Package 'OSCARS'

May 6, 2026

Type Package

Title Global Bounded Optimization by the OSCARS-II Algorithm

Version 0.1.2

Maintainer Chris Price <chrisj.price@canterbury.ac.nz>

Description A collection of general optimization routines based on variants of the One Side Cut Accelerated Random Search (OSCARS-II) algorithm (Price et al., 2020, <doi:10.1007/s10898-020-00928-6>). The main function , 'oscars()', performs black-box optimization of a general (including nonsmooth or discontinuous) function subject to simple bounds on the unknowns. If all bounds are finite, oscars searches globally. The main method implements a stochastic direct search method and is derivative free. Testing shows the OSCARS-II algorithm usually finds extrema with fewer function evaluations than similar global derivative-free methods.

License Apache License (>= 2)

Encoding UTF-8

Imports stats

RoxygenNote 7.3.3

NeedsCompilation no

Author Chris Price [aut, cre],
Trent McDonald [aut, ctb] (R packaging)

Repository CRAN

Date/Publication 2026-05-06 19:50:11 UTC

Contents

`oscars` *OSCARS-II bound constrained global optimization*

Description

Performs black-box optimization of a general function subject to bounds on the unknown parameters using a variant of the OSCARS-II algorithm (Price, Reale and Robertson (2020) <doi.org/10.1007/s10898-020-00928-6>). If all bounds are finite, Oscars acts as a global optimization algorithm. It has been adapted to handle infinite upper and lower bounds, in which case the method has the characteristics of a local method for nonsmooth problems. Oscars does not use or assume the existence of derivatives of the objective function. It is a low overhead method for cheaply evaluated black-box functions. Black-box optimization methods for arbitrary functions do not and cannot provide certificates of optimality if halted after a finite amount of time.

Oscars is a stochastic direct search method which uses only function values at selected points. It generates a finite sequence of nested boxes around a control point, and randomly samples each box once, in turn. A new set of nested boxes is formed if the current set is exhausted or a point better than the control point is found. In the latter case the better point replaces the control. Initially the control point is set to the better of an internal initial point and a user supplied start point (if given).

From time to time the control is reset alternately to a random point, or to the best known point. Each reset marks the end of one cycle and the start of the next. All even numbered cycles start with control points chosen randomly from the feasible region. All odd numbered cycles (other than the first) set the control point equal to the best known point.

Oscars either performs a fixed number of function evaluations, or it halts if progress stalls for a significant period of time. In both cases it returns the best known point and the function evaluated at that point.

Usage

```
oscars(fname, n, lwr, upr, ..., start = NULL, controls = oscars.control())
```

Arguments

`fname`	An R function to be minimized. This function must take a vector of parameter values as its first argument, and return a scalar. Additional arguments can be supplied via . . . Missing (NaN and NA) function values are acceptable as they are replaced with Inf when minimizing (or -Inf when maximizing).
`n`	The number of parameters with which `fname` is minimized.
`lwr`	A vector of lower bounds for the parameters of `fname`. If a single value `lwr` is supplied, this value will be used for all lower bounds. Lower bounds of minus infinity are acceptable. In order to maximize oscars effectiveness, it is recommended that the gap between upper and lower bounds not be unnecessarily wide.
`upr`	A vector of upper bounds for the parameters of `fname`. If a single value `upr` is supplied, this value will be used for all upper bounds. Upper bounds of infinity are acceptable. It is suggested that Inf be used for parameters that are unbounded above rather than a very large finite number as this signals the method to operate as a local search rather than attempting to cover all values between the bounds.

`...`	Additional parameters supplied to function `fname`.
`start`	This is an optional start point for the algorithm. It allows the user to direct the method to a region the user considers promising. If the start point is infeasible (i.e. violates some bounds) the closest feasible point to it is used. If a single value is provided, it is used for all dimensions. Default is null. The algorithm also generates an internal initial point as follows. When all bounds are finite, this is the centre point of the box. Otherwise each parameter is started at the average of its bounds when both are finite; if one bound is finite, it uses a feasible value near that bound; otherwise it uses the user supplied start value (if one is given) for that parameter, or zero otherwise.
`controls`	A list of oscar control parameters, such as iteration budget, tolerance, etc. See `oscars.control` for the full list and descriptions.

Value

A list containing the best known set of parameters found along with function value at the best known parameters. The number of function evaluations used, and reason for halting are also given.

Examples

```
# Camel function with global minima of f = -1.0316 at
# (0.0898,0.7127) and (0.0898,-0.7127) with four other local minima
camel <- function(par) {
  x = par[1]
  y = par[2]
  f = 4*x^2 - 2.1*x^4 + (1/3)*x^6 + x*y + 4*(y^4-y^2)
  return(f) }
out <- oscars(camel, n = 2, lwr = c(-5,-5), upr = c(5,5))

# How to use repeated upper and lower bounds.
# Bird function in 2 dimensions.  Global minimum = -106.7645367198
bird <- function(par)  {
  x1 = par[1];  x2 = par[2]
  f = sin(x1)*exp((1-cos(x2))^2) + cos(x2)*exp((1-sin(x1))^2) + (x1-x2)^2
  return(f)
} # end of bird function
out <- oscars(bird, 2, -10, 50)

# Hosaki function with global minimum of -2.3458 at (4,2) and one local minimum
hosaki <- function(par)  {
  x = par[1]
  y = par[2]
  f = (1 - 8*x + 7*x^2 - (7/3)*x^3 + (1/4)*x^4)*y*y*exp(-y)
  return(f) }
out <- oscars(hosaki, 2, 0, upr = c(5,6))

# The proper way to specify control parameters.
out <- oscars(hosaki, 2, lwr = c(0,0), upr = c(5,6),
  controls = oscars.control(nfmax = 100000, fTol=10*oscars.control()$fTol))

# An example of where the full function evaluation budget is used.
```

```
out <- oscars(hosaki,2,0,5,controls = oscars.control(nfmax=10000,fTol=-1))

# how to pass other values to the objective function
# Rosenbrocks "banana" function with global minimum of zero at (a, a^2)
rosenbrock <- function(par, a = 1, b = 100){
   f = (a - par[1])^2 + b*(par[2] - par[1]^2)^2
   return(f)
}
out <- oscars(rosenbrock, 2, -3, 3, a = 0.5)

# Providing a user start point to the algorithm.
# Weka_1 function with global minimum of 0 wherever x[1] = -1 and a local
# minimum at the origin.  Dimension n is arbitrary with bounds -1 <= x <= 2
weka_1 <- function(par)   {
  f1 = 1 + sqrt( sum( par^2 ))
  f2 = 4*par[1] + 4
  f = min(f1,f2)
  return(f)
}
out <- oscars(weka_1, 10, -1, 2, start = 0)

# An example of the use of infinite bounds.
# Active faces is a nonsmooth function with global minimum = 0 at the
# origin.   Standard bounds are 0 <= par <= 5.  Solution is on boundary.
activefaces <- function(par)  {
f1 = max( log( abs(par) + 1) )
f2 = log( abs( sum(par) ) + 1)
f = max(f1,f2)
return(f)
}
out <- oscars(activefaces, 10, 0, Inf, start = 4)
```

`oscars.control` *Control parameters for oscars routine*

Description

Provides control over oscar parameters, such as number of iterations, tolerance, etc.

Usage

```
oscars.control(
  nfmax = 50000,
  infol = 2,
  DoMax = FALSE,
  fTol = 1e-06,
  xTol = 1e-08
)
```

Arguments

nfmax	The maximum number of function evaluations to perform. Default for nfmax is 50000.
infol	Verbosity during iterations. If infol is positive, each new best function value is printed. Default is 1.
DoMax	logical variable set to TRUE if the objective is to be maximized. Default is FALSE.
fTol	Stopping tolerance for the objective function f. This tolerance is multiplied by the larger of the absolute value of the current objective function value and 1. This gives a relative tolerance for large f, and and absolute one otherwise. If fTol is negative the algorithm will do the maximum nfmax of allowed function evaluations and then stop. Default is 1e-6.
xTol	Tolerance in the decision variables which is used to define the minimum sampling box size along each axis. For each decision variable xTol is scaled by the larger of 1 and the magnitude of the current value of that variable. This yields a relative tolerance of xTol for large magnitude decision variables, and an absolute tolerance for small ones. Once the sampling box is less than tolerance along all axes, the sequence of nested sample boxes is ended. xTol must be positive and the algorithm will impose a minimum value of 1e-12. Difference between current and previous best known points must be within relative or absolute tolerance of xTol for oscars to halt before the function budget is exhausted. Default is 1e-8.

Details

A subset of parameters can be specified. All non-specified parameters revert to their defaults. No parameter abbreviations.

Value

A named list of control parameters for oscars.

Examples

```
oscars.control()  # default values
oscars.control(nfmax = 100000) # bump iteration budget
oscars.control(xTol = 10*oscars.control()$xTol) # increase xTol
```

---

print.oscars *Print method for 'oscars' objects*

Description

Prints an 'oscars' object showing minimized (or maximized) parameters and the optimization message.

Usage

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

Arguments

x	An 'oscars' object returned by oscars.
...	Included for compatibility with other print methods. Ignored here.

Value

No return value, called for side effects. Technically, NULL is returned invisibly.

See Also

oscars

Examples

```
# Branins camel function with global minimum of f = -1.0316 at
# (0.0898,0.7127) and (0.0898,-0.7127) with four other local minimizers
camel <- function(par) {
  x = par[1]
  y = par[2]
  f = 4*x^2 - 2.1*x^4 + (1/3)*x^6 + x*y + 4*(y^4-y^2)
  return(f) }
out <- oscars(camel, n = 2, lwr = c(-5,-5), upr = c(5,5))
out
```

---

summary.oscars *Summary method for 'oscars' objects*

Description

Summarizes an 'oscars' object. Shows an 'oscars' object's minimized (or maximized) parameters, optimization message, iterations, etc..

Usage

```
## S3 method for class 'oscars'
summary(object, ...)
```

Arguments

object	An 'oscars' object returned by oscars.
...	Ignored here. Included for use by other methods.

Value

No return value, called for side effects. Technically, NULL is returned invisibly.

See Also

oscars

Examples

```
# Branins camel function with global minimum of f = -1.0316 at
# (0.0898,0.7127) and (0.0898,-0.7127) with four other local minimizers
camel <- function(par) {
  x = par[1]
  y = par[2]
  f = 4*x^2 - 2.1*x^4 + (1/3)*x^6 + x*y + 4*(y^4-y^2)
  return(f) }
out <- oscars(camel, n = 2, lwr = c(-5,-5), upr = c(5,5))
summary(out)
```

Index