

Package ‘evoper’

January 28, 2026

Type Package

Title Evolutionary Parameter Estimation for 'Repast Symphony' Models

Version 0.6.0

Date 2026-01-27

Author Antonio Prestes Garcia [aut, cre],
Alfonso Rodriguez-Paton [aut, ths]

Maintainer Antonio Prestes Garcia <antonio.pgarcia@alumnos.upm.es>

URL <https://github.com/antonio-pgarcia/evoper>

BugReports <https://github.com/antonio-pgarcia/evoper/issues>

Description

The EvoPER, Evolutionary Parameter Estimation for Individual-based Models is an extensible package providing optimization driven parameter estimation methods using metaheuristics and evolutionary computation techniques (Particle Swarm Optimization, Simulated Annealing, Ant Colony Optimization for continuous domains, Tabu Search, Evolutionary Strategies, ...) which could be more efficient and require, in some cases, fewer model evaluations than alternatives relying on experimental design. Currently there are built in support for models developed with 'Repast Symphony' Agent-Based framework (<<https://repast.github.io/>>) and with NetLogo (<<https://www.netlogo.org/>>) which are the most used frameworks for Agent-based modeling.

License MIT + file LICENSE

Depends rrepast

Imports methods, logging, boot, reshape, ggplot2, deSolve, plot3D,
plyr, data.table, utils, RNetLogo

RoxygenNote 7.3.3

Suggests testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2026-01-28 12:40:02 UTC

Contents

| | |
|-------------------------------|----|
| abm.acor | 5 |
| abm.ees1 | 6 |
| abm.ees2 | 6 |
| abm.ga | 7 |
| abm.pso | 8 |
| abm.saa | 9 |
| abm.tabu | 10 |
| acor.archive | 11 |
| acor.F | 12 |
| acor.lthgaussian | 12 |
| acor.N | 13 |
| acor.probabilities | 13 |
| acor.S | 14 |
| acor.sigma | 14 |
| acor.updateants | 15 |
| acor.W | 15 |
| acor.weigth | 16 |
| assert | 16 |
| bestFitness | 17 |
| bestSolution | 17 |
| cbuf | 18 |
| compare.algorithms1 | 18 |
| contourplothelper | 19 |
| ees1.challenge | 20 |
| ees1.explore | 20 |
| ees1.mating | 21 |
| ees1.mating1 | 21 |
| ees1.mutation | 22 |
| ees1.recombination | 22 |
| ees1.selection | 23 |
| elog.debug | 23 |
| elog.error | 23 |
| elog.info | 24 |
| elog.level | 24 |
| enforceBounds | 25 |
| es.evaluate | 25 |
| Estimates-class | 26 |
| extremize | 26 |
| f0.ackley | 27 |
| f0.ackley4 | 27 |
| f0.adtn.rosenbrock2 | 28 |
| f0.bohachevsky | 28 |
| f0.bohachevsky4 | 29 |
| f0.cigar | 29 |
| f0.cigar4 | 30 |
| f0.griewank | 30 |

| | |
|-----------------------------------|----|
| f0.griewank4 | 31 |
| f0.nlnn.rosenbrock2 | 31 |
| f0.periodtuningpp | 32 |
| f0.periodtuningpp12 | 32 |
| f0.periodtuningpp24 | 33 |
| f0.periodtuningpp48 | 34 |
| f0.periodtuningpp72 | 35 |
| f0.rosenbrock2 | 36 |
| f0.rosenbrock4 | 36 |
| f0.rosenbrockn | 37 |
| f0.schaffer | 37 |
| f0.schaffer4 | 38 |
| f0.schwefel | 38 |
| f0.schwefel4 | 39 |
| f0.test | 39 |
| f1.ackley | 40 |
| f1.adtn.rosenbrock2 | 40 |
| f1.bohachevsky | 41 |
| f1.cigar | 41 |
| f1.griewank | 42 |
| f1.nlnn.rosenbrock2 | 42 |
| f1.rosenbrock2 | 43 |
| f1.rosenbrockn | 43 |
| f1.schaffer | 44 |
| f1.schwefel | 44 |
| f1.test | 45 |
| fixdfcolumns | 45 |
| generateSolution | 46 |
| getFitness | 46 |
| GetLogLevel | 47 |
| getSolution | 47 |
| gm.mean | 48 |
| gm.sd | 48 |
| histplotheper | 49 |
| initSolution | 49 |
| lowerBound | 50 |
| Magnitude | 50 |
| naiveperiod | 51 |
| NetLogoFunction-class | 51 |
| NLWrapper.FindJar | 51 |
| NLWrapper.GetParameter | 52 |
| NLWrapper.Model | 52 |
| NLWrapper.Run | 53 |
| NLWrapper.RunExperiment | 54 |
| NLWrapper.SetParameter | 55 |
| NLWrapper.SetRandomSeed | 55 |
| NLWrapper.Shutdown | 56 |
| ObjectiveFunction-class | 56 |

| | |
|--------------------------------|----|
| Options-class | 56 |
| OptionsACOR-class | 57 |
| OptionsEES1-class | 57 |
| OptionsEES2-class | 57 |
| OptionsFactory | 57 |
| OptionsGA-class | 58 |
| OptionsPSO-class | 58 |
| OptionsSAA-class | 58 |
| OptionsTS-class | 58 |
| paramconverter | 59 |
| partSolutionSpace | 59 |
| PlainFunction-class | 60 |
| pop.first | 60 |
| pop.last | 60 |
| predatorprey | 61 |
| predatorprey.plot0 | 61 |
| predatorprey.plot1 | 62 |
| pso.best | 63 |
| pso.chi | 63 |
| pso.lbest | 64 |
| pso.neighborhood.K2 | 64 |
| pso.neighborhood.K4 | 65 |
| pso.neighborhood.KN | 65 |
| pso.printbest | 66 |
| pso.Velocity | 66 |
| push | 67 |
| random.wheel | 67 |
| RepastFunction-class | 67 |
| saa.bolt | 68 |
| saa.neighborhood | 68 |
| saa.neighborhood1 | 69 |
| saa.neighborhoodH | 69 |
| saa.neighborhoodN | 70 |
| saa.tbyk | 70 |
| saa.tcte | 71 |
| saa.texp | 71 |
| scatterplotlohelper | 72 |
| searchrow | 72 |
| SetLogLevel | 73 |
| show.comp1 | 73 |
| slope | 74 |
| slopes | 74 |
| sortSolution | 75 |
| summarize.comp1 | 75 |
| tabu.getNeighbors | 76 |
| tabu.istabu | 76 |
| upperBound | 77 |
| xmeanci1 | 77 |

| | |
|------------------------|-----------|
| <i>abm.acor</i> | 5 |
| xmeanci2 | 78 |
| xyplothelper | 78 |
| Index | 79 |

| | |
|-----------------|---|
| <i>abm.acor</i> | <i>Ant colony optimization for continuous domains</i> |
|-----------------|---|

Description

An implementation of Ant Colony Optimization algorithm for continuous variables.

Usage

```
abm.acor(objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An appropriate instance from a subclass of Options class |

References

[1] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), 1155-1173. <http://doi.org/10.1016/j.ejor.2006.06.046>

Examples

```
## Not run:
f<- PlainFunction$new(f0.rosenbrock2)

f$Parameter(name="x1",min=-100,max=100)
f$Parameter(name="x2",min=-100,max=100)

extremize("acor", f)

## End(Not run)
```

 abm.ees1

EvoPER Evolutionary Strategy 1

Description

This function tries to provide a rough approximation to best solution when no information is available for the correct range of input parameters for the objective function. It can be useful for studying the behavior of individual-based models with high variability in the output variables showing non-linear behaviors.

Usage

```
abm.ees1(objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An appropriate instance from a subclass of Options class |

Examples

```
## Not run:
f<- PlainFunction$new(f0.rosenbrock2)

f$Parameter(name="x1",min=-100,max=100)
f$Parameter(name="x2",min=-100,max=100)

extremize("ees1", f)

## End(Not run)
```

 abm.ees2

EvoPER Evolutionary Strategy 2

Description

This function tries to provide a rough approximation to best solution when no information is available for the correct range of input parameters for the objective function. It can be useful for studying the behavior of individual-based models with high variability in the output variables showing non-linear behaviors.

Usage

```
abm.ees2(objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An appropriate instance from a subclass of Options class |

Examples

```
## Not run:  
f<- PlainFunction$new(f0.rosenbrock2)  
  
f$Parameter(name="x1",min=-100,max=100)  
f$Parameter(name="x2",min=-100,max=100)  
  
extremize("ees2", f)  
  
## End(Not run)
```

abm.ga

Genetic Algorithm metaheuristic

Description

An implementation of Genetic Algorithm metaheuristic for parameter estimation

Usage

```
abm.ga(objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An appropriate instance from a subclass of Options class |

References

[1] John Henry Holland (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. [2] Zbigniew Michalewicz (1994). "Genetic Algorithms + Data Structures = Evolution Programs (2nd Ed.)". Springer-Verlag, Berlin, Heidelberg. ISBN 3540580905.

Examples

```
## Not run:
f<- PlainFunction$new(f0.rosenbrock2)

f$Parameter(name="x1",min=-100,max=100)
f$Parameter(name="x2",min=-100,max=100)

or

f$Parameter0(name="x1",levels=c(0:4))
f$Parameter0(name="x2",levels=c(-2,-1,0,1,2))

extremize("tabu", f)

## End(Not run)
```

abm.pso

abm.pso

Description

An implementaion of Particle Swarm Optimization method for parameter estimation of Individual-based models.

Usage

```
abm.pso(objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An apropiate instance from a subclass of Options class |

References

- [1] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Proceedings of ICNN 95 - International Conference on Neural Networks (Vol. 4, pp. 1942-1948). IEEE.
- [2] Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. Swarm Intelligence, 1(1), 33-57.

Examples

```
## Not run:
f<- PlainFunction$new(f0.rosenbrock2)

f$Parameter(name="x1",min=-100,max=100)
f$Parameter(name="x2",min=-100,max=100)
```



```

    extremize("pso", f)

## End(Not run)

```

 abm.saa

abm.saa

Description

An implementation of Simulated Annealing Algorithm optimization method for parameter estimation of Individual-based models.

Usage

```
abm.saa(objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An appropriate instance from a subclass of Options class |

Value

The best solution.

References

[1] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598).

Examples

```

## Not run:
f<- PlainFunction$new(f0.rosenbrock2)

f$Parameter(name="x1",min=-100,max=100)
f$Parameter(name="x2",min=-100,max=100)

extremize("saa", f)

## End(Not run)

## Not run:
## A Repast defined function
f<- RepastFunction$new("/usr/models/BactoSim(HaldaneEngine-1.0)","ds::Output",300)

## or a plain function

```

```
f1<- function(x1,x2,x3,x4) {
  10 * (x1 - 1)^2 + 20 * (x2 - 2)^2 + 30 * (x3 - 3)^2 + 40 * (x4 - 4)^2
}

f<- PlainFunction$new(f1)

f$addFactor(name="cyclePoint",min=0,max=90)
f$addFactor(name="conjugationCost",min=0,max=100)
f$addFactor(name="pilusExpressionCost",min=0,max=100)
f$addFactor(name="gamma0",min=1,max=10)

abm.saa(f, 100, 1, 100, 0.75)

## End(Not run)
```

abm.tabu

Tabu Search metaheuristic

Description

An implementation of Tabu Search algorithm for parameter estimation

Usage

```
abm.tabu(objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An appropriate instance from a subclass of Options class |

References

- [1] Fred Glover (1989). "Tabu Search - Part 1". ORSA Journal on Computing, 190-206. doi:10.1287/ijoc.1.3.190.
- [2] Fred Glover (1990). "Tabu Search - Part 2". ORSA Journal on Computing, 4-32. doi:10.1287/ijoc.2.1.4.

Examples

```
## Not run:
f<- PlainFunction$new(f0.rosenbrock2)

f$Parameter(name="x1",min=-100,max=100)
f$Parameter(name="x2",min=-100,max=100)

or

f$Parameter0(name="x1",levels=c(0:4))
```

```
f$Parameter0(name="x2",levels=c(-2,-1,0,1,2))  
extremize("tabu", f)  
## End(Not run)
```

acor.archive

acor.archive

Description

This function is used for creating and maintaining the ACO archive 'T'. The function keeps the track of 'k' solution in the archive.

Usage

```
acor.archive(s, f, w, k, T = NULL)
```

Arguments

| | |
|---|----------------------------|
| s | The solution 'ants' |
| f | The evaluation of solution |
| w | The weight vector |
| k | The archive size |
| T | The current archive |

Value

The solution archive

References

[1] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), 1155-1173. <http://doi.org/10.1016/j.ejor.2006.06.046>

`acor.F`*acor.F*

Description

Helper function for extracting the 'F' function evaluations from archive ACOOr 'T'

Usage`acor.F(T)`**Arguments**

T The solution archive

Value

The F matrix

`acor.lthgaussian`*Select the lth gaussian function*

Description

Given a weight vector calculate the probabilities of selecting the lth gaussian function and return the index of lth gaussian selected with probability p

Usage`acor.lthgaussian(W)`**Arguments**

W The vector of weights

Value

The index of lth gaussian function

References

[1] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. European Journal of Operational Research, 185(3), 1155-1173. <http://doi.org/10.1016/j.ejor.2006.06.046>

| | |
|--------|---------------|
| acor.N | <i>acor.N</i> |
|--------|---------------|

Description

Helper function for getting the size of solution

Usage

acor.N(T)

Arguments

T The solution archive

Value

The size 'n' of a solution 's'

| | |
|--------------------|---|
| acor.probabilities | <i>Gaussian kernel choosing probability</i> |
|--------------------|---|

Description

Calculate the probability of choosing the lth Gaussian function

Usage

acor.probabilities(W, l = NULL)

Arguments

W The vector of weights
 l The lth element of algorithm solution archive T

Value

The vector of probabilities 'p'

References

[1] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), 1155-1173. <http://doi.org/10.1016/j.ejor.2006.06.046>

| | |
|--------|---------------|
| acor.S | <i>acor.S</i> |
|--------|---------------|

Description

Helper function for extracting solution 'S' from archive 'T'

Usage

acor.S(T)

Arguments

| | |
|---|----------------------|
| T | The solution archive |
|---|----------------------|

Value

The solution matrix

| | |
|------------|----------------------------------|
| acor.sigma | <i>Sigma calculation for ACO</i> |
|------------|----------------------------------|

Description

Calculate the value of sigma

Usage

acor.sigma(Xi, k, T)

Arguments

| | |
|----|---------------------------|
| Xi | The algorithm parameter |
| k | The solution archive size |
| T | The solution archive |

Value

The sigma value

References

[1] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), 1155-1173. <http://doi.org/10.1016/j.ejor.2006.06.046>

acor.updateants *acor.updateants*

Description

Update the solution using the gaussian kernel

Usage

acor.updateants(S, N, W, t.mu, t.sigma)

Arguments

| | |
|---------|--|
| S | The current solution ants |
| N | The number of required ants in solution |
| W | The weight vector |
| t.mu | The 'mean' from solution archive |
| t.sigma | The value of sigma from solution archive |

Value

The new solution ants

References

[1] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. European Journal of Operational Research, 185(3), 1155-1173. <http://doi.org/10.1016/j.ejor.2006.06.046>

acor.W *acor.W*

Description

Helper function for extracting the 'W' function evaluations from archive ACOr 'T'

Usage

acor.W(T)

Arguments

| | |
|---|----------------------|
| T | The solution archive |
|---|----------------------|

Value

The weight vector

`acor.weigth`
Weight calculation for ant colony optimization

Description

Calculates the weight element of ACO algorithm for the solution archive.

Usage

```
acor.weigth(q, k, l)
```

Arguments

| | |
|----------------|---|
| <code>q</code> | The Algorithm parameter. When small best-ranked solution is preferred |
| <code>k</code> | The Archive size |
| <code>l</code> | The lth element of algorithm solution archive T |

Value

A scalar or a vector with calculated `weigth`.

References

[1] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), 1155-1173. <http://doi.org/10.1016/j.ejor.2006.06.046>

`assert`
assert

Description

The `assert` function stop the execution if the logical expression given by the parameter `expression` is false.

Usage

```
assert(expression, string)
```

Arguments

| | |
|-------------------------|---|
| <code>expression</code> | Some logical expression |
| <code>string</code> | The text message to show if <code>expression</code> does not hold |

bestFitness *bestFitness*

Description

Given a set S of N solutions created with `sortSolution`, this function returns the fitness component for the best solution.

Usage

`bestFitness(S)`

Arguments

S The solution set

Value

The best fitness value

bestSolution *bestSolution*

Description

Given a set S of N solutions created with `sortSolution`, this function returns the best solution found.

Usage

`bestSolution(S)`

Arguments

S The solution set

Value

The best solution

| | |
|------|-------------|
| cbuf | <i>cbuf</i> |
|------|-------------|

Description

Simple implementation of a circular buffer.

Usage

```
cbuf(b, v, e)
```

Arguments

| | |
|---|---|
| b | The variable holding the current buffer content |
| v | The new valued to be added to b |
| e | The length of circular buffer |

Value

The buffer b plus the element v minus the least recently added element

| | |
|---------------------|----------------------------|
| compare.algorithms1 | <i>compare.algorithms1</i> |
|---------------------|----------------------------|

Description

Compare the number of function evaluations and convergence for the following optimization algorithms, ("saa","pso","acor","ees1").

Usage

```
compare.algorithms1(
  F,
  seeds = c(27, 2718282, 36190727, 3141593, -91190721, -140743, 1321)
)
```

Arguments

| | |
|-------|--|
| F | The function to be tested |
| seeds | The random seeds which will be used for testing algorithms |

Examples

```
## Not run:
rm(list=ls())
d.cigar4<- compare.algorithms1(f0.cigar4)
d.schaffer4<- compare.algorithms1(f0.schaffer4)
d.griewank4<- compare.algorithms1(f0.griewank4)
d.bohachevsky4<- compare.algorithms1(f0.bohachevsky4)
d.rosenbrock4<- compare.algorithms1(f0.rosenbrock4)

## End(Not run)
```

contourplotheper *contourplotheper*

Description

Simple helper function for countour plots

Usage

```
contourplotheper(
  d,
  x,
  y,
  z,
  nbins = 32,
  binwidth = c(10, 10),
  points = c(300, 300),
  title = NULL
)
```

Arguments

| | |
|----------|--|
| d | A data frame. |
| x | A string with the dataframe column name for x axis. |
| y | A string with the dataframe column name for y axis. |
| z | A string with the dataframe column name for z axis. |
| nbins | The number bins. The default is 32. |
| binwidth | The binwidths for 'kde2d'. Can be an scalar or a vector. |
| points | The number of grid points. Can be an scalar or a vector. |
| title | The optional plot title. May be omitted. |

| | |
|----------------|-----------------------|
| ees1.challenge | <i>ees1.challenge</i> |
|----------------|-----------------------|

Description

Repeat the evaluation of best solution to tackle with variability.

Usage

```
ees1.challenge(solution, objective)
```

Arguments

| | |
|-----------|------------------------|
| solution | The Problem solution |
| objective | The objective function |

| | |
|--------------|---------------------|
| ees1.explore | <i>ees1.explore</i> |
|--------------|---------------------|

Description

Explore the solution space on the neighborhood of solution 's' in order to find a new best.

Usage

```
ees1.explore(s, weight, p = 0.01)
```

Arguments

| | |
|--------|---------------------------|
| s | The Problem solution |
| weight | The exploration intensity |
| p | The mutation probability |

 ees1.mating

ees1.mating

Description

This function 'mix' the elements present in the solution. The parameter 'mu' controls the intensity of mixing. Low values give preference to best solution components and high values make the values being select randomly.

Usage

```
ees1.mating(solution, mu)
```

Arguments

| | |
|----------|---|
| solution | The Problem solution |
| mu | The mixing intensity ratio, from 0 to 1. The mix intensity controls de the probability of chosing a worst solutions |

 ees1.mating1

ees1.mating1

Description

This function 'mix' the elements present in the solution. The parameter 'mu' controls the intensity of mixing. Low values give preference to best solution components and high values make the values being select randomly.

Usage

```
ees1.mating1(solution, mu)
```

Arguments

| | |
|----------|---|
| solution | The Problem solution |
| mu | The mixing intensity ratio, from 0 to 1. The mix intensity controls de the probability of chosing a worst solutions |

| | |
|---------------|----------------------|
| ees1.mutation | <i>ees1.mutation</i> |
|---------------|----------------------|

Description

Performs the mutation on generated solution

Usage

```
ees1.mutation(solution, mates, p = 0.01)
```

Arguments

| | |
|----------|--------------------------|
| solution | The Problem solution |
| mates | The mixed parents |
| p | The mutation probability |

| | |
|--------------------|---------------------------|
| ees1.recombination | <i>ees1.recombination</i> |
|--------------------|---------------------------|

Description

Performs the recombination on solution

Usage

```
ees1.recombination(solution, mates)
```

Arguments

| | |
|----------|----------------------|
| solution | The Problem solution |
| mates | The mixed parents |

| | |
|----------------|----------------------|
| ees1.selection | <i>ees.selection</i> |
|----------------|----------------------|

Description

Select the elements with best fitness but accept uphill moves with probability 'kkappa'.

Usage

```
ees1.selection(s0, s1, kkappa)
```

Arguments

| | |
|--------|-------------------------------|
| s0 | The current best solution set |
| s1 | The new solution |
| kkappa | The selection pressure |

| | |
|------------|-------------------|
| elog.debug | <i>elog.debug</i> |
|------------|-------------------|

Description

Wrapper for logging debug messages.

Usage

```
elog.debug(...)
```

Arguments

| | |
|-----|---|
| ... | Variable number of arguments including a format string. |
|-----|---|

| | |
|------------|-------------------|
| elog.error | <i>elog.error</i> |
|------------|-------------------|

Description

Wrapper for logging error messages.

Usage

```
elog.error(...)
```

Arguments

| | |
|-----|---|
| ... | Variable number of arguments including a format string. |
|-----|---|

| | |
|------------------------|------------------|
| <code>elog.info</code> | <i>elog.info</i> |
|------------------------|------------------|

Description

Wrapper for logging info messages.

Usage

```
elog.info(...)
```

Arguments

... Variable number of arguments including a format string.

| | |
|-------------------------|-------------------|
| <code>elog.level</code> | <i>elog.level</i> |
|-------------------------|-------------------|

Description

Configure the current log level

Usage

```
elog.level(level = NULL)
```

Arguments

level The log level (ERROR|WARN|INFO|DEBUG)

Value

The log level

| | |
|---------------|----------------------|
| enforceBounds | <i>enforceBounds</i> |
|---------------|----------------------|

Description

Checks if parameters fall within upper and lower bounds

Usage

```
enforceBounds(particles, factors)
```

Arguments

| | |
|-----------|---|
| particles | The particle set |
| factors | the defined range for objective function parameters |

Value

The particle inside the valid limits

| | |
|-------------|--------------------|
| es.evaluate | <i>es.evaluate</i> |
|-------------|--------------------|

Description

For each element in solution 's' evaluate the respective fitness.

Usage

```
es.evaluate(f, s, enforce = TRUE)
```

Arguments

| | |
|---------|---|
| f | A reference to an instance of objective function |
| s | The set of solutions |
| enforce | If true the values are enforced to fall within provided range |

Value

The solution ordered by its fitness.

| | |
|-----------------|------------------|
| Estimates-class | <i>Estimates</i> |
|-----------------|------------------|

Description

A simple class for encapsulating the return of metaheuristic methods

| | |
|-----------|------------------|
| extremize | <i>extremize</i> |
|-----------|------------------|

Description

Entry point for optimization functions

Usage

```
extremize(type, objective, options = NULL)
```

Arguments

| | |
|-----------|--|
| type | The optimization method (aco,pso,saa,sda) |
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| options | An appropriate instance from a subclass of Options class |

Examples

```
## Not run:
f<- PlainFunction$new(f0.rosenbrock2)

f$Parameter(name="x1",min=-100,max=100)
f$Parameter(name="x2",min=-100,max=100)

extremize("pso", f)

## End(Not run)
```

| | |
|-----------|------------------|
| f0.ackley | <i>f0.ackley</i> |
|-----------|------------------|

Description

The ackley function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$. Domain $x_i \in [-32.768, 32.768]$, for all $i = 1, \dots, d$

Usage

```
f0.ackley(...)
```

Arguments

... The variadic list of function variables.

Value

The function value

References

<https://www.sfu.ca/~ssurjano/ackley.html>

| | |
|------------|-------------------|
| f0.ackley4 | <i>f0.ackley4</i> |
|------------|-------------------|

Description

The ackley function of four variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.ackley4(x1, x2, x3, x4)
```

Arguments

| | |
|----|------------------------------|
| x1 | The first function variable |
| x2 | The second function variable |
| x3 | The third function variable |
| x4 | The fourth function variable |

Value

The function value

f0.adtn.rosenbrock2 *f0.adtn.rosenbrock2*

Description

Two variable Rosenbrock function with random additive noise.

Usage

```
f0.adtn.rosenbrock2(x1, x2)
```

Arguments

| | |
|----|-------------|
| x1 | Parameter 1 |
| x2 | Parameter 2 |

f0.bohachevsky *f0.bohachevsky*

Description

The Bohachevsky function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.bohachevsky(...)
```

Arguments

... The variadic list of function variables.

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

| | |
|-----------------|------------------------|
| f0.bohachevsky4 | <i>f0.bohachevsky4</i> |
|-----------------|------------------------|

Description

The Bohachevsky function of four variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

f0.bohachevsky4(x1, x2, x3, x4)

Arguments

| | |
|----|------------------------------|
| x1 | The first function variable |
| x2 | The second function variable |
| x3 | The third function variable |
| x4 | The fourth function variable |

Value

The function value

| | |
|----------|-----------------|
| f0.cigar | <i>f0.cigar</i> |
|----------|-----------------|

Description

The Cigar function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

f0.cigar(...)

Arguments

| | |
|-----|--|
| ... | The variadic list of function variables. |
|-----|--|

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

`f0.cigar4`*f0.cigar4*

Description

The Cigar function of four variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.cigar4(x1, x2, x3, x4)
```

Arguments

| | |
|-----------------|------------------------------|
| <code>x1</code> | The first function variable |
| <code>x2</code> | The second function variable |
| <code>x3</code> | The third function variable |
| <code>x4</code> | The fourth function variable |

Value

The function value

`f0.griewank`*f0.griewank*

Description

The griewank function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.griewank(...)
```

Arguments

| | |
|------------------|--|
| <code>...</code> | The variadic list of function variables. |
|------------------|--|

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

| | |
|--------------|---------------------|
| f0.griewank4 | <i>f0.griewank4</i> |
|--------------|---------------------|

Description

The griewank function of four variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.griewank4(x1, x2, x3, x4)
```

Arguments

| | |
|----|------------------------------|
| x1 | The first function variable |
| x2 | The second function variable |
| x3 | The third function variable |
| x4 | The fourth function variable |

Value

The function value

| | |
|---------------------|----------------------------|
| f0.nlnn.rosenbrock2 | <i>f0.nlnn.rosenbrock2</i> |
|---------------------|----------------------------|

Description

Two variable Rosenbrock function with random additive noise.

Usage

```
f0.nlnn.rosenbrock2(x1, x2)
```

Arguments

| | |
|----|-------------|
| x1 | Parameter 1 |
| x2 | Parameter 2 |

f0.periodtuningpp *Period tuning for Predator-Prey base*

Description

This function is an example on how EvoPER can be used for estimating the parameter values in order to produce oscillations with the desired period. It is not intended to be used directly, the provided wrappers should be instead.

Usage

```
f0.periodtuningpp(x1, x2, x3, x4, period)
```

Arguments

| | |
|--------|-----------------------------------|
| x1 | The growth rate of prey |
| x2 | The decay rate of predator |
| x3 | The predating effect on prey |
| x4 | The predating effecto on predator |
| period | The desired oscilation period |

Value

The solution fitness cost

f0.periodtuningpp12 *Period tuning of 12 time units for Predator-Prey*

Description

This function is an example on how EvoPER can be used for estimating the parameter values in order to produce oscillations with the desired period.

Usage

```
f0.periodtuningpp12(x1, x2, x3, x4)
```

Arguments

| | |
|----|-----------------------------------|
| x1 | The growth rate of prey |
| x2 | The decay rate of predator |
| x3 | The predating effect on prey |
| x4 | The predating effecto on predator |

Value

The solution fitness cost

Examples

```
## Not run:
rm(list=ls())
set.seed(-27262565)
f<- PlainFunction$new(f0.periodtuningpp12)
f$Parameter(name="x1",min=0.5,max=2)
f$Parameter(name="x2",min=0.5,max=2)
f$Parameter(name="x3",min=0.5,max=2)
f$Parameter(name="x4",min=0.5,max=2)
extremize("pso", f)

## End(Not run)
```

f0.periodtuningpp24 *Period tuning of 24 time units for Predator-Prey*

Description

This function is an example on how EvoPER can be used for estimating the parameter values in order to produce oscillations with the desired period.

Usage

```
f0.periodtuningpp24(x1, x2, x3, x4)
```

Arguments

| | |
|----|-----------------------------------|
| x1 | The growth rate of prey |
| x2 | The decay rate of predator |
| x3 | The predating effect on prey |
| x4 | The predating effecto on predator |

Value

The solution fitness cost

Examples

```
## Not run:
rm(list=ls())
set.seed(-27262565)
f<- PlainFunction$new(f0.periodtuningpp24)
f$Parameter(name="x1",min=0.5,max=2)
f$Parameter(name="x2",min=0.5,max=2)
f$Parameter(name="x3",min=0.5,max=2)
f$Parameter(name="x4",min=0.5,max=2)
extremize("pso", f)

## End(Not run)
```

f0.periodtuningpp48 *Period tuning of 48 time units for Predator-Prey*

Description

This function is an example on how EvoPER can be used for estimating the parameter values in order to produce oscillations with the desired period.

Usage

```
f0.periodtuningpp48(x1, x2, x3, x4)
```

Arguments

| | |
|----|-----------------------------------|
| x1 | The growth rate of prey |
| x2 | The decay rate of predator |
| x3 | The predating effect on prey |
| x4 | The predating effecto on predator |

Value

The solution fitness cost

Examples

```
## Not run:
rm(list=ls())
set.seed(-27262565)
f<- PlainFunction$new(f0.periodtuningpp24)
f$Parameter(name="x1",min=0.5,max=2)
f$Parameter(name="x2",min=0.5,max=2)
f$Parameter(name="x3",min=0.5,max=2)
f$Parameter(name="x4",min=0.5,max=2)
extremize("pso", f)
```

```
## End(Not run)
```

f0.periodtuningpp72 *Period tuning of 72 time units for Predator-Prey*

Description

This function is an example on how EvoPER can be used for estimating the parameter values in order to produce oscillations with the desired period.

Usage

```
f0.periodtuningpp72(x1, x2, x3, x4)
```

Arguments

| | |
|----|-----------------------------------|
| x1 | The growth rate of prey |
| x2 | The decay rate of predator |
| x3 | The predating effect on prey |
| x4 | The predating effecto on predator |

Value

The solution fitness cost

Examples

```
## Not run:  
rm(list=ls())  
set.seed(-27262565)  
f<- PlainFunction$new(f0.periodtuningpp24)  
f$Parameter(name="x1",min=0.5,max=2)  
f$Parameter(name="x2",min=0.5,max=2)  
f$Parameter(name="x3",min=0.5,max=2)  
f$Parameter(name="x4",min=0.5,max=2)  
extremize("pso", f)  
  
## End(Not run)
```

f0.rosenbrock2 *f0.rosenbrock2*

Description

Two variable Rosenbrock function, where $f(1,1) = 0$

Usage

f0.rosenbrock2(x1, x2)

Arguments

| | |
|----|-------------|
| x1 | Parameter 1 |
| x2 | Parameter 2 |

f0.rosenbrock4 *f0.rosenbrock4*

Description

The rosenbrock function of 4 variables for testing optimization methods. The global optima for the function is given by $x_i = 1$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

f0.rosenbrock4(x1, x2, x3, x4)

Arguments

| | |
|----|------------------------------|
| x1 | The first function variable |
| x2 | The second function variable |
| x3 | The third function variable |
| x4 | The fourth function variable |

Value

The function value

| | |
|----------------|-----------------------|
| f0.rosenbrockn | <i>f0.rosenbrockn</i> |
|----------------|-----------------------|

Description

The rosenbrock function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 1$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.rosenbrockn(...)
```

Arguments

... The variadic list of function variables.

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

| | |
|-------------|--------------------|
| f0.schaffer | <i>f0.schaffer</i> |
|-------------|--------------------|

Description

The schaffer function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.schaffer(...)
```

Arguments

... The variadic list of function variables.

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

f0.schaffer4

f0.schaffer4

Description

The Schaffer function of four variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f0.schaffer4(x1, x2, x3, x4)
```

Arguments

| | |
|----|------------------------------|
| x1 | The first function variable |
| x2 | The second function variable |
| x3 | The third function variable |
| x4 | The fourth function variable |

Value

The function value

f0.schwefel

f0.schwefel

Description

The schwefel function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 420.96874636$, for all $i \in \{1 \dots N\}$, $f(x) = 0$. The range of x_i is $[-500, 500]$

Usage

```
f0.schwefel(...)
```

Arguments

| | |
|-----|--|
| ... | The variadic list of function variables. |
|-----|--|

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

 f0.schwefel4

f0.schwefel4

Description

The schwefel function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 420.96874636$, for all $i \in \{1 \dots N\}$, $f(x) = 0$. The range of x_i is $[-500, 500]$

Usage

```
f0.schwefel4(x1, x2, x3, x4)
```

Arguments

| | |
|----|------------------------------|
| x1 | The first function variable |
| x2 | The second function variable |
| x3 | The third function variable |
| x4 | The fourth function variable |

Value

The function value

f0.test

f0.test

Description

Simple test function $f(1,2,3,4) = 0$

Usage

```
f0.test(x1, x2, x3, x4)
```

Arguments

| | |
|----|-------------|
| x1 | Parameter 1 |
| x2 | Parameter 2 |
| x3 | Parameter 3 |
| x4 | Parameter 4 |

f1.ackley

f1.ackley

Description

The ackley function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$. Domain $x_i \in [-32.768, 32.768]$, for all $i = 1, \dots, d$

Usage`f1.ackley(x)`**Arguments**

`x` The vector of function parameters

Value

The function value

References

<https://www.sfu.ca/~ssurjano/ackley.html>

f1.adtn.rosenbrock2

f1.adtn.rosenbrock2

Description

Two variable Rosenbrock function with random additive noise.

Usage`f1.adtn.rosenbrock2(x)`**Arguments**

`x` Parameter vector

| | |
|----------------|-----------------------|
| f1.bohachevsky | <i>f1.bohachevsky</i> |
|----------------|-----------------------|

Description

The Bohachevsky function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

f1.bohachevsky(x)

Arguments

x The vector of function parameters

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

| | |
|----------|-----------------|
| f1.cigar | <i>f1.cigar</i> |
|----------|-----------------|

Description

The Cigar function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

f1.cigar(x)

Arguments

x The vector of function variables.

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

f1.griewank

f1.griewank

Description

The griewank function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

```
f1.griewank(x)
```

Arguments

x The vector of function parameters

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

f1.nlnn.rosenbrock2

f1.nlnn.rosenbrock2

Description

Two variable Rosenbrock function with random additive noise.

Usage

```
f1.nlnn.rosenbrock2(x)
```

Arguments

x Parameter vector

| | |
|----------------|-----------------------|
| f1.rosenbrock2 | <i>f1.rosenbrock2</i> |
|----------------|-----------------------|

Description

Two variable Rosenbrock function, where $f(c(1,1)) = 0$

Usage

f1.rosenbrock2(x)

Arguments

x Parameter vector

| | |
|----------------|-----------------------|
| f1.rosenbrockn | <i>f1.rosenbrockn</i> |
|----------------|-----------------------|

Description

The rosenbrock function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 1$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage

f1.rosenbrockn(x)

Arguments

x The vector of function parameters

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

f1.schaffer

f1.schaffer

Description

The schaffer function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 0$, for all $i \in \{1 \dots N\}$, $f(x) = 0$.

Usage`f1.schaffer(x)`**Arguments**

x The vector of function parameters

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

f1.schwefel

f1.schwefel

Description

The schwefel function of N variables for testing optimization methods. The global optima for the function is given by $x_i = 420.96874636$, for all $i \in \{1 \dots N\}$, $f(x) = 0$. The range of x_i is $[-500, 500]$

Usage`f1.schwefel(x)`**Arguments**

x The vector of function variables.

Value

The function value

References

<http://deap.gel.ulaval.ca/doc/dev/api/benchmarks.html>

| | |
|---------|----------------|
| f1.test | <i>f1.test</i> |
|---------|----------------|

Description

Simple test function $f(c(1,2,3,4)) = 0$

Usage

```
f1.test(x)
```

Arguments

| | |
|---|------------------|
| x | Parameter vector |
|---|------------------|

| | |
|--------------|---------------------|
| fixdfcolumns | <i>fixdfcolumns</i> |
|--------------|---------------------|

Description

Coerce dataframe columns to a specific type.

Usage

```
fixdfcolumns(df, cols = c(), skip = TRUE, type = as.numeric)
```

Arguments

| | |
|------|---|
| df | The data frame. |
| cols | The dataframe columns to be skipped or included. |
| skip | If TRUE the column names in 'cols' are skipped. When FALSE logic is inverted. |
| type | The type for which data frame columns must be converted. |

Value

The data frame with converted column types.

| | |
|------------------|-------------------------|
| generateSolution | <i>generateSolution</i> |
|------------------|-------------------------|

Description

Generates a problema solution using discrete leves

Usage

```
generateSolution(parameters, size)
```

Arguments

| | |
|------------|---------------------------------------|
| parameters | The Objective Function parameter list |
| size | The solution size |

Value

The solution set

| | |
|------------|-------------------|
| getFitness | <i>getFitness</i> |
|------------|-------------------|

Description

Given a set S of N solutions created with sortSolution, this function returns the solution component for the best solution.

Usage

```
getFitness(S, i = NULL)
```

Arguments

| | |
|---|---|
| S | The solution set |
| i | The fitness index, if null return the whole column. |

Value

The selected fitness entry

`GetLogLevel`*GetLogLevel*

Description

Get the current log level

Usage

```
GetLogLevel()
```

Value

The log level

`getSolution`*getSolution*

Description

Given a set S of N solutions created with `sortSolution`, this function returns the solution component. A solutions is a set of solutions and their associated fitness

Usage

```
getSolution(S)
```

Arguments

S The solution set

Value

The solution set

gm.mean

gm.mean

Description

Simple implementation for geometric mean

Usage

gm.mean(x)

Arguments

x data

Value

geometric mean for data

gm.sd

gm.sd

Description

Simple implementation for geometric standard deviation

Usage

gm.sd(x, mu = NULL)

Arguments

x data
mu The geometric mean. If not provided it is calculated.

Value

geometric standard deviation for data

| | |
|---------------|----------------------|
| histplotheper | <i>histplotheper</i> |
|---------------|----------------------|

Description

Simple helper for plotting histograms

Usage

```
histplotheper(d, x, title = NULL)
```

Arguments

| | |
|-------|---|
| d | A data frame. |
| x | A string with the dataframe column name for histogram |
| title | The plot title |

Value

A ggplot2 plot object

| | |
|--------------|---------------------|
| initSolution | <i>initSolution</i> |
|--------------|---------------------|

Description

Creates the initial Solution population taking into account the lower and upper bounds of provided experiment factors.

Usage

```
initSolution(parameters, N = 20, sampling = "mcs")
```

Arguments

| | |
|------------|--|
| parameters | The Objective Function parameter list |
| N | The size of Solution population |
| sampling | The population sampling scheme, namely <mcs hslffs> standing respectively for montecarlo sampling, latin hypercube sampling and full factorial sampling |

Value

A random set of solutions

| | |
|------------|-------------------|
| lowerBound | <i>lowerBound</i> |
|------------|-------------------|

Description

Checks if parameters is greater than the lower bounds

Usage

```
lowerBound(particles, factors)
```

Arguments

| | |
|-----------|---|
| particles | The particle set |
| factors | the defined range for objective function parameters |

Value

The particle greater than or equal to lower limit

| | |
|-----------|------------------|
| Magnitude | <i>Magnitude</i> |
|-----------|------------------|

Description

Calculates the magnitude order for a given value

Usage

```
Magnitude(v)
```

Arguments

| | |
|---|---------------------|
| v | The numerical value |
|---|---------------------|

Value

The magnitude order

| | |
|-------------|--------------------|
| naiveperiod | <i>naiveperiod</i> |
|-------------|--------------------|

Description

A naive approach for finding the period in a series of data points

Usage

naiveperiod(d)

Arguments

d The data to search period

Value

A list with the average period and amplitude

NetLogoFunction-class *NetLogoFunction*

Description

NetLogoFunction class

| | |
|-------------------|--------------------------|
| NLWrapper.FindJar | <i>NLWrapper.FindJar</i> |
|-------------------|--------------------------|

Description

Search for the netlogo jar file on the provided path

Usage

NLWrapper.FindJar(path)

Arguments

path The base path for searching

Value

The path for NetLogo jar file

NLWrapper.GetParameter

NLWrapper.GetParameter

Description

Gets the value of a model parameter

Usage

```
NLWrapper.GetParameter(obj, name)
```

Arguments

| | |
|------|--|
| obj | The object returned by NLWrapper.Model |
| name | The parameter name string or the collection of parameter names |

Value

The parameter values

Examples

```
## Not run:
rm(list=ls())
p<- "C:/Program Files/NetLogo 6.1.1/app"
m<- file.path(nlpath, "models", "Sample Models", "Biology", "Wolf Sheep Predation.nlogo")
o<- NLWrapper.Model(p, m)
v<- NLWrapper.GetParameter(o, c("initial-number-sheep"))

or

v<- NLWrapper.GetParameter(o, c("initial-number-sheep","initial-number-wolves"))

## End(Not run)
```

NLWrapper.Model

NLWrapper.Model

Description

This wrapper prepares the environment and instantiates the model

Usage

```
NLWrapper.Model(netlogodir, modelfile, dataset, maxtime)
```

Arguments

| | |
|------------|--|
| netlogodir | The base path of NetLogo installation |
| modelfile | The absolute path for NetLogo model file |
| dataset | The names of model variables |
| maxtime | The total number of iterations |

Examples

```
## Not run:
rm(list=ls())
p<- "C:/Program Files/NetLogo 6.1.1/app"
output<- c("count sheep", "count wolves")
m<- file.path(p, "models", "Sample Models", "Biology", "Wolf Sheep Predation.nlogo")
o<- NLWrapper.Model(p, m, output, 150)

## End(Not run)
```

NLWrapper.Run

NLWrapper.Run

Description

Executes a NetLogo Model using rNetLogo

Usage

```
NLWrapper.Run(obj, r = 1, seed = c())
```

Arguments

| | |
|------|--|
| obj | The object returned by NLWrapper.Model |
| r | The number of replications |
| seed | The collection of random seeds |

Examples

```
## Not run:
p<- "C:/Program Files/NetLogo 6.1.1/app"
m<- file.path(p, "models", "Sample Models", "Biology", "Wolf Sheep Predation.nlogo")
output<- c("count sheep", "count wolves")
o<- NLWrapper.Model(p, m, output, 150)
v<- NLWrapper.Run(o)
NLWrapper.Shutdown(o)
## End(Not run)
```

NLWrapper.RunExperiment

NLWrapper.RunExperiment

Description

Executes a NetLogo Model using rNetLogo

Usage

```
NLWrapper.RunExperiment(obj, r = 1, design, FUN)
```

Arguments

| | |
|--------|--|
| obj | The object returned by NLWrapper.Model |
| r | The number of replications |
| design | The desing matrix holding parameter sampling |
| FUN | THE calibration function. |

Value

A list containing the the parameters, the calibration functio output and the whole resultset

Examples

```
## Not run:
rm(list=ls())
objectivefn<- function(params, results) { 0 }

f<- AddFactor(name="initial-number-sheep",min=100,max=250)
f<- AddFactor(factors=f, name="initial-number-wolves",min=50,max=150)
f<- AddFactor(factors=f, name="grass-regrowth-time",min=30,max=100)
f<- AddFactor(factors=f, name="sheep-gain-from-food",min=1,max=50)
f<- AddFactor(factors=f, name="wolf-gain-from-food",min=1,max=100)
f<- AddFactor(factors=f, name="sheep-reproduce",min=1,max=20)
f<- AddFactor(factors=f, name="wolf-reproduce",min=1,max=20)

design<- AoE.LatinHypercube(factors=f)

p<- "C:/Program Files/NetLogo 6.1.1/app"
m<- file.path(p, "models", "Sample Models", "Biology", "Wolf Sheep Predation.nlogo")
output<- c("count sheep", "count wolves")
o<- NLWrapper.Model(p, m, output, 150)
v<- RunExperiment(o, r=1, design, objectivefn)
NLWrapper.Shutdown(o)

## End(Not run)
```

`NLWrapper.SetParameter`*NLWrapper.SetParameter*

Description

Set parameter values

Usage

```
NLWrapper.SetParameter(obj, parameters)
```

Arguments

| | |
|-------------------------|--|
| <code>obj</code> | The object returned by NLWrapper.Model |
| <code>parameters</code> | The data frame containing the parameters |

Examples

```
## Not run:  
rm(list=ls())  
p<- "C:/Program Files/NetLogo 6.1.1/app"  
m<- file.path(nlpath, "models", "Sample Models", "Biology", "Wolf Sheep Predation.nlogo")  
o<- NLWrapper.Model(p, m)  
  
## End(Not run)
```

`NLWrapper.SetRandomSeed`*NLWrapper.SetRandomSeed*

Description

Configures the random seed

Usage

```
NLWrapper.SetRandomSeed(obj, seed)
```

Arguments

| | |
|-------------------|--|
| <code>obj</code> | The object returned by NLWrapper.Model |
| <code>seed</code> | The new random seed |

NLWrapper.Shutdown *NLWrapper.Shutdown*

Description

This wrapper terminates RNetLogo execution environment

Usage

NLWrapper.Shutdown(obj)

Arguments

obj The object returned by [NLWrapper.Model](#)

ObjectiveFunction-class
ObjectiveFunction class

Description

The base class for optimization functions.

Fields

object The raw output of objective function
objective The objective function
parameters The parameter list for objective function
value The results from objective function

Options-class *Options*

Description

The base class for the options for the optimization metaheuristics

Fields

type The configuration type
neighborhood The neighborhood function for population methods
discrete Flag indicating that and specific algorithm is discrete or continuous
nlevelz Default value for generating parameter levels when range is provided, default value is 5
container The object holding the configuration options

| | |
|-------------------|--------------------|
| OptionsACOR-class | <i>OptionsACOR</i> |
|-------------------|--------------------|

Description

Options for ACOR method

| | |
|-------------------|--------------------|
| OptionsEES1-class | <i>OptionsEES1</i> |
|-------------------|--------------------|

Description

Options for EvoPER Evolutionary Strategy 1

| | |
|-------------------|--------------------|
| OptionsEES2-class | <i>OptionsEES2</i> |
|-------------------|--------------------|

Description

Options for Serial Dilutions method

Fields

dilutions The desired dilutions

| | |
|----------------|-----------------------|
| OptionsFactory | <i>OptionsFactory</i> |
|----------------|-----------------------|

Description

Instantiate the Options class required for the specific metaheuristic method.

Usage

```
OptionsFactory(type, v = NULL)
```

Arguments

| | |
|------|--------------------------|
| type | The metaheuristic method |
| v | The options object |

Value

Options object

OptionsGA-class *OptionsGA*

Description

Options for Genetic Algorithm optimization metaheuristic

OptionsPSO-class *OptionsPSO*

Description

Options for PSO optimization metaheuristic

OptionsSAA-class *OptionsSAA*

Description

Options for SAA method

Fields

temperature The temperature decay function

OptionsTS-class *OptionsTS*

Description

Options for Tabu search optimization metaheuristic

| | |
|----------------|-----------------------|
| paramconverter | <i>paramconverter</i> |
|----------------|-----------------------|

Description

Convert parameter from continuous to discrete and vice-versa if needed

Usage

```
paramconverter(parameters, discrete, levelz = 5)
```

Arguments

| | |
|------------|--|
| parameters | The current parameter set |
| discrete | The desired parameter type |
| levelz | When discrete is true the number of levels to be generated |

Value

The parameter collection casted to desired mode

| | |
|-------------------|--------------------------|
| partSolutionSpace | <i>partSolutionSpace</i> |
|-------------------|--------------------------|

Description

Creates the initial Solution population taking into account the lower and upper bounds of provided experiment factors. This method works by dividing the solution space into partitions of size 'd' and then creating a full factorial combination of partitions.

Usage

```
partSolutionSpace(parameters, d = 4)
```

Arguments

| | |
|------------|---------------------------------------|
| parameters | The Objective Function parameter list |
| d | The partition size. Default value 4. |

Value

A set of solutions

PlainFunction-class *PlainFunction*

Description

PlainFunction Class

`pop.first` *pop.first*

Description

pop an element

Usage

`pop.first(x)`

Arguments

x The element collection

Value

The first element added to list FIFO

`pop.last` *pop.last*

Description

pop an element

Usage

`pop.last(x)`

Arguments

x The element collection

Value

The last element added to list LIFO

predatorprey *predatorprey*

Description

The solver for Lotka-Volterra differential equation.

Usage

predatorprey(x1, x2, x3, x4)

Arguments

| | |
|----|----------------------------------|
| x1 | The growth rate of prey |
| x2 | The decay rate of predator |
| x3 | The predating effect on prey |
| x4 | The predating effect on predator |

Value

The ODE solution

predatorprey.plot0 *predatorprey.plot0*

Description

Generate a plot for the predator-prey ODE output.

Usage

predatorprey.plot0(x1, x2, x3, x4, title = NULL)

Arguments

| | |
|-------|--|
| x1 | The growth rate of prey |
| x2 | The decay rate of predator |
| x3 | The predating effect on prey |
| x4 | The predating effect on predator |
| title | The optional plot title. May be omitted. |

Value

An ggplot2 object

Examples

```
## Not run:  
predatorprey.plot0(1.351888, 1.439185, 1.337083, 0.9079049)  
  
## End(Not run)
```

predatorprey.plot1 *predatorprey.plot1*

Description

Simple wrapper for 'predatorprey.plot0' accepting the parameters as a list.

Usage

```
predatorprey.plot1(x, title = NULL)
```

Arguments

| | |
|-------|---|
| x | A list containing the values of predator/prey parameters c1, c2, c3 and c4 denoting respectively the growth rate of prey, the decay rate of predator, the predating effect on prey and the predating effect on predator |
| title | The optional plot title. May be omitted. |

Value

An ggplot2 object

Examples

```
## Not run:  
rm(list=ls())  
predatorprey.plot1(v$getBest()[1:4])  
  
## End(Not run)
```

pso.best

pso.best

Description

Search for the best particle solution which minimize the objective function.

Usage

```
pso.best(objective, particles)
```

Arguments

| | |
|-----------|--|
| objective | The results of evaluating the objective function |
| particles | The particles tested |

Value

The best particle

pso.chi

pso.chi

Description

Implementation of constriction coefficient

Usage

```
pso.chi(phi1, phi2)
```

Arguments

| | |
|------|---|
| phi1 | Acceleration coefficient toward the previous best |
| phi2 | Acceleration coefficient toward the global best |

Value

The calculated constriction coefficient

`pso.lbest`*pso.lbest*

Description

Finds the lbest for the particle 'i' using the topology function given by the topology parameter.

Usage

```
pso.lbest(i, pbest, topology)
```

Arguments

| | |
|-----------------------|-------------------------------|
| <code>i</code> | The particle position |
| <code>pbest</code> | The pbest particle collection |
| <code>topology</code> | The desired topology function |

Value

The lbest for i th particle

`pso.neighborhood.K2`*pso.neighborhood.K2*

Description

The neighborhood function for a simple linear topology where every particle has k = 2 neighbors

Usage

```
pso.neighborhood.K2(i, n)
```

Arguments

| | |
|----------------|---------------------------------|
| <code>i</code> | The particle position |
| <code>n</code> | the size of particle population |

pso.neighborhood.K4 *pso.neighborhood.K4*

Description

The von neumann neighborhood function for a lattice-based topology where every particle has k = 4 neighbors

Usage

pso.neighborhood.K4(i, n)

Arguments

| | |
|---|---------------------------------|
| i | The particle position |
| n | the size of particle population |

pso.neighborhood.KN *pso.neighborhood.KN*

Description

Simple helper method for 'gbest' neighborhood

Usage

pso.neighborhood.KN(i, n)

Arguments

| | |
|---|---------------------------------|
| i | The particle position |
| n | the size of particle population |

pso.printbest *pso.printbest*

Description

Shows the best particle of each of simulated generations

Usage

pso.printbest(objective, particles, generation, title)

Arguments

| | |
|------------|--|
| objective | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| particles | The current particle population |
| generation | The current generation |
| title | Some informational text to be shown |

pso.Velocity *pso.velocity*

Description

Calculates the PSO Velocity

Usage

pso.Velocity(W = 1, Vi, phi1, phi2, Pi, Pg, Xi)

Arguments

| | |
|------|---|
| W | Weight (Inertia weight or constriction coefficient) |
| Vi | Current Velocity vector |
| phi1 | Acceleration coefficient toward the previous best |
| phi2 | Acceleration coefficient toward the global best |
| Pi | Personal best |
| Pg | Neighborhood best |
| Xi | Particle vector |

Value

Updated velocity

| | |
|------|-------------|
| push | <i>push</i> |
|------|-------------|

Description

push an element

Usage

```
push(x, v)
```

Arguments

| | |
|---|----------------------------|
| x | The collection of elements |
| v | The value to be pushed |

Value

The collection of elements

| | |
|--------------|---------------------|
| random.wheel | <i>random.whell</i> |
|--------------|---------------------|

Description

A simple random seed generator

Usage

```
random.wheel()
```

Value

A random number for seeding

| | |
|----------------------|-----------------------|
| RepastFunction-class | <i>RepastFunction</i> |
|----------------------|-----------------------|

Description

RepastFunction class

| | |
|----------|-----------------|
| saa.bolt | <i>saa.bolt</i> |
|----------|-----------------|

Description

Temperature function boltzmann

Usage

```
saa.bolt(t0, k)
```

Arguments

| | |
|----|-------------------------|
| t0 | The current temperature |
| k | The annealing value |

Value

The new temperature

| | |
|------------------|-------------------------|
| saa.neighborhood | <i>saa.neighborhood</i> |
|------------------|-------------------------|

Description

Generates neighbor solutions for simulated annealing

Usage

```
saa.neighborhood(f, S, d, n)
```

Arguments

| | |
|---|--|
| f | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| S | The current solution to find a neighbor |
| d | The distance from current solution S distance = (max - min) * d |
| n | The number of parameters to be perturbed |

Value

The neighbor of solution S

saa.neighborhood1 *saa.neighborhood1*

Description

Generates neighbor solutions perturbing one parameter from current solution S picked randomly.

Usage

```
saa.neighborhood1(f, S, d)
```

Arguments

f An instance of ObjectiveFunction (or subclass) class [ObjectiveFunction](#)
S The current solution to find a neighbor
d The distance from current solution S distance = (max - min) * d

Value

The neighbor of solution of S

saa.neighborhoodH *saa.neighborhoodH*

Description

Generates neighbor solutions perturbing half parameters from current solution S.

Usage

```
saa.neighborhoodH(f, S, d)
```

Arguments

f An instance of ObjectiveFunction (or subclass) class [ObjectiveFunction](#)
S The current solution to find a neighbor
d The distance from current solution S distance = (max - min) * d

Value

The neighbor of solution of S

saa.neighborhoodN *saa.neighborhoodN*

Description

Generates neighbor solutions perturbing all parameters from current solution S.

Usage

saa.neighborhoodN(f, S, d)

Arguments

| | |
|---|--|
| f | An instance of ObjectiveFunction (or subclass) class ObjectiveFunction |
| S | The current solution to find a neighbor |
| d | The distance from current solution S distance = (max - min) * d |

Value

The neighbor of solution of S

saa.tbyk *saa.tbyk*

Description

Temperature function t/k

Usage

saa.tbyk(t0, k)

Arguments

| | |
|----|-------------------------|
| t0 | The current temperature |
| k | The annealing value |

Value

The new temperature

`saa.tcte`*saa.tcte*

Description

Temperature function $cte * t_0$

Usage

```
saa.tcte(t0, k)
```

Arguments

| | |
|----|-------------------------|
| t0 | The current temperature |
| k | The annealing value |

Value

The new temperature

`saa.texp`*saa.texp*

Description

Temperature function exponential

Usage

```
saa.texp(t0, k)
```

Arguments

| | |
|----|-------------------------|
| t0 | The current temperature |
| k | The annealing value |

Value

The new temperature

scatterplotlohelper *scatterplotlohelper*

Description

Simple helper for plotting 3d scatterplots

Usage

```
scatterplotlohelper(d, x, y, z, title = NULL)
```

Arguments

| | |
|-------|--|
| d | A data frame. |
| x | A string with the dataframe column name for x axis |
| y | A string with the dataframe column name for y axis |
| z | A string with the dataframe column name for z axis |
| title | The optional plot title. May be omitted. |

Value

A scatter3D plot

searchrow *searchrow*

Description

Search for a value value on a matrix

Usage

```
searchrow(ddata, value)
```

Arguments

| | |
|-------|-----------------------------------|
| ddata | The matrix containing the dataset |
| value | The value to search for |

Value

Boolean TRUE for those indexes matching value

| | |
|-------------|--------------------|
| SetLogLevel | <i>SetLogLevel</i> |
|-------------|--------------------|

Description

Configure the current log level for the package

Usage

```
SetLogLevel(level)
```

Arguments

| | |
|-------|---------------------------------------|
| level | The log level (ERROR WARN INFO DEBUG) |
|-------|---------------------------------------|

| | |
|------------|-------------------|
| show.comp1 | <i>show.comp1</i> |
|------------|-------------------|

Description

Generates a barplot comparing the number of evaluations for algorithms ("saa", "pso", "acor", "ees1").

Usage

```
show.comp1(mydata, what, title = NULL)
```

Arguments

| | |
|--------|---|
| mydata | The data generated with 'summarize.comp1' |
| what | The name of variable to plot on 'y' axis |
| title | the plot title |

Examples

```
## Not run:
p.a<- show.comp1(d.cigar4,"evals","(a) Cigar function")
p.b<- show.comp1(d.schaffer4,"evals","(b) Schafer function")
p.c<- show.comp1(d.griewank4,"evals","(c) Griewank function")
p.d<- show.comp1(d.bohachevsky4,"evals","(d) Bohachevsky function")

## End(Not run)
```

slope

slope

Description

Simple function for calculate the slope on the ith element position

Usage

slope(x, y, i)

Arguments

| | |
|---|--------------|
| x | The x vector |
| y | The y vector |
| i | The position |

Value

The slope

slopes

slopes

Description

Calculate all slopes for the discrete x,y series

Usage

slopes(x, y)

Arguments

| | |
|---|--------------|
| x | The x vector |
| y | The y vector |

Value

A vector with all slopes

| | |
|--------------|---------------------|
| sortSolution | <i>sortSolution</i> |
|--------------|---------------------|

Description

Sort solution by its respective fitness

Usage

```
sortSolution(s, f)
```

Arguments

| | |
|---|-------------------------------|
| s | Problem solution |
| f | The function evaluation for s |

| | |
|-----------------|------------------------|
| summarize.comp1 | <i>summarize.comp1</i> |
|-----------------|------------------------|

Description

Provides as summary with averaged values of experimental setup

Usage

```
summarize.comp1(mydata)
```

Arguments

| | |
|--------|---|
| mydata | The data frame generated with 'compare.algorithms1' |
|--------|---|

Value

The summarized data

| | |
|-------------------|--------------------------|
| tabu.getNeighbors | <i>tabu.getNeighbors</i> |
|-------------------|--------------------------|

Description

create neighbor solutions

Usage

```
tabu.getNeighbors(tabu, parameters, solution, size)
```

Arguments

| | |
|------------|-----------------------|
| tabu | The tabu list |
| parameters | The parameter set |
| solution | The current solution |
| size | The neighborhood size |

Value

The neighbor for solution

| | |
|-------------|--------------------|
| tabu.istabu | <i>tabu.istabu</i> |
|-------------|--------------------|

Description

Check whether a solution is present on tabulist

Usage

```
tabu.istabu(tabulist, solution)
```

Arguments

| | |
|----------|----------------------------------|
| tabulist | The matrix of tabu solutions |
| solution | The solution value to be checked |

Value

Boolean TRUE tabulist contains the solution

| | |
|------------|-------------------|
| upperBound | <i>upperBound</i> |
|------------|-------------------|

Description

Checks if parameters is below the upper bounds

Usage

```
upperBound(particles, factors)
```

Arguments

| | |
|-----------|---|
| particles | The particle set |
| factors | the defined range for objective function parameters |

Value

The particle inside the valid upper bound

| | |
|----------|-----------------|
| xmeanci1 | <i>xmeanci1</i> |
|----------|-----------------|

Description

Calculates confidence interval of mean for provided data with desired confidence level. This functions uses bootstrap resampling scheme for estimating the CI.

Usage

```
xmeanci1(x, alpha = 0.95)
```

Arguments

| | |
|-------|---|
| x | The data set for which CI will be calculated |
| alpha | The confidence level. The default value is 0.95 (95%) |

Value

The confidence interval for the mean calculated using 'boot.ci'

 xmeanci2

xmeanci2

Description

Calculates confidence interval of mean for provided data with desired confidence level.

Usage

```
xmeanci2(x, alpha = 0.95)
```

Arguments

| | |
|-------|---|
| x | The data set for which CI will be calculated |
| alpha | The confidence level. The default value is 0.95 (95%) |

Value

The confidence interval for the mean

xyplothelper

xyplothelper

Description

Simple helper for plotting xy dispersion points.

Usage

```
xyplothelper(d, x, y, title = NULL)
```

Arguments

| | |
|-------|--|
| d | A data frame. |
| x | A string with the dataframe column name for x axis |
| y | A string with the dataframe column name for y axis |
| title | The optional plot title. May be omitted. |

Value

A ggplot2 plot object

Index

abm.acor, 5
abm.ees1, 6
abm.ees2, 6
abm.ga, 7
abm.pso, 8
abm.saa, 9
abm.tabu, 10
acor.archive, 11
acor.F, 12
acor.lthgaussian, 12
acor.N, 13
acor.probabilities, 13
acor.S, 14
acor.sigma, 14
acor.updateants, 15
acor.W, 15
acor.weigth, 16
assert, 16

bestFitness, 17
bestSolution, 17

cbuf, 18
compare.algorithms1, 18
contourplotheper, 19

ees1.challenge, 20
ees1.explore, 20
ees1.mating, 21
ees1.mating1, 21
ees1.mutation, 22
ees1.recombination, 22
ees1.selection, 23
elog.debug, 23
elog.error, 23
elog.info, 24
elog.level, 24
enforceBounds, 25
es.evaluate, 25
Estimates (Estimates-class), 26
Estimates-class, 26
extremize, 26

f0.ackley, 27
f0.ackley4, 27
f0.adtn.rosenbrock2, 28
f0.bohachevsky, 28
f0.bohachevsky4, 29
f0.cigar, 29
f0.cigar4, 30
f0.griewank, 30
f0.griewank4, 31
f0.nlnn.rosenbrock2, 31
f0.periodtuningpp, 32
f0.periodtuningpp12, 32
f0.periodtuningpp24, 33
f0.periodtuningpp48, 34
f0.periodtuningpp72, 35
f0.rosenbrock2, 36
f0.rosenbrock4, 36
f0.rosenbrockn, 37
f0.schaffer, 37
f0.schaffer4, 38
f0.schwefel, 38
f0.schwefel4, 39
f0.test, 39
f1.ackley, 40
f1.adtn.rosenbrock2, 40
f1.bohachevsky, 41
f1.cigar, 41
f1.griewank, 42
f1.nlnn.rosenbrock2, 42
f1.rosenbrock2, 43
f1.rosenbrockn, 43
f1.schaffer, 44
f1.schwefel, 44
f1.test, 45
fixdfcolumns, 45

generateSolution, 46

- getFitness, 46
- GetLogLevel, 47
- getSolution, 47
- gm.mean, 48
- gm.sd, 48
- histplothead, 49
- initSolution, 49
- lowerBound, 50
- Magnitude, 50
- naiveperiod, 51
- NetLogoFunction
 - (NetLogoFunction-class), 51
- NetLogoFunction-class, 51
- NLWrapper.FindJar, 51
- NLWrapper.GetParameter, 52
- NLWrapper.Model, 52, 52, 53–56
- NLWrapper.Run, 53
- NLWrapper.RunExperiment, 54
- NLWrapper.SetParameter, 55
- NLWrapper.SetRandomSeed, 55
- NLWrapper.Shutdown, 56
- ObjectiveFunction, 5–10, 26, 66, 68–70
- ObjectiveFunction
 - (ObjectiveFunction-class), 56
- ObjectiveFunction-class, 56
- Options, 5–10, 26
- Options (Options-class), 56
- Options-class, 56
- OptionsACOR (OptionsACOR-class), 57
- OptionsACOR-class, 57
- OptionsEES1 (OptionsEES1-class), 57
- OptionsEES1-class, 57
- OptionsEES2 (OptionsEES2-class), 57
- OptionsEES2-class, 57
- OptionsFactory, 57
- OptionsGA (OptionsGA-class), 58
- OptionsGA-class, 58
- OptionsPSO (OptionsPSO-class), 58
- OptionsPSO-class, 58
- OptionsSAA (OptionsSAA-class), 58
- OptionsSAA-class, 58
- OptionsTS (OptionsTS-class), 58
- OptionsTS-class, 58
- paramconverter, 59
- partSolutionSpace, 59
- PlainFunction (PlainFunction-class), 60
- PlainFunction-class, 60
- pop.first, 60
- pop.last, 60
- predatorprey, 61
- predatorprey.plot0, 61
- predatorprey.plot1, 62
- pso.best, 63
- pso.chi, 63
- pso.lbest, 64
- pso.neighborhood.K2, 64
- pso.neighborhood.K4, 65
- pso.neighborhood.KN, 65
- pso.printbest, 66
- pso.Velocity, 66
- push, 67
- random.wheel, 67
- RepastFunction (RepastFunction-class), 67
- RepastFunction-class, 67
- saa.bolt, 68
- saa.neighborhood, 68
- saa.neighborhood1, 69
- saa.neighborhoodH, 69
- saa.neighborhoodN, 70
- saa.tbyk, 70
- saa.tcte, 71
- saa.texp, 71
- scatterplotthead, 72
- searchrow, 72
- SetLogLevel, 73
- show.comp1, 73
- slope, 74
- slopes, 74
- sortSolution, 75
- summarize.comp1, 75
- tabu.getNeighbors, 76
- tabu.istabu, 76
- upperBound, 77
- xmeanci1, 77
- xmeanci2, 78
- xythead, 78