Package ‘DEoptimR’

July 12, 2023

Version 1.1-0
Date 2023-07-10
Title Differential Evolution Optimization in Pure R

Description Differential Evolution (DE) stochastic heuristic algorithms for
global optimization of problems with and without general constraints.
The aim is to curate a collection of its variants that
(1) do not sacrifice simplicity of design,
(2) are essentially tuning-free, and
(3) can be efficiently implemented directly in the R language.
Currently, it provides implementations of the algorithms ‘jDE’ by
Brest et al. (2006) <doi:10.1109/TEVC.2006.872133> for single-objective
optimization and ‘NCDE’ by Qu et al. (2012) <doi:10.1109/TEVC.2011.2161873>
for multimodal optimization.

Imports stats
Enhances robustbase
License GPL (>= 2)
Author Eduardo L. T. Conceicao [aut, cre],
Martin Maechler [ctb] (<https://orcid.org/0000-0002-8685-9910>)
Maintainer Eduardo L. T. Conceicao <mail@eduardoconceicao.org>
Repository CRAN
Repository/R-Forge/Project robustbase
Repository/R-Forge/Revision 951
Repository/R-Forge/DateTimeStamp 2023-07-10 19:56:11
Date/Publication 2023-07-12 18:30:32 UTC

R topics documented:

    JDEoptim ................................................................. 2
    NCDEoptim .............................................................. 8

Index 13
JDEoptim

Bound-Constrained and Nonlinear Constrained Single-Objective Optimization via Differential Evolution

Description

A bespoke implementation of the ‘jDE’ variant by Brest et al. (2006) doi:10.1109/TEVC.2006.872133.

Usage

JDEoptim(lower, upper, fn,
    constr = NULL, meq = 0, eps = 1e-05,
    NP = 10*length(lower), FI = 0.1, Fu = 1,
    tau_F = 0.1, tau_CR = 0.1, tau_pF = 0.1,
    jitter_factor = 0.001,
    tol = 1e-15, maxiter = 200*length(lower), fnscale = 1,
    compare_to = c("median", "max"),
    add_to_init_pop = NULL,
    trace = FALSE, triter = 1,
    details = FALSE, ...)

Arguments

lower, upper numeric vectors of lower and upper bounds, respectively, for the parameters to be optimized over. Must be finite (is.finite) as they bound the hyper-rectangle of the initial random population.

fn (nonlinear) objective function to be minimized. It takes as first argument the vector of parameters over which minimization is to take place. It must return the value of the function at that point.

constr an optional function for specifying the nonlinear constraints under which we want to minimize fn. Nonlinear equalities should be given first and defined to equal zero \( h_j(X) = 0 \), followed by nonlinear inequalities defined as lesser than zero \( g_i(X) \leq 0 \). This function takes the vector of parameters as its first argument and returns a real vector with the length of the total number of constraints. It defaults to NULL, meaning that bound-constrained minimization is used.

meq an optional positive integer specifying that the first meq constraints are treated as equality constraints, and all the remaining as inequality constraints. Defaults to 0 (inequality constraints only).

eps maximal admissible constraint violation for equality constraints. An optional real vector of small positive tolerance values with length meq used in the transformation of equalities into inequalities of the form \( |h_j(X)| - \epsilon \leq 0 \). A scalar value is expanded to apply to all equality constraints. Default is 1e-5.

NP an optional positive integer giving the number of candidate solutions in the randomly distributed initial population. Defaults to 10*length(lower).
JDEoptim

JDEoptim is a software tool for the optimization of functions using the Differential Evolution (DE) algorithm. It provides a simple, self-adaptive scheme to set the control parameters automatically, which is crucial for the algorithm's performance. The setting of the control parameters of standard Differential Evolution (DE) is crucial for the algorithm’s performance. Unfortunately, when the generally recommended values for these parameters (see, e.g., Storn and Price, 1997) are unsuitable for use, their determination is often difficult and time consuming. The jDE algorithm proposed in Brest et al. (2006) employs a simple self-adaptive scheme to perform the automatic setting of control parameters scale factor $F$ and crossover rate $CR$.

### Parameters

- **F_l** an optional scalar which represents the minimum value that the scaling factor $F$ could take. Default is $0.1$, which is almost always satisfactory.
- **F_u** an optional scalar which represents the maximum value that the scaling factor $F$ could take. Default is $1$, which is almost always satisfactory.
- **tau_F** an optional scalar which represents the probability that the scaling factor $F$ is updated.Defaults to $0.1$, which is almost always satisfactory.
- **tau_CR** an optional constant value which represents the probability that the crossover probability $CR$ is updated. Defaults to $0.1$, which is almost always satisfactory.
- **tau_pF** an optional scalar which represents the probability that the mutation probability $p_F$ in the mutation strategy DE/rand/1/either-or is updated. Defaults to $0.1$.
- **jitter_factor** an optional tuning constant for jitter. If NULL only dither is used. Defaults to $0.001$.
- **tol** an optional positive scalar giving the tolerance for the stopping criterion. Default is $1e-15$.
- **maxiter** an optional positive integer specifying the maximum number of iterations that may be performed before the algorithm is halted. Defaults to $200\times\text{length}(\text{lower})$.
- **fnscale** an optional positive scalar specifying the typical magnitude of $fn$. It is used only in the stopping criterion. Defaults to $1$. See ‘Details’.
- **compare_to** an optional character string controlling which function should be applied to the $fn$ values of the candidate solutions in a generation to be compared with the so-far best one when evaluating the stopping criterion. If “median” the median function is used; else, if “max” the max function is used. It defaults to “median”. See ‘Details’.
- **add_to_init_pop** an optional real vector of length $\text{length}(\text{lower})$ or matrix with $\text{length}(\text{lower})$ rows specifying initial values of the parameters to be optimized which are appended to the randomly generated initial population. It defaults to NULL.
- **trace** an optional logical value indicating if a trace of the iteration progress should be printed. Default is FALSE.
- **triter** an optional positive integer that controls the frequency of tracing when trace = TRUE. Default is triter = 1, which means that iteration : $<\text{value of stopping test}>\{\text{value of best solution}\}$ best solution { index of violated constraints } is printed at every iteration.
- **details** an optional logical value. If TRUE the output will contain the parameters in the final population and their respective $fn$ values. Defaults to FALSE.
- **...** optional additional arguments passed to fn() and constr() if that is not NULL.

### Details

**Overview:** The setting of the control parameters of standard Differential Evolution (DE) is crucial for the algorithm’s performance. Unfortunately, when the generally recommended values for these parameters (see, e.g., Storn and Price, 1997) are unsuitable for use, their determination is often difficult and time consuming. The jDE algorithm proposed in Brest et al. (2006) employs a simple self-adaptive scheme to perform the automatic setting of control parameters scale factor $F$ and crossover rate $CR$. 
This implementation differs from the original description, most notably in the use of the DE/rand/1/either-or mutation strategy (Price et al., 2005), and combination of jitter with dither (Storn, 2008).

As done by jDE and its variants (Brest et al., 2021) each worse parent in the current population is immediately replaced (asynchronous update) by its newly generated better or equal offspring (Babu and Angira, 2006) instead of updating the current population with all the new solutions at the same time as in classical DE (synchronous update).

As the algorithm subsamples via sample() which from R version 3.6.0 depends on RNGkind(*, sample.kind), exact reproducibility of results from R versions 3.5.3 and earlier requires setting RNGversion("3.5.0"). In any case, do use set.seed() additionally for reproducibility!

**Constraint Handling:** Constraint handling is done using the approach described in Zhang and Rangaiah (2012), but with a different reduction updating scheme for the constraint relaxation value ($\mu$). Instead of doing it once for every generation or iteration, the reduction is triggered for two cases when the constraints only contain inequalities. Firstly, every time a feasible solution is selected for replacement in the next generation by a new feasible trial candidate solution with a better objective function value. Secondly, whenever a current infeasible solution gets replaced by a feasible one. If the constraints include equalities, then the reduction is not triggered in this last case. This constitutes an original feature of the implementation.

The performance of the constraint handling technique is severely impaired by a small feasible region. Therefore, equality constraints are particularly difficult to handle due to the tiny feasible region they define. So, instead of explicitly including all equality constraints in the formulation of the optimization problem, it might prove advantageous to eliminate some of them. This is done by expressing one variable $x_k$ in terms of the remaining others for an equality constraint $h_j (X) = 0$ where $X = [x_1, \ldots, x_k, \ldots, x_d]$ is the vector of solutions, thereby obtaining a relationship as $x_k = R_{k,j}([x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_d])$. But this means that both the variable $x_k$ and the equality constraint $h_j(X) = 0$ can be removed altogether from the original optimization formulation, since the value of $x_k$ can be calculated during the search process by the relationship $R_{k,j}$. Notice, however, that two additional inequalities

$$l_k \leq R_{k,j}([x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_d]) \leq u_k,$$

where the values $l_k$ and $u_k$ are the lower and upper bounds of $x_k$, respectively, must be provided in order to obtain an equivalent formulation of the problem. For guidance and examples on applying this approach see Wu et al. (2015).

**Discrete and Integer Variables:** Any DE variant is easily extended to deal with mixed integer non-linear programming problems using a small variation of the technique presented by Lampinen and Zelinka (1999). Integer values are obtained by means of the floor() function only for the evaluation of the objective function. This is because DE itself works with continuous variables. Additionally, each upper bound of the integer variables should be added by 1. Notice that the final solution needs to be converted with floor() to obtain its integer elements.

**Stopping Criterion:** The algorithm is stopped if

$$\frac{\text{compare}_\text{to}([\text{fn}(X_1), \ldots, \text{fn}(X_{\text{npop}})]) - \text{fn}(X_{\text{best}})}{\text{fnscale}} \leq \text{tol}$$

where the “best” individual $X_{\text{best}}$ is the feasible solution with the lowest objective function value in the population and the total number of elements in the population, npop, is $\text{NP}+\text{NCOL}(\text{add} \_ \text{to} \_ \text{init} \_ \text{pop})$. This is a variant of the Diff criterion studied by Zielinski and Laur (2008), which was found to yield the best results.
Value

A list with the following components:

- **par**: The best set of parameters found.
- **value**: The value of `fn` corresponding to `par`.
- **iter**: Number of iterations taken by the algorithm.
- **convergence**: An integer code. 0 indicates successful completion. 1 indicates that the iteration limit `maxiter` has been reached.

And if `details = TRUE`:

- **poppar**: Matrix of dimension `c(length(lower), npop)`, with columns corresponding to the parameter vectors remaining in the population.
- **popcost**: The values of `fn` associated with `poppar`, vector of length `npop`.

Note

It is possible to perform a warm start, i.e., starting from the previous run and resume optimization, using `NP = 0` and the component `poppar` for the `add_to_init_pop` argument.

Author(s)

Eduardo L. T. Conceicao <mail@eduardoconceicao.org>

References


See Also

Function *DEoptim()* in the *DEoptim* package has many more options than *JDEoptim()* but does not allow constraints in the same flexible manner.

Examples

```r
# NOTE: Examples were excluded from testing
# to reduce package check time.

# Use a preset seed so test values are reproducible.
set.seed(1234)

# Bound-constrained optimization

# Griewank function
#
# -600 <= x_i <= 600, i = {1, 2, ..., n}
# The function has a global minimum located at
# xstar = (0, 0, ..., 0) with f(xstar) = 0. Number of local minima
# for arbitrary n is unknown, but in the two dimensional case
# there are some 500 local minima.
#
# Source:
# Ali, M. Montaz, Khompatraporn, Charoenchai, and
# A numerical evaluation of several stochastic algorithms
# on selected continuous global optimization test problems.
# Journal of Global Optimization 31, 635-672.
# https://doi.org/10.1007/s10898-004-9972-2

griewank <- function(x) {
  1 + crossprod(x)/4000 - prod( cos(x/sqrt(seq_along(x))) )
}

JDEoptim(rep(-600, 10), rep(600, 10), griewank,
  tol = 1e-7, trace = TRUE, triter = 50)

# Nonlinear constrained optimization

# 0 <= x1 <= 34, 0 <= x2 <= 17, 100 <= x3 <= 300
```
The global optimum is
(x1, x2, x3; f) = (0, 16.666667, 100; 189.311627).
Source:
Assuring a global optimum by the use of an upper bound
on the lower (dual) bound.
Computers and Chemical Engineering 2, 83-92.
https://doi.org/10.1016/0098-1354(78)80012-X

```
fcn <-
  list(obj = function(x) {
    35*x[1]^0.6 + 35*x[2]^0.6
  },
  eq = 2,
  con = function(x) {
    x1 <- x[1]; x3 <- x[3]
    c(600*x1 - 50*x3 - x1*x3 + 5000,
      600*x[2] + 50*x3 - 15000)
  })
JDEoptim(c(0, 0, 100), c(34, 17, 300),
  fn = fcn$obj, constr = fcn$con, meq = fcn$eq,
  tol = 1e-7, trace = TRUE, triter = 50)
```

Designing a pressure vessel
Case A: all variables are treated as continuous
1.1 <= x1 <= 12.5*, 0.6 <= x2 <= 12.5*,
0.0 <= x3 <= 240.0*, 0.0 <= x4 <= 240.0
Roughly guessed*
The global optimum is (x1, x2, x3, x4; f) =
(1.100000, 0.600000, 56.99482, 51.00125; 7019.031).
Source:
Lampinen, Jouni, and Zelinka, Ivan (1999).
Mechanical engineering design optimization
by differential evolution.
In: David Corne, Marco Dorigo and Fred Glover (Editors),
New Ideas in Optimization, McGraw-Hill, pp 127-146

```
pinenglish_vessel_A <-
  list(obj = function(x) {
    x1 <- x[1]; x2 <- x[2]; x3 <- x[3]; x4 <- x[4]
    0.6224*x1*x3*x4 + 1.7781*x2*x3^2 +
    3.1611*x1^2*x4 + 19.84*x1^2*x3
  },
  con = function(x) {
    x1 <- x[1]; x2 <- x[2]; x3 <- x[3]; x4 <- x[4]
    c(0.0193*x3 - x1,
      0.00954*x3 - x2,
      750.0*1728.0 - pi*x3^2*x4 - 4/3*pi*x3^3)
  })
JDEoptim(c(1.1, 0.6, 0.0, 0.0),
```
Mixed integer nonlinear programming

Designing a pressure vessel
Case B: solved according to the original problem statements
steel plate available in thicknesses multiple of 0.0625 inch

wall thickness of the shell 1.1 [18*0.0625] <= x1 <= 12.5 [200*0.0625]
heads 0.6 [10*0.0625] <= x2 <= 12.5 [200*0.0625]
0.0 <= x3 <= 240.0, 0.0 <= x4 <= 240.0
The global optimum is (x1, x2, x3, x4; f) =
(1.125 [18*0.0625], 0.625 [10*0.0625],
58.29016, 43.69266; 7197.729).

pressure_vessel_B <-
list(obj = function(x) {
  x1 <- floor(x[1])*0.0625
  x2 <- floor(x[2])*0.0625
  x3 <- x[3]; x4 <- x[4]
  0.6224*x1*x3*x4 + 1.7781*x2*x3^2 +
  3.1611*x1^2*x4 + 19.84*x1^2*x3
},
  con = function(x) {
    x1 <- floor(x[1])*0.0625
    x2 <- floor(x[2])*0.0625
    x3 <- x[3]; x4 <- x[4]
    c(0.0193*x3 - x1,
      0.00954*x3 - x2,
      750.0*1728.0 - pi*x3^2*x4 - 4/3*pi*x3^3)
  })
res <- JDEoptim(c(18, 10, 0.0, 0.0),
c(200+1, 200+1, 240.0, 240.0),
fn = pressure_vessel_B$obj,
constr = pressure_vessel_B$con,
tol = 1e-7, trace = TRUE, triter = 50)
res
# Now convert to integer x1 and x2
c(floor(res$par[1:2]), res$par[3:4])
NCDEoptim

Description

A bespoke implementation of the ‘NCDE’ (neighborhood based crowding DE) algorithm by Qu et al. (2012) doi:10.1109/TEVC.2011.2161873, assisted with the dynamic archive mechanism of Epitropakis et al. (2013) doi:10.1109/CEC.2013.6557556.

Usage

NCDEoptim(lower, upper, fn, 
    constr = NULL, meq = 0, eps = 1e-5, 
    crit = 1e-5, niche_radius = NULL, archive_size = 100, 
    reinit_if_solu_in_arch = TRUE, 
    NP = 100, Fl = 0.1, Fu = 1, CRl = 0, CRu = 1.1, 
    nbngbrsl = NP/20, nbngbrsu = NP/5, 
    tau_F = 0.1, tau_CR = 0.1, tau_pF = 0.1, 
    tau_nbngbrs = 0.1, 
    jitter_factor = 0.001, 
    maxiter = 2000, 
    add_to_init_pop = NULL, trace = FALSE, triter = 1, 
    ...)

Arguments

lower, upper numeric vectors, the lower and upper bounds of the search space (box constraints); must be finite (is.finite).

fn a function to be minimized that takes a numeric vector \(X_i\) as first argument and returns the value of the objective.

constr a vector function specifying equality constraints defined to equal zero \(h_j(X_i) = 0, j = 1, \ldots, meq\), followed by inequality constraints defined as lesser than zero \(g_j(X_i) \leq 0, j = meq + 1, \ldots\). This function takes \(X_i\) as its first argument and returns a numeric vector with the same length of the total number of constraints. It defaults to NULL, which means that bound-constrained minimization is used.

meq an integer, the first meq constraints are equality constraints whereas the remaining ones are inequality constraints. Defaults to 0 (inequality constraints only).

eps the maximal admissible constraint violation for equality constraints. A numeric vector of small positive tolerance values with length meq used in the transformation of equalities into inequalities of the form \(|h_j(X_i)| - \epsilon \leq 0\). A scalar value is expanded to apply to all equality constraints. Default is 1e-5.

crit a numeric, the absolute acceptance threshold on the archive strategy. If \(|fn(X_{best\_so\_far\_in\_archive}) - X_i| \leq crit\), a solution \(X_i\) is checked for possible insertion into the dynamic archive. Defaults to 1e-5.

niche_radius a numeric, the absolute tolerance used to decide whether the solution \(X_i\) is identical to an already existing local or global solution in the archive. It defaults to NULL, meaning that the niche radius is adaptively chosen during the search. Results are much better if one is able to provide a reasonable value.

archive_size an integer, the maximum number of solutions that can be kept in the archive; entries above this limit are discarded. Default is 100.
**reinit_if_solu_in_arch**

A logical, if TRUE, any solution \(X_i\) already in the archive **reinitializes** its nearest neighbor in the population within the range \([\text{lower}, \text{upper}]\). Default is TRUE.

**NP**

An integer, the population size. Defaults to 100.

**Fl**

A numeric, the minimum value that the **scaling factor** \(f\) could take. It defaults to 0.1.

**Fu**

A numeric, the maximum value that the **scaling factor** \(f\) could take. It defaults to 1.

**CRl**

A numeric, the minimum value to be used for the **crossover constant** \(CR\). It defaults to 0.

**CRu**

A numeric, the maximum value to be used for the **crossover constant** \(CR\). It defaults to 1

**nbngbrsl**

An integer, the lower limit for the **neighborhood size** \(nbngbrs\). It defaults to \(1/20\) of the population size.

**nbngbrsu**

An integer, the upper limit for the **neighborhood size** \(nbngbrs\). It defaults to \(1/5\) of the population size.

**tau_F**

A numeric, the probability that the **scaling factor** \(F\) is updated. Defaults to 0.1.

**tau_CR**

A numeric, the probability that the **crossover constant** \(CR\) is updated. Defaults to 0.1.

**tau_pF**

A numeric, the probability that the **mutation probability** \(p_F\) in the mutation strategy **DE/rand/1/either-or** is updated. Defaults to 0.1.

**tau_nbngbrs**

A numeric, the probability that the **neighborhood size** \(nbngbrs\) is updated. Defaults to 0.1.

**jitter_factor**

A numeric, the tuning constant for jitter. If NULL only **dither** is used. Default is 0.001.

**maxiter**

An integer, the maximum number of iterations allowed which is the **stopping condition**. Default is 2000.

**add_to_init_pop**

A numeric vector of length length(\text{lower}) or column-wise matrix with length(\text{lower}) rows specifying initial candidate solutions which are appended to the randomly generated initial population. Default is NULL.

**trace**

A logical, determines whether or not to monitor the iteration process. Default is FALSE.

**triter**

An integer, trace output is printed at every triter iterations. Default is 1.

... additional arguments passed to fn and constr.

**Details**

This implementation differs mainly from the original ‘NCDE’ algorithm of Qu et al. (2012) by employing the archiving procedure proposed in Epitropakis et al. (2013) and the adaptive ‘jDE’ strategy instead of canonical Differential Evolution. The key reason for archiving good solutions during the search process is to prevent them from being lost during evolution. Constraints are tackled through the \(\epsilon\)-constraint method as proposed in Poole and Allen (2019). The ‘jDE’ and \(\epsilon\)-constraint mechanisms are applied in the same way as in **JDEoptim**, but with **synchronous** mode.
of population update. In contrast, the reinitialization in the current population triggered by already
found solutions is done \textit{asynchronously}.

Each line of trace output follows the format of:

\textbf{iteration} : \langle \text{value of niche radius} \rangle \text{population} \rangle \langle \text{value of best solution} \rangle \text{best solution}
\{ \text{index of violated constraints} \} \text{archive} \rangle \langle \text{number of solutions found} \rangle \langle \text{value of best}
\text{solution} \rangle \text{best solution}

\textbf{Value}

A list with the following components:

- \textbf{solution\_arch} a \textbf{matrix} whose columns are the local and global minima stored in the \textbf{archive}
of feasible solutions in ascending order of the objective function values.

- \textbf{objective\_arch} the values of $f_i(X_i)$ for the corresponding columns of \textbf{solution\_arch}.

- \textbf{solution\_pop} a \textbf{matrix} whose columns are the local and global minima stored in the \textbf{final}
\textbf{population} in ascending order of the objective function values; feasible solutions come first followed by the infeasible ones.

- \textbf{objective\_pop} the values of $f_i(X_i)$ for the corresponding columns of \textbf{solution\_pop}.

- \textbf{iter} the number of iterations used.

and if there are general constraints present:

- \textbf{constr\_value\_arch} a \textbf{matrix} whose columns contain the values of the constraints for \textbf{solution\_arch}.

- \textbf{constr\_value\_pop} a \textbf{matrix} whose columns contain the values of the constraints for \textbf{solution\_pop}.

\textbf{Note}

\textit{This function is in an experimental stage.}

\textbf{Author(s)}

Eduardo L. T. Conceicao <mail@eduardoconceicao.org>

\textbf{References}

Epitropakis, M. G., Li, X. and Burke, E. K. (2013) A dynamic archive niching differential evolu-
tion algorithm for multimodal optimization; in \textit{2013 IEEE Congress on Evolutionary Computation}
(\textit{CEC}). IEEE, pp. 79–86. doi:10.1109/CEC.2013.6557556.

Evolutionary Computation} \textbf{44}, 74–100. doi:10.1016/j.swevo.2018.11.004.

Qu, B. Y., Suganthan, P. N. and Liang, J. J. (2012) Differential evolution with neighborhood muta-
doi:10.1109/TEVC.2011.2161873.
Examples

# NOTE: Examples were excluded from testing
to reduce package check time.

# Use a preset seed so test values are reproducible.
set.seed(1234)

# Warning: the examples are run using a very small number of
# iterations to decrease execution time.

# Bound-constrained optimization

# Vincent function
#
# f(x) = -mean(sin(10*log(x)))
#
# 0.25 <= xi <= 10, i = {1, 2, ..., n}
# The function has 6^n global minima without local minima.

NCDEoptim(c(0.25, 0.25), c(10, 10),
          \(x\) -mean(sin(10*log(x))),
          niche_radius = 0.2,
          maxiter = 200, trace = TRUE, triter = 20)

# Nonlinear constrained optimization

# Function F10 of Poole and Allen (2019)
#
# f(x) = -sin(5*pi*x)^6 + 1
# subject to:
# g(x) = -cos(10*pi*x) <= 0
#
# 0 <= x <= 1
# The 10 global optima are
# (x1*, ..., x10*; f*) = ((2*(1:10) - 1)/20; 0.875).

NCDEoptim(0, 1,
          \(x\) -sin(5*pi*x)^6 + 1, \(x\) -cos(10*pi*x),
          niche_radius = 0.05,
          maxiter = 200, trace = TRUE, triter = 20)
Index

* **global optimization**
  JDEoptim, 2

* **multimodal optimization**
  NCDEoptim, 8

* **nonlinear**
  JDEoptim, 2

* **optimize**
  JDEoptim, 2

DEoptim, 6

function, 2, 9

is.finite, 2, 9

JDEoptim, 2, 10

matrix, 3, 10, 11

NCDEoptim, 8

RNGkind, 4

RNGversion, 4

sample, 4

set.seed, 4