Package 'pso'

July 23, 2025

Version 1.0.4

Date 2022-04-12

Title Particle Swarm Optimization

Author Claus Bendtsen <papyrus.bendtsen@gmail.com>.

Maintainer Claus Bendtsen <papyrus.bendtsen@gmail.com>

Depends R (>= 2.10.0), methods

Suggests numDeriv, stats

Description Provides an implementation of particle swarm optimisation consistent with the standard PSO 2007/2011 by Maurice Clerc. Additionally a number of ancillary routines are provided for easy testing and graphics.

License LGPL-3

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-12 16:10:06 UTC

Contents

pso-package *A Particle Swarm Optimizer*

Description

The package provides an implementation of particle swarm optimization which is consistent with the standard PSO 2007 and 2011 by Maurice Clerc et al. Additionally a number of ancillary routines are provided for easy testing and graphics.

Details

Package:	pso
Type:	Package
Version:	1.0.4
Date:	2022-04-12
License:	LGPL-3
Depends:	methods

The core function in the package is `psoptim` which can be used as a drop in replacement for `optim`. When used without additional control parameters the implementation is intended to be equivalent to SPSO 2007 (by M. Clerc et al.).

Control parameters can be specified for SPSO 2011 (in its basic implementation), to clamp the maximal velocity, provide restarting when the swarm converges to a region as well as using BFGS as a local search strategy. See `psoptim` for details.

Author(s)

Maintainer: Claus Bendtsen <papyrus.bendtsen@gmail.com>

See Also

`optim`.

Examples

```
## Not run:
## Some examples of using the functions in the package

## Using basic "optim" interface to minimize a function
set.seed(1)
psoptim(rep(NA,2),function(x) 20+sum(x^2-10*cos(2*pi*x)),
        lower=-5,upper=5,control=list(abstol=1e-8))

## Parabola
p <- test.problem("parabola",10) # one local=global minimum
set.seed(1)
```

```
o1 <- psoptim(p,control=list(trace=1,REPORT=50))
show(o1)

set.seed(1)
o2 <- psoptim(p,control=list(trace=1,REPORT=50,w=c(.7,.1)))
show(o2)

set.seed(1)
o3 <- psoptim(p,control=list(trace=1,REPORT=1,hybrid=TRUE))
show(o3) ## hybrid much faster

## Griewank
set.seed(2)
p <- test.problem("griewank",10) # lots of local minima
o1 <- psoptim(p,control=list(trace=1,REPORT=50))
show(o1)

## The above sometimes get stuck in a local minima.
## Adding a restart to increase robustness.
set.seed(2)
o2 <- psoptim(p,control=list(trace=1,REPORT=50,reltol=1e-4))
show(o2)

## An then adding the hybrid
set.seed(2)
o3 <- psoptim(p,control=list(trace=1,REPORT=50,reltol=1e-4,
              hybrid=TRUE,hybrid.control=list(maxit=10)))
show(o3)

## Rosenbrock
set.seed(1)
p <- test.problem("rosenbrock",1)
o1 <- psoptim(p,control=list(trace=1,REPORT=50))
show(o1)

## Change to fully informed
set.seed(1)
o2 <- psoptim(p,control=list(trace=1,REPORT=50,p=1))
show(o2)

## Rastrigin
p <- test.problem("rastrigin",10)
set.seed(1)
o1 <- psoptim(p,control=list(trace=1,REPORT=50))
show(o1)

set.seed(1)
o2 <- psoptim(p,control=list(trace=1,REPORT=50,hybrid=TRUE,
              hybrid.control=list(maxit=10)))
show(o2) # better
plot(o1,xlim=c(0,p@maxf),ylim=c(0,100))
lines(o2,col=2) # and much faster convergence
```

```
## Ackley
set.seed(1)
p <- test.problem("ackley",10)
o1 <- psoptim(p,control=list(trace=1,REPORT=50))
show(o1)

## End(Not run)
```

`getSuccessRate-methods` *Methods for Function getSuccessRate*

Description

Provides the success rate as the result of conducting a test. Only implemented method is for objects of class `"test.result"`

Methods

Calculates the success rate from the number of successful tests conducted as a function of the number of function evaluations used.

`signature(object = "test.result")` This method is used internally by the graphical functions. Returns a list with components:

feval: The number of function evaluations.

rate: The corresponding success rate (between 0 and 1).

See Also

`test.result`.

`lines-methods` *Methods for Function lines*

Description

Graphical methods for adding line segments to existing plots.

Methods

`signature(x = "test.result")` Add lines of the success rate versus the number of function evaluations for the test resulted provided as `x` to the current plot. Any additional arguments to the method will be passed on to `lines`. Typically this method is used to add new test results to an existing plot.

See Also

`lines`, `test.result`.

plot-methods *Plot methods for test.result objects*

Description

Graphical methods for plotting test results.

Methods

`signature(x = "test.result", y = "missing")` Produces a plot of the success rate versus the number of function evaluations for the test result provided as `x`. Any additional arguments to the method will be passed on to `plot`.

See Also

`plot`, `test.result`.

points-methods *Methods for Function points*

Description

Graphical methods for adding points to existing plots.

Methods

`signature(x = "test.result")` Add points with the success rate versus the number of function evaluations for the test resulted provided as `x` to the current plot. Any additional arguments to the method will be passed on to `points`. Typically this method is used to add new test results to an existing plot.

See Also

`points`, `test.result`.

psoptim *Particle Swarm Optimizer*

Description

General implementation of particle swarm optimization usable as a direct replacement for optim.

Usage

```
psoptim(par, fn, gr = NULL, ..., lower = -1, upper = 1, control = list())
```

Arguments

par	Vector with length defining the dimensionality of the optimization problem. Providing actual values of par are not necessary (NA is just fine). Included primarily for compatibility with optim but if values are provided within the lower and upper bounds then the first particle will be initialized to the position provided by par.
fn	A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
gr	A function to return the gradient if local search is BFGS. If it is NULL, a finite-difference approximation will be used.
...	Further arguments to be passed to fn and gr.
lower	Lower bounds on the variables.
upper	Upper bounds on the variables.
control	A list of control parameters. See "Details".

Details

By default this function performs minimization using a particle swarm algorithm, but it will maximize if control$fnscale is negative.

The default control arguments implies that the algorithm follows the Standard PSO 2007 implementation by Maurice Clerc, but the code also provides support for PSO 2011, clamping the maximal velocity, restarting when all particles converge to a single area and using BFGS as the local search direction.

The control argument is a list that can supply any of the following components:

trace: Non-negative integer. If positive, tracing information on the progress of the optimization is produced. Defaults to 0.

fnscale: An overall scaling to be applied to the value of fn and gr (if used) during optimization. If negative, turns the problem into a maximization problem. Optimization is performed on fn(par)/fnscale. Defaults to 1.

maxit: The maximum number of iterations. Defaults to 1000.

maxf: The maximum number of function evaluations (not considering any performed during numerical gradient computation). Defaults to `Inf`.

abstol: The absolute convergence tolerance. The method converges once the best fitness obtained is less than or equal to `abstol`. Defaults to `-Inf`.

reltol: The tolerance for restarting. Once the maximal distance between the best particle and all other particles is less than `reltol*d` the algorithm restarts. Defaults to `0` which disables the check for restarting.

REPORT: The frequency for reports if `control$trace` is positive. Defaults to `10`.

trace.stats: Logical; if `TRUE` statistics at every reporting step are collected and returned. Defaults to `FALSE`.

s: The swarm size. Defaults to `floor(10+2*sqrt(length(par)))` unless `type` is "SPSO2011" in which case the default is `40`.

k: The exponent for calculating number of informants. Defaults to `3`.

p: The average percentage of informants for each particle. A value of `1` implies that all particles are fully informed. Defaults to `1-(1-1/s)^k`.

w: The exploitation constant. A vector of length `1` or `2`. If the length is two, the actual constant used is gradially changed from `w[1]` to `w[2]` as the number of iterations or function evaluations approach the limit provided. Defaults to `1/(2*log(2))`.

c.p: The local exploration constant. Defaults to `.5+log(2)`.

c.g: The global exploration constant. Defaults to `.5+log(2)`.

d: The diameter of the search space. Defaults to the euclidean distance between `upper` and `lower`.

v.max: The maximal (euclidean) length of the velocity vector. Defaults to `NA` which disables clamping of the velocity. However, if specified the actual clamping of the length is `v.max*d`.

rand.order: Logical; if `TRUE` the particles are processed in random order. If `vectorize` is `TRUE` then the value of `rand.order` does not matter. Defaults to `TRUE`.

max.restart: The maximum number of restarts. Defaults to `Inf`.

maxit.stagnate: The maximum number of iterations without improvement. Defaults to `Inf`.

vectorize: Logical; if `TRUE` the particles are processed in a vectorized manner. This reduces the overhead associated with iterating over each particle and may be more time efficient for cheap function evaluations. Defaults to `FALSE`.

hybrid: If true, each normal PSO position update is followed by an L-BFGS-B search with the provided position as initial guess. This makes the implementation a hybrid approach. Defaults to `FALSE` which disables BFGS for the local search. Note that no attempt is done to control the maximal number of function evaluations within the local search step (this can be done separately through `hybrid.control`) but the number of function evaluations used by the local search method counts towards the limit provided by `maxf` AFTER the local search returns. To support a broader class of hybrid approaches a character vector can also be supplied with "off" being equivalent to false, "on" equivalent to true, and "improved" implying that the local search will only be performed when the swarm finds an improvement.

hybrid.control: List with any additional control parameters to pass on to `optim` when using L-BFGS-B for the local search. Defaults to `NULL`.

type: Character vector which describes which reference implementation of SPSO is followed. Can take the value of "SPSO2007" or "SPSO2011". Defaults to "SPSO2007".

Value

A list, compatible with the output from optim, with components:

par	The best set of parameters found.
value	The value of fn corresponding to par.
counts	A three-element vector containing the number of function evaluations, the number of iterations, and the number of restarts.
convergence	An integer code. 0 indicates that the algorithm terminated by reaching the absolute tolerance; otherwise: 1: Maximal number of function evaluations reached. 2: Maximal number of iterations reached. 3: Maximal number of restarts reached. 4: Maximal number of iterations without improvement reached.
message	A descriptive message of the reason for termination.

If trace is positive and trace.stats is TRUE additionally the component:

stats	A list of statistics collected at every reporting step with the following components: it A vector with the iteration numbers error A vector with the corresponding best fitness values obtained f A list with the corresponding current swarm fitness values as a vector x A list with the corresponding current swarm positions as a matrix

References

Default parameters follow:

Clerc, M. (2011) https://hal.archives-ouvertes.fr/hal-00764996/document. Notice that the SPSO 2011 implementation does not include any of the bells and whistles from the implementation by M. Clerc et al. and effectively only differes from the SPSO 2007 implementation in the default swarm size, how velocities are initiated and the update of velocities/positions which in the SPSO 2011 implementation are invariant to rotation.

The gradual change of w and clamping the maximal velocity is described in:

Parsopoulos, K.E. and Vrahatis M.N. (2002) *Recent approaches to global optimization problems through Particle Swarm Optimization*. Natural Computing 1: 235-306.

The restart (provided through reltol) is similar to:

Evers G.I. and Ghalia M.B. *Regrouping Particle Swarm Optimization: A New Global Optimization Algorithm with Improved Performance Consistency Across Benchmarks*. https://bee22.com/resources/Evers%202009.pdf

The hybrid approach is similar to:

Qin J., Yin Y. and Ban X. (2010) *A Hybrid of Particle Swarm Optimization and Local Search for Multimodal Functions*. Lecture Notes in Computer Science, Volume 6145/2010, 589-596, DOI: 10.1007/978-3-642-13495-1_72

See Also

optim, test.problem.

Examples

```
set.seed(1)
## Rastrigin function
psoptim(rep(NA,2),function(x) 20+sum(x^2-10*cos(2*pi*x)),
        lower=-5,upper=5,control=list(abstol=1e-8))

set.seed(1)
## Rastrigin function - local refinement with L-BFGS-B on improvements
psoptim(rep(NA,2),function(x) 20+sum(x^2-10*cos(2*pi*x)),
        lower=-5,upper=5,control=list(abstol=1e-8,hybrid="improved"))

## Griewank function
psoptim(rep(NA,2),function(x) sum(x*x)/4000-prod(cos(x/sqrt(1:2)))+1,
        lower=-100,upper=100,control=list(abstol=1e-2))

set.seed(1)
## Rastrigin function with reporting
o <- psoptim(rep(NA,2),function(x) 20+sum(x^2-10*cos(2*pi*x)),
             lower=-5,upper=5,control=list(abstol=1e-8,trace=1,REPORT=1,
             trace.stats=TRUE))
## Not run:
plot(o$stats$it,o$stats$error,log="y",xlab="It",ylab="Error")
points(o$stats$it,sapply(o$stats$f,min),col="blue",pch=2)

## End(Not run)
```

---

psoptim-methods *Methods for function psoptim (Particle Swarm Optimization)*

Description

General implementation of particle swarm optimization usable as a direct replacement for optim.

Methods

signature(par = "ANY", fn = "ANY", gr = "ANY", lower = "ANY", upper = "ANY"):

This is the standard replacement for optim without S4 object usage.

signature(par = "test.problem", fn = "missing", gr = "missing", lower = "missing", upper = "missing"):

This is for running PSO on a specific test problem. Typically this is invoked on repetitive runs of a test problem and used to assess the choice of parameters for the underlying PSO algorithm. The function is essentially a wrapper function for psoptim but returns an instance of test.result containing summary results.

show-methods *Methods for Function show*

Description

Displays descriptive information of the object provided as argument.

Methods

`signature(object = "test.problem")` Provide information on test problem. This includes: problem name, dimension, objective value, maximal number of function evaluations, and the number of test repetitions to perform.

`signature(object = "test.result")` Provide summary statistics for the test. This includes information on the mean, s.d., min and max obtained for the value over all conducted repetitions as well as the overall success rate (percentage of test runs for which the target objective was reached) and a measure of efficiency (the area under the success-rate curve normalized to the maximal area possible). Additionally displays the timing information for the test conducted.

test.problem *Convenience constructor for the test.problem class.*

Description

The method enables creating of objects of class `"test.problem"` for a few standard test problems.

Usage

```
test.problem(name, n.test = 100, dim, maxf, objective, lower, upper)
```

Arguments

`name`	The name of the test problem. Currently supports one of `"parabola"`, `"griewank"`, `"rosenbrock"`, `"rastrigin"`, or `"ackley"`.
`n.test`	The number of tests to perform.
`dim`	Override the default dimension of the problem.
`maxf`	Override the default maximal number of function evaluations for the problem.
`objective`	Override the default objective for the function.
`lower`	Override the default lower bounds for the problem.
`upper`	Override the default upper bounds for the problem.

Value

An object of class `"test.problem"`.

See Also

test.problem.

Examples

```
test.problem("rast")

test.problem("rast",dim=4,n.test=10)

test.problem("grie")
```

---

test.problem-class *Class "test.problem"*

Description

The class contains a test problem including domain definition and reference solution. Generally objects from the class facilitate easy testing of PSO with various parameters.

Objects from the Class

Objects can be created by calls of the form `new("test.problem", ...)`, but the convenience constructer `test.problem` is the usual approach.

Slots

name: The name of the test problem. Object of class `"character"`.

f: Function to be minimized. Object of class `"function"`.

grad: Gradient of `f`. Only used with BFGS for the local search. Object of class `"function"`.

n: Problem dimensionality. Object of class `"integer"`.

maxf: Maximal number of function evaluations to use. Object of class `"integer"`

objective: The absolute tolerance when running PSO. Object of class `"numeric"`.

ntest: The number of tests to perform. Object of class `"integer"`.

lower: The lower bounds. Object of class `"numeric"`.

upper: The upper bounds. Object of class `"numeric"`.

Methods

psoptim `signature(par = "test.problem", fn = "missing",gr = "missing", lower = "missing", upper = "missing")`: for running PSO on the test problem. See `psoptim-methods` for details.

show `signature(object = "test.problem")`: descriptive information of the test problem. See `show-methods` for details.

See Also

test.problem.

Examples

```
test.problem("rast")
test.problem("rast",10) # modified for 10 repetitions.

test.problem("para")
```

---

test.result-class *Class "test.result"*

Description

A container class with results from executing a (repetition of) test problem(s).

Objects from the Class

Objects can be created by calls of the form `new("test.result", ...)`, but the object is normally provided as the result of executing `psoptim` on an object of class `"test.problem"`.

Slots

`problem`**:** Object of class `"test.problem"`.

`result`**:** A list with each of the results from repetitive invocation of `psoptim` on `problem`.

`time`**:** The overall time taken for executing the test. Object of class `"numeric"`.

Methods

getSuccessRate `signature(object = "test.result")`: internal method used to calculate the success rate for a series of test results. See `getSuccessRate-methods` for details.

lines `signature(x = "test.result")`: add lines with the test result to an existing plot. See `lines-methods` for details.

plot `signature(x = "test.result", y = "missing")`: plot the test result. See `plot-methods` for details.

points `signature(x = "test.result")`: add points with the test result to an existing plot. See `points-methods` for details.

show `signature(object = "test.result")`: summary statistics of the test. See `show-methods` for details.

Examples

```
showClass("test.result")

set.seed(1)
t <- test.problem("rastrigin",10)
o <- psoptim(t)
show(o)

## Not run:
plot(o)

## End(Not run)
```

Index