# Package 'DEoptim'

November 11, 2022

**Version** 2.2-8

**Title** Global Optimization by Differential Evolution

**Description** Implements the Differential Evolution algorithm for global optimization of a real-valued function of a real-valued parameter vector as described in Mullen et al. (2011) <doi:10.18637/jss.v040.i06>.

**Suggests** foreach, iterators, colorspace, lattice, parallelly

**Depends** parallel

**Imports** methods

**BugReports** https://github.com/ArdiaD/DEoptim/issues

**URL** https://github.com/ArdiaD/DEoptim

**License** GPL (>= 2)

**Repository** CRAN

**Maintainer** Katharine Mullen <mullenkate@gmail.com>

**NeedsCompilation** yes

**Author** David Ardia [aut] (<https://orcid.org/0000-0003-2823-782X>),
Katharine Mullen [aut, cre],
Brian Peterson [aut],
Joshua Ulrich [aut],
Kris Boudt [ctb]

**Date/Publication** 2022-11-11 18:10:09 UTC

## R topics documented:

DEoptim *Differential Evolution Optimization*

## Description

Performs evolutionary global optimization via the Differential Evolution algorithm.

## Usage

```
DEoptim(fn, lower, upper, control = DEoptim.control(), ..., fnMap=NULL)
```

## Arguments

| | |
|---|---|
| fn | the function to be optimized (minimized). The function should have as its first argument the vector of real-valued parameters to optimize, and return a scalar real result. NA and NaN values are not allowed. |
| lower, upper | two vectors specifying scalar real lower and upper bounds on each parameter to be optimized, so that the i-th element of lower and upper applies to the i-th parameter. The implementation searches between lower and upper for the global optimum (minimum) of fn. |
| control | a list of control parameters; see DEoptim.control. |
| fnMap | an optional function that will be run after each population is created, but before the population is passed to the objective function. This allows the user to impose integer/cardinality constraints. See the the sandbox directory of the source code for a simple example. |
| ... | further arguments to be passed to fn. |

## Details

DEoptim performs optimization (minimization) of fn.

The control argument is a list; see the help file for DEoptim.control for details.

The R implementation of Differential Evolution (DE), **DEoptim**, was first published on the Comprehensive R Archive Network (CRAN) in 2005 by David Ardia. Early versions were written in pure R. Since version 2.0-0 (published to CRAN in 2009) the package has relied on an interface to a C implementation of DE, which is significantly faster on most problems as compared to the implementation in pure R. The C interface is in many respects similar to the MS Visual C++ v5.0 implementation of the Differential Evolution algorithm distributed with the book *Differential Evolution – A Practical Approach to Global Optimization* by Price, K.V., Storn, R.M., Lampinen J.A, Springer-Verlag, 2006. Since version 2.0-3 the C implementation dynamically allocates the memory required to store the population, removing limitations on the number of members in the population and length of the parameter vectors that may be optimized. Since version 2.2-0, the package allows for parallel operation, so that the evaluations of the objective function may be performed using all available cores. This is accomplished using either the built-in **parallel** package or the **foreach** package. If parallel operation is desired, the user should set parallelType and make sure that the

arguments and packages needed by the objective function are available; see DEoptim.control, the example below and examples in the sandbox directory for details.

Since becoming publicly available, the package **DEoptim** has been used by several authors to solve optimization problems arising in diverse domains; see Mullen et al. (2011) for a review.

To perform a maximization (instead of minimization) of a given function, simply define a new function which is the opposite of the function to maximize and apply DEoptim to it.

To integrate additional constraints (other than box constraints) on the parameters x of fn(x), for instance x[1] + x[2]^2 < 2, integrate the constraint within the function to optimize, for instance:

```
fn <- function(x){
  if (x[1] + x[2]^2 >= 2){
    r <- Inf
  else{
    ...
  }
  return(r)
}
```

This simplistic strategy usually does not work all that well for gradient-based or Newton-type methods. It is likely to be alright when the solution is in the interior of the feasible region, but when the solution is on the boundary, optimization algorithm would have a difficult time converging. Furthermore, when the solution is on the boundary, this strategy would make the algorithm converge to an inferior solution in the interior. However, for methods such as DE which are not gradient based, this strategy might not be that bad.

Note that DEoptim stops if any NA or NaN value is obtained. You have to redefine your function to handle these values (for instance, set NA to Inf in your objective function).

It is important to emphasize that the result of DEoptim is a random variable, i.e., different results may be obtained when the algorithm is run repeatedly with the same settings. Hence, the user should set the random seed if they want to reproduce the results, e.g., by setting set.seed(1234) before the call of DEoptim.

DEoptim relies on repeated evaluation of the objective function in order to move the population toward a global minimum. Users interested in making DEoptim run as fast as possible should consider using the package in parallel mode (so that all CPU's available are used), and also ensure that evaluation of the objective function is as efficient as possible (e.g. by using vectorization in pure R code, or writing parts of the objective function in a lower-level language like C or Fortran).

Further details and examples of the R package **DEoptim** can be found in Mullen et al. (2011) and Ardia et al. (2011a, 2011b) or look at the package's vignette by typing vignette("DEoptim"). Also, an illustration of the package usage for a high-dimensional non-linear portfolio optimization problem is available by typing vignette("DEoptimPortfolioOptimization").

Please cite the package in publications. Use citation("DEoptim").

## Value

The output of the function DEoptim is a member of the S3 class DEoptim. More precisely, this is a list (of length 2) containing the following elements:

`optim`, a list containing the following elements:

- `bestmem`: the best set of parameters found.
- `bestval`: the value of `fn` corresponding to `bestmem`.
- `nfeval`: number of function evaluations.
- `iter`: number of procedure iterations.

`member`, a list containing the following elements:

- `lower`: the lower boundary.
- `upper`: the upper boundary.
- `bestvalit`: the best value of `fn` at each iteration.
- `bestmemit`: the best member at each iteration.
- `pop`: the population generated at the last iteration.
- `storepop`: a list containing the intermediate populations.

Members of the class `DEoptim` have a `plot` method that accepts the argument `plot.type`. `plot.type = "bestmemit"` results in a plot of the parameter values that represent the lowest value of the objective function each generation. `plot.type = "bestvalit"` plots the best value of the objective function each generation. Finally, `plot.type = "storepop"` results in a plot of stored populations (which are only available if these have been saved by setting the `control` argument of `DEoptim` appropriately). Storing intermediate populations allows us to examine the progress of the optimization in detail. A summary method also exists and returns the best parameter vector, the best value of the objective function, the number of generations optimization ran, and the number of times the objective function was evaluated.

## Note

*Differential Evolution* (DE) is a search heuristic introduced by Storn and Price (1997). Its remarkable performance as a global optimization algorithm on continuous numerical minimization problems has been extensively explored; see Price et al. (2006). DE belongs to the class of genetic algorithms which use biology-inspired operations of crossover, mutation, and selection on a population in order to minimize an objective function over the course of successive generations (see Mitchell, 1998). As with other evolutionary algorithms, DE solves optimization problems by evolving a population of candidate solutions using alteration and selection operators. DE uses floating-point instead of bit-string encoding of population members, and arithmetic operations instead of logical operations in mutation. DE is particularly well-suited to find the global optimum of a real-valued function of real-valued parameters, and does not require that the function be either continuous or differentiable.

Let $NP$ denote the number of parameter vectors (members) $x \in R^d$ in the population. In order to create the initial generation, $NP$ guesses for the optimal value of the parameter vector are made, either using random values between lower and upper bounds (defined by the user) or using values given by the user. Each generation involves creation of a new population from the current population members $\{x_i \,|\, i = 1, \ldots, NP\}$, where $i$ indexes the vectors that make up the population. This is accomplished using *differential mutation* of the population members. An initial mutant parameter vector $v_i$ is created by choosing three members of the population, $x_{r_0}$, $x_{r_1}$ and $x_{r_2}$, at random. Then $v_i$ is generated as

$$v_i \doteq x_{r_0} + F \cdot (x_{r_1} - x_{r_2})$$

where $F$ is the differential weighting factor, effective values for which are typically between 0 and 1. After the first mutation operation, mutation is continued until $d$ mutations have been made, with a crossover probability $CR \in [0, 1]$. The crossover probability $CR$ controls the fraction of the parameter values that are copied from the mutant. If an element of the trial parameter vector is found to violate the bounds after mutation and crossover, it is reset in such a way that the bounds are respected (with the specific protocol depending on the implementation). Then, the objective function values associated with the children are determined. If a trial vector has equal or lower objective function value than the previous vector it replaces the previous vector in the population; otherwise the previous vector remains. Variations of this scheme have also been proposed; see Price et al. (2006) and `DEoptim.control`.

Intuitively, the effect of the scheme is that the shape of the distribution of the population in the search space is converging with respect to size and direction towards areas with high fitness. The closer the population gets to the global optimum, the more the distribution will shrink and therefore reinforce the generation of smaller difference vectors.

As a general advice regarding the choice of $NP$, $F$ and $CR$, Storn et al. (2006) state the following: Set the number of parents $NP$ to 10 times the number of parameters, select differential weighting factor $F = 0.8$ and crossover constant $CR = 0.9$. Make sure that you initialize your parameter vectors by exploiting their full numerical range, i.e., if a parameter is allowed to exhibit values in the range [-100, 100] it is a good idea to pick the initial values from this range instead of unnecessarily restricting diversity. If you experience misconvergence in the optimization process you usually have to increase the value for $NP$, but often you only have to adjust $F$ to be a little lower or higher than 0.8. If you increase $NP$ and simultaneously lower $F$ a little, convergence is more likely to occur but generally takes longer, i.e., DE is getting more robust (there is always a convergence speed/robustness trade-off).

DE is much more sensitive to the choice of $F$ than it is to the choice of $CR$. $CR$ is more like a fine tuning element. High values of $CR$ like $CR = 1$ give faster convergence if convergence occurs. Sometimes, however, you have to go down as much as $CR = 0$ to make DE robust enough for a particular problem. For more details on the DE strategy, we refer the reader to Storn and Price (1997) and Price et al. (2006).

## Author(s)

David Ardia, Katharine Mullen <mullenkate@gmail.com>, Brian Peterson and Joshua Ulrich.

## References

Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2011) Differential Evolution with **DEoptim**. An Application to Non-Convex Portfolio Optimization. *R Journal*, 3(1), 27-34. doi:10.32614/RJ2011005

Ardia, D., Ospina Arango, J.D., Giraldo Gomez, N.D. (2011) Jump-Diffusion Calibration using Differential Evolution. *Wilmott Magazine*, 55 (September), 76-79. doi:10.1002/wilm.10034

Mitchell, M. (1998) *An Introduction to Genetic Algorithms*. The MIT Press. ISBN 0262631857.

Mullen, K.M, Ardia, D., Gil, D., Windover, D., Cline,J. (2011). **DEoptim:** An R Package for Global Optimization by Differential Evolution. *Journal of Statistical Software*, 40(6), 1-26. doi:10.18637/jss.v040.i06

Price, K.V., Storn, R.M., Lampinen J.A. (2006) *Differential Evolution - A Practical Approach to Global Optimization*. Berlin Heidelberg: Springer-Verlag. ISBN 3540209506.

Storn, R. and Price, K. (1997) Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization*, 11:4, 341–359.

## See Also

`DEoptim.control` for control arguments, `DEoptim-methods` for methods on `DEoptim` objects, including some examples in plotting the results; `optim` or `constrOptim` for alternative optimization algorithms.

## Examples

```
## Rosenbrock Banana function
## The function has a global minimum f(x) = 0 at the point (1,1).
## Note that the vector of parameters to be optimized must be the first
## argument of the objective function passed to DEoptim.
Rosenbrock <- function(x){
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}

## DEoptim searches for minima of the objective function between
## lower and upper bounds on each parameter to be optimized. Therefore
## in the call to DEoptim we specify vectors that comprise the
## lower and upper bounds; these vectors are the same length as the
## parameter vector.
lower <- c(-10,-10)
upper <- -lower

## run DEoptim and set a seed first for replicability
set.seed(1234)
DEoptim(Rosenbrock, lower, upper)

## increase the population size
DEoptim(Rosenbrock, lower, upper, DEoptim.control(NP = 100))

## change other settings and store the output
outDEoptim <- DEoptim(Rosenbrock, lower, upper, DEoptim.control(NP = 80,
                      itermax = 400, F = 1.2, CR = 0.7))

## plot the output
plot(outDEoptim)

## 'Wild' function, global minimum at about -15.81515
Wild <- function(x)
  10 * sin(0.3 * x) * sin(1.3 * x^2) +
     0.00001 * x^4 + 0.2 * x + 80

plot(Wild, -50, 50, n = 1000, main = "'Wild function'")
```

```
outDEoptim <- DEoptim(Wild, lower = -50, upper = 50,
                      control = DEoptim.control(trace = FALSE))

plot(outDEoptim)

DEoptim(Wild, lower = -50, upper = 50,
        control = DEoptim.control(NP = 50))

## The below examples shows how the call to DEoptim can be
## parallelized.
## Note that if your objective function requires packages to be
## loaded or has arguments supplied via \code{...}, these should be
## specified using the \code{packages} and \code{parVar} arguments
## in control.
## Not run:

Genrose <- function(x) {
   ## One generalization of the Rosenbrock banana valley function (n parameters)
   n <- length(x)
   ## make it take some time ...
   Sys.sleep(.001)
   1.0 + sum (100 * (x[-n]^2 - x[-1])^2 + (x[-1] - 1)^2)
}

# get some run-time on simple problems
maxIt <- 250
n <- 5

oneCore <- system.time( DEoptim(fn=Genrose, lower=rep(-25, n), upper=rep(25, n),
                 control=list(NP=10*n, itermax=maxIt)))

withParallel <-  system.time( DEoptim(fn=Genrose, lower=rep(-25, n), upper=rep(25, n),
                 control=list(NP=10*n, itermax=maxIt, parallelType=1)))

## Compare timings
(oneCore)
(withParallel)

## End(Not run)
```

---

DEoptim-methods *DEoptim-methods*

---

## Description

Methods for DEoptim objects.

## Usage

```
## S3 method for class 'DEoptim'
```

```
summary(object, ...)
## S3 method for class 'DEoptim'
plot(x, plot.type = c("bestmemit", "bestvalit", "storepop"), ...)
```

## Arguments

| | |
|---|---|
| object | an object of class DEoptim; usually, a result of a call to DEoptim. |
| x | an object of class DEoptim; usually, a result of a call to DEoptim. |
| plot.type | should we plot the best member at each iteration, the best value at each iteration or the intermediate populations? |
| ... | further arguments passed to or from other methods. |

## Details

Members of the class DEoptim have a plot method that accepts the argument plot.type. plot.type = "bestmemit" results in a plot of the parameter values that represent the lowest value of the objective function each generation. plot.type = "bestvalit" plots the best value of the objective function each generation. Finally, plot.type = "storepop" results in a plot of stored populations (which are only available if these have been saved by setting the control argument of DEoptim appropriately). Storing intermediate populations allows us to examine the progress of the optimization in detail. A summary method also exists and returns the best parameter vector, the best value of the objective function, the number of generations optimization ran, and the number of times the objective function was evaluated.

## Note

Further details and examples of the R package **DEoptim** can be found in Mullen et al. (2011) and Ardia et al. (2011a, 2011b) or look at the package's vignette by typing vignette("DEoptim").

Please cite the package in publications. Use citation("DEoptim").

## Author(s)

David Ardia, Katharine Mullen <mullenkate@gmail.com>, Brian Peterson and Joshua Ulrich.

## References

Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2011) Differential Evolution with **DEoptim**. An Application to Non-Convex Portfolio Optimization. *R Journal*, 3(1), 27-34. doi:10.32614/RJ2011005

Ardia, D., Ospina Arango, J.D., Giraldo Gomez, N.D. (2011) Jump-Diffusion Calibration using Differential Evolution. *Wilmott Magazine*, 55 (September), 76-79. doi:10.1002/wilm.10034

Mullen, K.M, Ardia, D., Gil, D., Windover, D., Cline,J. (2011). **DEoptim:** An R Package for Global Optimization by Differential Evolution. *Journal of Statistical Software*, 40(6), 1-26. doi:10.18637/jss.v040.i06

## See Also

DEoptim and DEoptim.control.

## Examples

```
## Rosenbrock Banana function
## The function has a global minimum f(x) = 0 at the point (1,1).
## Note that the vector of parameters to be optimized must be the first
## argument of the objective function passed to DEoptim.
Rosenbrock <- function(x){
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}

lower <- c(-10, -10)
upper <- -lower

set.seed(1234)
outDEoptim <- DEoptim(Rosenbrock, lower, upper)

## print output information
summary(outDEoptim)

## plot the best members
plot(outDEoptim, type = 'b')

## plot the best values
dev.new()
plot(outDEoptim, plot.type = "bestvalit", type = 'b', col = 'blue')

## rerun the optimization, and store intermediate populations
outDEoptim <- DEoptim(Rosenbrock, lower, upper,
                      DEoptim.control(itermax = 500,
                      storepopfrom = 1, storepopfreq = 2))
summary(outDEoptim)

## plot intermediate populations
dev.new()
plot(outDEoptim, plot.type = "storepop")

## Wild function
Wild <- function(x)
  10 * sin(0.3 * x) * sin(1.3 * x^2) +
     0.00001 * x^4 + 0.2 * x + 80

outDEoptim = DEoptim(Wild, lower = -50, upper = 50,
                     DEoptim.control(trace = FALSE, storepopfrom = 50,
                     storepopfreq = 1))

plot(outDEoptim, type = 'b')

dev.new()
plot(outDEoptim, plot.type = "bestvalit", type = 'b')

## Not run:
```

```
## an example with a normal mixture model: requires package mvtnorm
library(mvtnorm)

## neg value of the density function
negPdfMix <- function(x) {
   tmp <- 0.5 * dmvnorm(x, c(-3, -3)) + 0.5 * dmvnorm(x, c(3, 3))
   -tmp
}

## wrapper plotting function
plotNegPdfMix <- function(x1, x2)
   negPdfMix(cbind(x1, x2))

## contour plot of the mixture
x1 <- x2 <- seq(from = -10.0, to = 10.0, by = 0.1)
thexlim <- theylim <- range(x1)
z <- outer(x1, x2, FUN = plotNegPdfMix)

contour(x1, x2, z, nlevel = 20, las = 1, col = rainbow(20),
   xlim = thexlim, ylim = theylim)

set.seed(1234)
outDEoptim <- DEoptim(negPdfMix, c(-10, -10), c(10, 10),
   DEoptim.control(NP = 100, itermax = 100, storepopfrom = 1,
   storepopfreq = 5))

## convergence plot
dev.new()
plot(outDEoptim)

## the intermediate populations indicate the bi-modality of the function
dev.new()
plot(outDEoptim, plot.type = "storepop")

## End(Not run)
```

---

`DEoptim.control` *Control various aspects of the DEoptim implementation*

---

## Description

Allow the user to set some characteristics of the Differential Evolution optimization algorithm implemented in `DEoptim`.

## Usage

```
DEoptim.control(VTR = -Inf, strategy = 2, bs = FALSE, NP = NA,
  itermax = 200, CR = 0.5, F = 0.8, trace = TRUE, initialpop = NULL,
  storepopfrom = itermax + 1, storepopfreq = 1, p = 0.2, c = 0, reltol,
  steptol, parallelType = c("none", "auto", "parallel", "foreach"),
```

```
  cluster = NULL, packages = c(), parVar = c(),
  foreachArgs = list(), parallelArgs = NULL)
```

## Arguments

| | |
|---|---|
| VTR | the value to be reached. The optimization process will stop if either the maximum number of iterations itermax is reached or the best parameter vector bestmem has found a value fn(bestmem) <= VTR. Default to -Inf. |
| strategy | defines the Differential Evolution strategy used in the optimization procedure:<br>1: DE / rand / 1 / bin (classical strategy)<br>2: DE / local-to-best / 1 / bin (default)<br>3: DE / best / 1 / bin with jitter<br>4: DE / rand / 1 / bin with per-vector-dither<br>5: DE / rand / 1 / bin with per-generation-dither<br>6: DE / current-to-p-best / 1<br>any value not above: variation to DE / rand / 1 / bin: either-or-algorithm. Default strategy is currently 2. See *Details*. |
| bs | if FALSE then every mutant will be tested against a member in the previous generation, and the best value will proceed into the next generation (this is standard trial vs. target selection). If TRUE then the old generation and NP mutants will be sorted by their associated objective function values, and the best NP vectors will proceed into the next generation (best of parent and child selection). Default is FALSE. |
| NP | number of population members. Defaults to NA; if the user does not change the value of NP from NA or specifies a value less than 4 it is reset when DEoptim is called as 10*length(lower). For many problems it is best to set NP to be at least 10 times the length of the parameter vector. |
| itermax | the maximum iteration (population generation) allowed. Default is 200. |
| CR | crossover probability from interval [0,1]. Default to 0.5. |
| F | differential weighting factor from interval [0,2]. Default to 0.8. |
| trace | Positive integer or logical value indicating whether printing of progress occurs at each iteration. The default value is TRUE. If a positive integer is specified, printing occurs every trace iterations. |
| initialpop | an initial population used as a starting population in the optimization procedure. May be useful to speed up the convergence. Default to NULL. If given, each member of the initial population should be given as a row of a numeric matrix, so that initialpop is a matrix with NP rows and a number of columns equal to the length of the parameter vector to be optimized. |
| storepopfrom | from which generation should the following intermediate populations be stored in memory. Default to itermax + 1, i.e., no intermediate population is stored. |
| storepopfreq | the frequency with which populations are stored. Default to 1, i.e., every intermediate population is stored. |
| p | when strategy = 6, the top (100 * p)% best solutions are used in the mutation. p must be defined in (0,1]. |
| c | c controls the speed of the crossover adaptation. Higher values of c give more weight to the current successful mutations. c must be defined in (0,1]. |

| | |
|---|---|
| reltol | relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of reltol * (abs(val) + reltol) after steptol steps. Defaults to sqrt(.Machine$double.eps), typically about 1e-8. |
| steptol | see reltol. Defaults to itermax. |
| parallelType | Defines the type of parallelization to employ, if any. none: The default, this uses DEoptim on only one core. auto: will attempt to auto-detect foreach, or parallel. parallel: This uses all available cores, via the **parallel** package, to run DEoptim. foreach: This uses the **foreach** package for parallelism; see the sandbox directory in the source code for examples. |
| cluster | Existing **parallel** cluster object. If provided, overrides + specified parallelType. Using cluster allows fine-grained control + over the number of used cores and exported data. |
| packages | Used if parallelType='parallel'; a list of package names (as strings) that need to be loaded for use by the objective function. |
| parVar | Used if parallelType='parallel'; a list of variable names (as strings) that need to exist in the environment for use by the objective function or are used as arguments by the objective function. |
| foreachArgs | A list of named arguments for the foreach function from the package **foreach**. The arguments i, .combine and .export are not possible to set here; they are set internally. |
| parallelArgs | A list of named arguments for the parallel engine. For package **foreach**, the argument i is not possible to set here; it is set internally. |

## Details

This defines the Differential Evolution strategy used in the optimization procedure, described below in the terms used by Price et al. (2006); see also Mullen et al. (2009) for details.

- strategy = 1: DE / rand / 1 / bin.
  This strategy is the classical approach for DE, and is described in DEoptim.
- strategy = 2: DE / local-to-best / 1 / bin.
  In place of the classical DE mutation the expression

$$v_{i,g} = old_{i,g} + (best_g - old_{i,g}) + x_{r0,g} + F \cdot (x_{r1,g} - x_{r2,g})$$

  is used, where $old_{i,g}$ and $best_g$ are the $i$-th member and best member, respectively, of the previous population. This strategy is currently used by default.
- strategy = 3: DE / best / 1 / bin with jitter.
  In place of the classical DE mutation the expression

$$v_{i,g} = best_g + jitter + F \cdot (x_{r1,g} - x_{r2,g})$$

  is used, where $jitter$ is defined as 0.0001 * rand + F.
- strategy = 4: DE / rand / 1 / bin with per vector dither.
  In place of the classical DE mutation the expression

$$v_{i,g} = x_{r0,g} + dither \cdot (x_{r1,g} - x_{r2,g})$$

  is used, where $dither$ is calculated as $F + \text{rand} * (1 - F)$.

- `strategy = 5`: DE / rand / 1 / bin with per generation dither.
  The strategy described for 4 is used, but $dither$ is only determined once per-generation.
- `strategy = 6`: DE / current-to-p-best / 1.
  The top $(100 * p)$ percent best solutions are used in the mutation, where $p$ is defined in $(0, 1]$.
- any value not above: variation to DE / rand / 1 / bin: either-or algorithm.
  In the case that `rand` < 0.5, the classical strategy `strategy = 1` is used. Otherwise, the expression

  $$v_{i,g} = x_{r0,g} + 0.5 \cdot (F + 1) \cdot (x_{r1,g} + x_{r2,g} - 2 \cdot x_{r0,g})$$

  is used.

Several conditions can cause the optimization process to stop:

- if the best parameter vector (`bestmem`) produces a value less than or equal to `VTR` (i.e. `fn(bestmem) <= VTR`), or
- if the maximum number of iterations is reached (`itermax`), or
- if a number (`steptol`) of consecutive iterations are unable to reduce the best function value by a certain amount (`reltol * (abs(val) + reltol)`). `100*reltol` is approximately the percent change of the objective value required to consider the parameter set an improvement over the current best member.

Zhang and Sanderson (2009) define several extensions to the DE algorithm, including strategy 6, DE/current-to-p-best/1. They also define a self-adaptive mechanism for the other control parameters. This self-adaptation will speed convergence on many problems, and is defined by the control parameter `c`. If `c` is non-zero, crossover and mutation will be adapted by the algorithm. Values in the range of `c=.05` to `c=.5` appear to work best for most problems, though the adaptive algorithm is robust to a wide range of `c`.

## Value

The default value of `control` is the return value of `DEoptim.control()`, which is a list (and a member of the `S3` class `DEoptim.control`) with the above elements.

## Note

Further details and examples of the R package **DEoptim** can be found in Mullen et al. (2011) and Ardia et al. (2011a, 2011b) or look at the package's vignette by typing `vignette("DEoptim")`. Also, an illustration of the package usage for a high-dimensional non-linear portfolio optimization problem is available by typing `vignette("DEoptimPortfolioOptimization")`.

Please cite the package in publications. Use `citation("DEoptim")`.

## Author(s)

David Ardia, Katharine Mullen <mullenkate@gmail.com>, Brian Peterson and Joshua Ulrich.

## References

Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2011) Differential Evolution with **DEoptim**. An Application to Non-Convex Portfolio Optimization. *R Journal*, 3(1), 27-34. doi:10.32614/RJ2011005

Ardia, D., Ospina Arango, J.D., Giraldo Gomez, N.D. (2011) Jump-Diffusion Calibration using Differential Evolution. *Wilmott Magazine*, 55 (September), 76-79. doi:10.1002/wilm.10034

Mullen, K.M, Ardia, D., Gil, D., Windover, D., Cline,J. (2011). **DEoptim:** An R Package for Global Optimization by Differential Evolution. *Journal of Statistical Software*, 40(6), 1-26. doi:10.18637/jss.v040.i06

Price, K.V., Storn, R.M., Lampinen J.A. (2006) *Differential Evolution - A Practical Approach to Global Optimization*. Berlin Heidelberg: Springer-Verlag. ISBN 3540209506.

Zhang, J. and Sanderson, A. (2009) *Adaptive Differential Evolution* Springer-Verlag. ISBN 978-3-642-01526-7

## See Also

DEoptim and DEoptim-methods.

## Examples

```
## set the population size to 20
DEoptim.control(NP = 20)

## set the population size, the number of iterations and don't
## display the iterations during optimization
DEoptim.control(NP = 20, itermax = 100, trace = FALSE)
```

---

# SMI *Swiss Market Index data*

---

## Description

See Mullen et al. (2011) for description of this dataset.

## Usage

```
data("SMI")
```

## References

Mullen, K.M, Ardia, D., Gil, D., Windover, D., Cline,J. (2011). **DEoptim:** An R Package for Global Optimization by Differential Evolution. *Journal of Statistical Software*, 40(6), 1-26. doi:10.18637/jss.v040.i06

`xrrData` *X-ray reflectometry data*

## Description

See Mullen et al. (2011) for description of this dataset.

## Usage

```
data("xrrData")
```

## References

Mullen, K.M, Ardia, D., Gil, D., Windover, D., Cline,J. (2011). **DEoptim:** An R Package for Global Optimization by Differential Evolution. *Journal of Statistical Software*, 40(6), 1-26. doi:10.18637/jss.v040.i06

# Index