwo` returns an object of class `GWOConfig`.

References

Mirjalili S, Mirjalili SM, Lewis A (2014). "Grey Wolf Optimizer." *Advances in Engineering Software*, **69**, 46–61. doi: 10.1016/j.advengsoft.2013.12.007.

Examples

```
conf <- config_gwo(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL)
```

config_hs *Configuration object for the Harmony Search Algorithm*

Description

Create a configuration object for the Harmony Search Algorithm (HS). At minimum the number of iterations (parameter `iterations`) and the number of solutions in the harmony memory (parameter `population_size`) have to be provided.

Usage

```
config_hs(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  considering_rate = 0.5,
  adjusting_rate = 0.5,
  distance_bandwidth = 0.1
)
```

Arguments

`iterations` maximum number of iterations.

`population_size` number of solutions in the harmony memory.

`iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.

`absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.

`considering_rate` probability for each component of a newly generated solution to be recalled from the harmony memory.

`adjusting_rate` probability of the pitch adjustment in case of a component recalled from the harmony memory.

`distance_bandwidth` amplitude of the random pitch adjustment.

Value

`config_hs` returns an object of class `HSConfig`.

References

Lee KS, Geem ZW (2004). "A new structural optimization method based on the harmony search algorithm." *Computers & Structures*, **82**(9-10), 781–798. doi: 10.1016/j.compstruc.2004.01.002.

Examples

```
conf <- config_hs(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL, considering_rate = 0.5, adjusting_rate = 0.5,
distance_bandwidth = 0.1)
```

`config_ihs` *Configuration object for the Improved Harmony Search Algorithm*

Description

Create a configuration object for the Improved Harmony Search Algorithm (IHS). At minimum the number of iterations (parameter `iterations`) and the number of solutions in the harmony memory (parameter `population_size`) have to be provided.

Usage

```
config_ihs(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  considering_rate = 0.5,
  min_adjusting_rate = 0.3,
  max_adjusting_rate = 0.99,
  min_distance_bandwidth = 1e-04,
  max_distance_bandwidth = 1
)
```

Arguments

`iterations` maximum number of iterations.

`population_size` number of solutions in the harmony memory.

`iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.

`absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.

`considering_rate`
: probability for each component of a newly generated solution to be recalled from the harmony memory.

`min_adjusting_rate`
: minimum value of the pitch adjustment probability.

`max_adjusting_rate`
: maximum value of the pitch adjustment probability.

`min_distance_bandwidth`
: minimum amplitude of the random pitch adjustment.

`max_distance_bandwidth`
: maximum amplitude of the random pitch adjustment.

Value

`config_ihs` returns an object of class `IHSConfig`.

References

Mahdavi M, Fesanghary M, Damangir E (2007). "An improved harmony search algorithm for solving optimization problems." *Applied Mathematics and Computation*, **188**(2), 1567–1579. doi: 10.1016/j.amc.2006.11.033.

Examples

```
conf <- config_ihs(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL,considering_rate = 0.5, min_adjusting_rate = 0.3,
max_adjusting_rate = 0.99, min_distance_bandwidth = 0.0001, max_distance_bandwidth = 1)
```

`config_mfo` *Configuration object for the Moth-flame Optimization Algorithm*

Description

Create a configuration object for the Moth-flame Optimization Algorithm (MFO). At minimum the number of iterations (parameter `iterations`) and the number of moths (parameter `population_size`) have to be provided.

Usage

```
config_mfo(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL
)
```

Arguments

- `iterations` — maximum number of iterations.
- `population_size` — number of moths.
- `iterations_same_cost` — maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.
- `absolute_tol` — absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.

Value

`config_mfo` returns an object of class `MFOConfig`.

References

Mirjalili S (2015). "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm." *Knowledge-Based Systems*, **89**, 228–249. doi: 10.1016/j.knosys.2015.07.006.

Examples

```
conf <- config_mfo(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL)
```

---

config_ps — *Configuration object for the Particle Swarm Algorithm*

Description

Create a configuration object for the Particle Swarm Algorithm (PS). At minimum the number of iterations (parameter `iterations`) and the number of particles (parameter `population_size`) have to be provided.

Usage

```
config_ps(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  alpha_vel = 0.5,
  alpha_evolution = 1,
  cognitive = 2,
  social = 2,
  inertia = 0.9
)
```

Arguments

`iterations` maximum number of iterations.

`population_size` number of particles.

`iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.

`absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.

`alpha_vel` maximum velocity of particles, defined as a fraction of the range on each parameter. Default is `0.5`.

`alpha_evolution` parameter to control the decreasing alpha_vel value with the iterations. Default is `1.0` (linear).

`cognitive` parameter influencing the motion of the particle on the basis of distance between its current and best positions. Default is `2.0`.

`social` parameter influencing the motion of the particle on the basis of distance between its current position and the best position in the swarm. Default is `2.0`.

`inertia` parameter influencing the dependency of the velocity on its value at the previous iteration. Default `0.9`.

Value

`config_ps` returns an object of class `PSConfig`.

References

Eberhart R, Kennedy J (1995). "A new optimizer using particle swarm theory." In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 39–43. Ieee.

Examples

```
conf <- config_ps(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL,alpha_vel = 0.5, alpha_evolution = 1.0, cognitive = 2.0,
social = 2.0, inertia = 0.9)
```

config_sa — *Configuration object for the Simulated Annealing Algorithm*

Description

Create a configuration object for the Simulated Annealing algorithm (SA). At minimum the number of iterations (parameter `iterations`) and the number of particles (parameter `population_size`) have to be provided.

Usage

```
config_sa(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL,
  T0 = 50,
  Ns = 3,
  Nt = 3,
  c_step = 2,
  Rt = 0.85,
  Wmin = 0.25,
  Wmax = 1.25
)
```

Arguments

`iterations` maximum number of iterations.

`population_size` number of particles.

`iterations_same_cost` maximum number of consecutive iterations with the *same* (see the parameter `absolute_tol`) best cost before ending the minimization. If `NULL` the minimization continues for the number of iterations specified by the parameter `iterations`. Default is `NULL`.

`absolute_tol` absolute tolerance when comparing best costs from consecutive iterations. If `NULL` the machine epsilon is used. Default is `NULL`.

`T0` initial temperature. Default is `50`.

`Ns` number of iterations before changing velocity. Default is `3`.

`Nt` number of iterations before changing the temperature. Default is `3`.

`c_step` parameter involved in the velocity update. Default is `2`.

`Rt` scaling factor for the temperature. Default is `0.85`.

`Wmin` parameter involved in the generation of the starting point. Default is `0.25`.

`Wmax` parameter involved in the generation of the starting point. Default is `1.25`.

Value

config_sa returns an object of class SAConfig.

References

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). "Optimization by Simulated Annealing." *Science*, **220**(4598), 671–680. doi: 10.1126/science.220.4598.671.

Examples

```
conf <- config_sa(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL, T0 = 50., Ns = 3., Nt = 3., c_step = 2., Rt = 0.85, Wmin = 0.25,
 Wmax = 1.25)
```

---

config_woa — *Configuration object for the Whale Optimization Algorithm*

Description

Create a configuration object for the Whale Optimization Algorithm (WOA). At minimum the number of iterations (parameter iterations) and the number of whales (parameter population_size) have to be provided.

Usage

```
config_woa(
  iterations,
  population_size,
  iterations_same_cost = NULL,
  absolute_tol = NULL
)
```

Arguments

- iterations — maximum number of iterations.
- population_size — number of whales.
- iterations_same_cost — maximum number of consecutive iterations with the *same* (see the parameter absolute_tol) best cost before ending the minimization. If NULL the minimization continues for the number of iterations specified by the parameter iterations. Default is NULL.
- absolute_tol — absolute tolerance when comparing best costs from consecutive iterations. If NULL the machine epsilon is used. Default is NULL.

Value

`config_woa` returns an object of class `WOAConfig`.

References

Mirjalili S, Lewis A (2016). "The Whale Optimization Algorithm." *Advances in Engineering Software*, **95**, 51-67. ISSN 0965-9978, doi: 10.1016/j.advengsoft.2016.01.008, https://www.sciencedirect.com/science/article/pii/S0965997816300163.

Examples

```
conf <- config_woa(iterations = 100, population_size = 50, iterations_same_cost = NULL,
absolute_tol = NULL)
```

`constrained_function` *Constrained function for minimization*

Description

Create a constrained function for minimization.

Usage

```
constrained_function(func, ...)
```

Arguments

`func`	original objective function.
`...`	one or more constraints of class `Constraint`. See constraint.

Value

`constrained_function` returns an object of class `ConstrainedFunction`.

constraint *Constraint for minimization*

Description

Create a constraint function for constrained optimization. Only inequality constraints are supported.

Usage

```
constraint(func, inequality)
```

Arguments

`func`	function describing the constraint.
`inequality`	inequality type. Possible values: >, >=, <=, <.

Value

`constraint` returns an object of class `Constraint`.

Examples

```
g1 <- function(x) 0.0193*x[3] - (x[1]*0.0625)
c1 <- constraint(g1, "<=")
```

freudenstein_roth_func *Freudenstein Roth Function*

Description

Implementation of 2-dimensional Freudenstein Roth function.

Usage

```
freudenstein_roth_func(x)
```

Arguments

`x`	numeric or complex vector.

Details

On an 2-dimensional domain it is defined by

$$f(\vec{x}) = (x_1 - 13 + ((5 - x_2)x_2 - 2)x_2)^2 + (x_1 - 29 + ((x_2 + 1)x_2 - 14)x_2)^2$$

and is usually evaluated on $x_i \in [-10, 10]$, for all $i = 1, 2$. The function has one global minimum at $f(\vec{x}) = 0$ for $\vec{x} = [5, 4]$.

Value

The value of the function.

References

Rao S (2019). *Engineering optimization : theory and practice*. John Wiley \& Sons, Ltd, Hoboken, NJ, USA. ISBN 978-1-119-45479-3.

`G01InitPop` *Data set for example G01*

Description

This data set contains the initial positions for a population of size 20 to be used with the example G01.

`get_population` *Get population positions*

Description

Return a `data.frame` with the position of all individuals in the population at the specified iteration, from an object of class `OptimizationResults` produced with the option `save_pop_history` set to `TRUE` (see MinimizerOpts).

Usage

```
get_population(minimizer_result, iteration)
```

Arguments

`minimizer_result`
: an object of class `OptimizationResults` (see OptimizationResults).

`iteration`
: iteration number.

Value

An object of class `data.frame`.

griewank_func *Griewank Function*

Description

Implementation of n-dimensional Griewank function.

Usage

```
griewank_func(x)
```

Arguments

`x`	numeric or complex vector.

Details

On an n-dimensional domain it is defined by

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

and is usually evaluated on $x_i \in [-600, 600]$, for all $i = 1, ..., n$. The function has global minima at $f(\vec{x}) = 0$ for $x_i = 0$ for all $i = 1, ..., n$.

Value

The value of the function.

References

Griewank AO (1981). "Generalized descent for global optimization." *Journal of optimization theory and applications*, **34**(1), 11–39.

list_of_algorithms *Return the list of algorithms in EmiR*

Description

Return a `data.frame` with the ID, description and configuration function name of all the algorithms implemented in EmiR.

Usage

```
list_of_algorithms()
```

Value

An object of class `data.frame`.

`list_of_functions` *Return the list of pre-defined functions in EmiR*

Description

Return a `data.frame` with function name, full name, and minimum and maximum number of parameters accepted for all the pre-defined functions in EmiR.

Usage

```
list_of_functions()
```

Value

An object of class `data.frame`.

`miele_cantrell_func` *Miele Cantrell Function*

Description

Implementation of 4-dimensional Miele Cantrell Function.

Usage

```
miele_cantrell_func(x)
```

Arguments

`x`	numeric or complex vector.

Details

On an 4-dimensional domain it is defined by

$$f(\vec{x}) = \left(e^{-x_1} - x_2\right)^4 + 100(x_2 - x_3)^6 + (\tan(x_3 - x_4))^4 + x_1^8$$

and is usually evaluated on $x_i \in [-2, 2]$, for all $i = 1, ..., 4$. The function has one global minimum at $f(\vec{x}) = 0$ for $\vec{x} = [0, 1, 1, 1]$.

Value

The value of the function.

References

Cragg EE, Levy AV (1969). “Study on a supermemory gradient method for the minimization of functions.” *Journal of Optimization Theory and Applications*, **4**(3), 191–205.

`minimize` *Minimize an Objective Function*

Description

Minimize (or maximize) an objective function, possibly subjected to inequality constraints, using any of the algorithms available in EmiR.

Usage

```
minimize(algorithm_id, obj_func, parameters, config, constraints = NULL, ...)
```

Arguments

`algorithm_id` id of the algorithm to be used. See list_of_algorithms for the list of the available algorithms.

`obj_func` objective function be minimized/maximized.

`parameters` list of parameters composing the search space for the objective function. Parameters are requested to be objects of class `Parameter` (see parameter).

`config` an object with the configuration parameters of the chosen algorithm. For each algorithm there is different function for the tuning of its configuration parameter, as reported in the following list:

- config_abc – configuration function for the *Artificial Bee Colony Algorithm*.
- config_bat – configuration function for the *Bat Algorithm*.
- config_cs – configuration function for the *Cuckoo Search Algorithm*.
- config_ga – configuration function for the *Genetic Algorithm*.
- config_gsa – configuration function for the *Gravitational Search Algorithm*.
- config_gwo – configuration function for the *Grey Wolf Optimizer Algorithm*.
- config_hs – configuration function for the *Harmony Search Algorithm*.
- config_ihs – configuration function for the *Improved Harmony Search Algorithm*.
- config_mfo – configuration function for the *Moth-flame Optimization Algorithm*.
- config_ps – configuration function for the *Particle Swarm Algorithm*.
- config_sa – configuration function for the *Simulated Annealing algorithm*.
- config_woa – configuration function for the *Whale Optimization Algorithm*.

`constraints` list of constraints. Constraints are requested to be objects of class `Constraint` (see constraint).

`...` additional options (see MinimizerOpts).

Value

`minimize` returns an object of class `OptimizationResults` (see OptimizationResults).

Examples

```
## Not run:
 results <- minimize(algorithm_id = "BAT", obj_func = ob, config = conf,
 parameters = list(p1,p2, p3, p4), constraints = list(c1,c2,c3),
    save_pop_history = TRUE, constrained_method = "BARRIER",
    constr_init_pop = TRUE, oob_solutions = "RBC", seed = 1)

## End(Not run)
```

`MinimizerOpts` *EmiR optimization options*

Description

A S4 class storing the options for the optimization algorithms in EmiR.

Slots

`maximize` if `TRUE` the objective function is maximized instead of being minimized. Default is `FALSE`.

`silent_mode` if `TRUE` no output to console is generated. Default is `FALSE`.

`save_pop_history` if `TRUE` the position of all individuals in the population at each iteration is stored. This is necessary for functions like `plot_population` and `animate_population` to work. Default is `FALSE`.

`constrained_method` method for constrained optimization. Possible values are:

- `"PENALTY"` - Penalty Method: the constrained problem is converted to an unconstrained one, by adding a *penalty function* to the objective function. The penalty function consists of a *penalty parameter* multiplied by a measure of violation of the constraints. The penalty parameter is multiplied by a scale factor (see `penalty_scale`) at every iteration;
- `"BARRIER"` - Barrier Method: the value of the objective function is set equal to an arbitrary large positive (or negative in case of maximization) number if any of the constraints is violated;
- `"ACCREJ"` - Acceptance-Rejection method: a solution violating any of the constraints is replaced by a randomly generated new one in the feasible region. Default is `"PENALTY"`.

`penalty_scale` scale factor for the *penalty parameter* at each iteration. It should be greater than 1. Default is 10.

`start_penalty_param` initial value of the *penalty parameter*. It should be greater than 0. Default is 2.

max_penalty_param maximum value for the *penalty parameter*. It should be greater than 0. Default is 1.e+10.

constr_init_pop if TRUE the initial population is generated in the *feasible region* only. Default is TRUE.

oob_solutions strategy to treat out-of-bound solutions. Possible values are:

- "RBC" - Reflective Boundary Condition: the solution is placed back inside the search domain at a position which is distanced from the boundary as the out-of-bound excess. Depending on the optimization algorithm, the velocity of the corresponding individual of the population could be also inverted;
- "PBC" - Periodic Boundary Condition: the solution is placed back inside the search domain at a position which is distanced from the *opposite* boundary as the out-of-bound excess;
- "BAB" - Back At Boundary: the solution is placed back at the boundaries for the out-of-bound dimensions;
- "DIS" - Disregard the solution: the solution is replaced by a new one, which is randomly generated in the search space. Default is "DIS".

seed seed for the internal random number generator. Accepted values are strictly positive integers. If NULL a random seed at each execution is used. Default is NULL.

initial_population manually specify the position of the initial population. A $n \times d$ matrix has to be provided, where n is the population size and d is the number of parameters the objective function is minimized with respect to.

OptimizationResults *EmiR optimization results*

Description

A S4 class storing all relevant data from an optimization with EmiR.

Slots

algorithm the name of the algorithm.

iterations the number of iterations.

population_size the number of individuals in the population.

obj_function the minimized/maximized objective function.

constraints the constraints the objective function is subjected to.

best_cost the best value of the objective function found.

best_parameters the parameter values for which the best cost was obtained.

parameter_range the range on the parameters.

pop_history list containing the positions of all individuals in the population at each iteration. The list is filled only if save_pop_history is TRUE in the options of the minimizer (see MinimizerOpts).

cost_history the vector storing the best value of the objective function at each iteration.

exec_time_sec the execution time in seconds.

is_maximization if TRUE the objective function has been maximized insted of being minimized.

parameter *Parameter for minimization*

Description

Create a parameter the objective function is minimized with respect to.

Usage

```
parameter(name, min_val, max_val, integer = FALSE)
```

Arguments

`name`	name of the parameter.
`min_val`	minimum value the parameter is allowed to assume during minimization.
`max_val`	maximum value the parameter is allowed to assume during minimization.
`integer`	if `TRUE` the parameter is constrained to be integer. Default is `FALSE`.

Value

`parameter` returns an object of class `Parameter`.

Examples

```
p1 <- parameter("x1", 18, 32, integer = TRUE)
```

parameters *Set of parameters for minimization*

Description

Create the set of parameters the objective function is minimized with respect to. A $2 \times n$ matrix or a $3 \times n$ matrix, where the first row is for the lower limits, the second one is for the upper limits, and the (optional) third one is to specify if a parameter is constrained to be integer. In case the third row is not provided, all the parameters are treated as continuous. The name of each of the n parameters is automatically generated and it is of the form xi, where $i = 1, ..., n$.

Usage

```
parameters(values)
```

Arguments

`values`	a $2 \times n$ matrix or a $3 \times n$ matrix.

Value

`parameters` returns a list of objects of class `Parameter`.

`plot_history` *Plot minimization history*

Description

Plot the minimization history as a function of the number of iterations.

Usage

```
plot_history(minimizer_result, ...)
```

Arguments

`minimizer_result`	an object of class `OptimizationResults` (see OptimizationResults).
`...`	additional arguments, such as graphical parameters (see plot).

`plot_population` *Plot the population position*

Description

Plot the position of all individuals in the population, at a given iteration, for 1D and 2D functions. The plot can be produced only if `save_pop_history` is `TRUE` in the options of the minimizer (see MinimizerOpts).

Usage

```
plot_population(minimizer_result, iteration, n_points = 100)
```

Arguments

`minimizer_result`	an object of class `OptimizationResults` (see OptimizationResults).
`iteration`	iteration at which the population is plotted.
`n_points`	number of points per dimention used to draw the objective function. Default is `100`.

rastrigin_func *Rastrigin Function*

Description

Implementation of n-dimensional Rastrigin function.

Usage

```
rastrigin_func(x)
```

Arguments

x	numeric or complex vector.

Details

On an n-dimensional domain it is defined by:

$$f(\vec{x}) = 20n + \sum_{i=1}^{n} \left(x_i^2 - 20\cos(2\pi x_i) \right),$$

and is usually evaluated on $x_i \in [-5.12, 5.12]$, for all $i = 1, ..., n$. The function has one global minimum at $f(\vec{x}) = 0$ for $x_i = 0$ for all $i = 1, ..., n$.

Value

The value of the function.

References

Rastrigin LA (1974). "Systems of extremal control." *Nauka*.

rosenbrock_func *Rosenbrock Function*

Description

Implementation of n-dimensional Rosenbrock function, with $n \geq 2$.

Usage

```
rosenbrock_func(x)
```

Arguments

x	numeric or complex vector.

Details

On an n-dimensional domain it is defined by

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

and is usually evaluated on $x_i \in [-5, 10]$, for all $i = 1, ..., n$. The function has one global minimum at $f(\vec{x}) = 0$ for $x_i = 1$ for all $i = 1, ..., n$.

Value

The value of the function.

References

Rosenbrock HH (1960). "An Automatic Method for Finding the Greatest or Least Value of a Function." *The Computer Journal*, **3**(3), 175–184. doi: 10.1093/comjnl/3.3.175.

---

schwefel_func *Schwefel Function*

Description

Implementation of n-dimensional Schwefel function.

Usage

```
schwefel_func(x)
```

Arguments

`x`	numeric or complex vector.

Details

On an n-dimensional domain it is defined by

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

and is usually evaluated on $x_i \in [-500, 500]$, for all $i = 1, ..., n$. The function has one global minimum at $f(\vec{x}) = -418.9829n$ for $x_i = 420.9687$ for all $i = 1, ..., n$.

Value

The value of the function.

References

Schwefel H (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

styblinski_tang_func *Styblinski-Tang Function*

Description

Implementation of n-dimensional Styblinski-Tang function.

Usage

```
styblinski_tang_func(x)
```

Arguments

x	numeric or complex vector.

Details

On an n-dimensional domain it is defined by

$$f(\vec{x}) = \frac{1}{2}\sum_{i=1}^{n}\left(x_i^4 - 16x_i^2 + 5x_i\right),$$

and is usually evaluated on $x_i \in [-5, 5]$, for all $i = 1, ..., n$. The function has one global minimum at $f(\vec{x}) = -39.16599n$ for $x_i = -2.903534$ for all $i = 1, ..., n$.

Value

The value of the function.

References

Styblinski MA, Tang T (1990). "Experiments in nonconvex optimization: Stochastic approximation with function smoothing and simulated annealing." *Neural Networks*, **3**(4), 467–483. doi: 10.1016/08936080(90)90029k.

Index