Package: nleqslv (via r-universe)

October 28, 2026

Version 3.3.7

Title Solve Systems of Nonlinear Equations

Description Solve a system of nonlinear equations using a Broyden or a Newton method with a choice of global strategies such as line search and trust region. There are options for using a numerical or user supplied Jacobian, for specifying a banded numerical Jacobian and for allowing a singular or ill-conditioned Jacobian.

License GPL (>= 2)

Encoding UTF-8

URL https://bertcarnell.github.io/nleqslv/, https://github.com/bertcarnell/nleqslv/

BugReports https://github.com/bertcarnell/nleqslv/issues/

RoxygenNote 7.3.3

NeedsCompilation yes

Author Berend Hasselman [aut] (Original Author), Rob Carnell [aut, cre] (ORCID: <https://orcid.org/0009-0009-0465-7564>)

Maintainer Rob Carnell <bertcarnell@gmail.com>

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Repository https://cran.r-universe.dev

Date/Publication 2026-04-10 08:03:28 UTC

RemoteUrl https://github.com/cran/nleqslv

RemoteRef HEAD

RemoteSha b3a89583ce1378bf9acea5d3765e41ec5e699a47

Contents

nleqslv 2
nleqslv-iterationreport 9
print.test.nleqslv 13
searchZeros 14
testnslv 15

Index **18**

`nleqslv` *Solving systems of nonlinear equations with Broyden or Newton*

Description

The function solves a system of nonlinear equations with either a Broyden or a full Newton method. It provides line search and trust region global strategies for difficult systems.

Usage

```
nleqslv(
  x,
  fn,
  jac = NULL,
  ...,
  method = c("Broyden", "Newton"),
  global = c("dbldog", "pwldog", "cline", "qline", "gline", "hook", "none"),
  xscalm = c("fixed", "auto"),
  jacobian = FALSE,
  control = list()
)
```

Arguments

`x`	A numeric vector with an initial guess of the root of the function.
`fn`	A function of `x` returning a vector of function values with the same length as the vector `x`.
`jac`	A function to return the Jacobian for the `fn` function. For a vector valued function `fn` the Jacobian must be a numeric matrix of the correct dimensions. For a scalar valued function `fn` the `jac` function may return a scalar. If not supplied numerical derivatives will be used.
`...`	Further arguments to be passed to `fn` and `jac`.
`method`	The method to use for finding a solution. See 'Details'.
`global`	The global strategy to apply. See 'Details'.
`xscalm`	The type of x scaling to use. See 'Details'.
`jacobian`	A logical indicating if the estimated (approximate) jacobian in the solution should be returned. See 'Details'.
`control`	A named list of control parameters. See 'Details'.

Details

The algorithms implemented in `nleqslv` are based on Dennis and Schnabel (1996).

Methods:

Method `Broyden` starts with a computed Jacobian of the function and then updates this Jacobian after each successful iteration using the so-called Broyden update. This method often shows super linear convergence towards a solution. When `nleqslv` determines that it cannot continue with the current Broyden matrix it will compute a new Jacobian.

Method `Newton` calculates a Jacobian of the function `fn` at each iteration. Close to a solution this method will usually show quadratic convergence.

Both methods apply a so-called (backtracking) global strategy to find a better (more acceptable) iterate. The function criterion used by the algorithm is half of the sum of squares of the function values and "acceptable" means sufficient decrease of the current function criterion value compared to that of the previous iteration. A comprehensive discussion of these issues can be found in Dennis and Schnabel (1996). Both methods apply an unpivoted QR-decomposition to the Jacobian as implemented in Lapack. The Broyden method applies a rank-1 update to the Jacobian at the end of each iteration and is based on a simplified and modernized version of the algorithm described in Reichel and Gragg (1990).

Global strategies:

When applying a full Newton or Broyden step does not yield a sufficiently smaller function criterion value `nleqslv` will attempt to decrease the steplength using one of several so-called global strategies.

The `global` argument indicates which global strategy to use or to use no global strategy

`cline` a cubic line search

`qline` a quadratic line search

`gline` a geometric line search

`dbldog` a trust region method using the double dogleg method as described in Dennis and Schnabel (1996)

`pwldog` a trust region method using the Powell dogleg method as developed by Powell (1970).

`hook` a trust region method described by Dennis and Schnabel (1996) as *The locally constrained optimal ("hook") step*. It is equivalent to a Levenberg-Marquardt algorithm as described in MoréMore (1978) and Nocedal and Wright (2006).

`none` Only a pure local Newton or Broyden iteration is used. The maximum stepsize (see below) is taken into account. The default maximum number of iterations (see below) is set to 20.

The double dogleg method is the default global strategy employed by this package.

Which global strategy to use in a particular situation is a matter of trial and error. When one of the trust region methods fails, one of the line search strategies should be tried. Sometimes a trust region will work and sometimes a line search method; neither has a clear advantage but in many cases the double dogleg method works quite well.

When the function to be solved returns non-finite function values for a parameter vector `x` and the algorithm is *not* evaluating a numerical Jacobian, then any non-finite values will be replaced by a large number forcing the algorithm to backtrack, i.e. decrease the line search factor or decrease the trust region radius.

Scaling:

The elements of vector x may be scaled during the search for a zero of fn. The xscalm argument provides two possibilities for scaling

fixed the scaling factors are set to the values supplied in the control argument and remain unchanged during the iterations. The scaling factor of any element of x should be set to the inverse of the typical value of that element of x, ensuring that all elements of x are approximately equal in size.

auto the scaling factors are calculated from the euclidean norms of the columns of the Jacobian matrix. When a new Jacobian is computed, the scaling values will be set to the euclidean norm of the corresponding column if that is larger than the current scaling value. Thus the scaling values will not decrease during the iteration. This is the method described in MoréMore (1978). Usually manual scaling is preferable.

Jacobian:

When evaluating a numerical Jacobian, an error message will be issued on detecting non-finite function values. An error message will also be issued when a user supplied jacobian contains non-finite entries.

When the jacobian argument is set to TRUE the final Jacobian or Broyden matrix will be returned in the return list. The default value is FALSE; i.e. to not return the final matrix. There is no guarantee that the final Broyden matrix resembles the actual Jacobian.

The package can cope with a singular or ill-conditioned Jacobian if needed by setting the allowSingular component of the control argument. The method used is described in Dennis and Schnabel (1996); it is equivalent to a Levenberg-Marquardt type adjustment with a small damping factor. *There is no guarantee that this method will be successful.* Warning: nleqslv *may report spurious convergence in this case.*

By default nleqslv returns an error if a Jacobian becomes singular or very ill-conditioned. A Jacobian is considered to be very ill-conditioned when the estimated inverse condition is less than or equal to a specified tolerance with a default value equal to 10^{-12}; this can be changed and made smaller with the cndtol item of the control argument. *There is no guarantee that any change will be effective.*

Control options:

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

xtol The relative steplength tolerance. When the relative steplength of all scaled x values is smaller than this value convergence is declared. The default value is 10^{-8}.

ftol The function value tolerance. Convergence is declared when the largest absolute function value is smaller than ftol. The default value is 10^{-8}.

btol The backtracking tolerance. When the relative steplength in a backtracking step to find an acceptable point is smaller than the backtracking tolerance, the backtracking is terminated. In the Broyden method a new Jacobian will be calculated if the Jacobian is outdated. The default value is 10^{-3}.

cndtol The tolerance of the test for ill conditioning of the Jacobian or Broyden approximation. If less than the machine precision it will be silently set to the machine precision. When the estimated inverse condition of the (approximated) Jacobian matrix is less than or equal to the value of cndtol the matrix is deemed to be ill-conditioned, in which case an error will be reported if the allowSingular component is set to FALSE. The default value is 10^{-12}.

sigma Reduction factor for the geometric line search. The default value is 0.5.

`scalex` a vector of scaling values for the parameters. The inverse of a scale value is an indication of the size of a parameter. The default value is 1.0 for all scale values.

`maxit` The maximum number of major iterations. The default value is 150 if a global strategy has been specified. If no global strategy has been specified the default is 20.

`trace` Non-negative integer. A value of 1 will give a detailed report of the progress of the iteration. For a description see `Iteration-report`.

`chkjac` A logical value indicating whether to check a user supplied Jacobian, if supplied. The default value is `FALSE`. The first 10 errors are printed. The code for this check is derived from the code in Bouaricha and Schnabel (1997).

`delta` Initial (scaled) trust region radius. A value of -1.0 or `"cauchy"` is replaced by the length of the Cauchy step in the initial point. A value of -2.0 or `"newton"` is replaced by the length of the Newton step in the initial point. Any numeric value less than or equal to 0 and not equal to -2.0, will be replaced by -1.0; the algorithm will then start with the length of the Cauchy step in the initial point. If it is numeric and positive it will be set to the smaller of the value supplied or the maximum stepsize. If it is not numeric and not one of the permitted character strings then an error message will be issued. The default is -2.0.

`stepmax` Maximum scaled stepsize. If this is negative then the maximum stepsize is set to the largest positive representable number. The default is -1.0, so there is no default maximum stepsize.

`dsub` Number of non zero subdiagonals of a banded Jacobian. The default is to assume that the Jacobian is *not* banded. Must be specified if `dsuper` has been specified and must be larger than zero when `dsuper` is zero.

`dsuper` Number of non zero super diagonals of a banded Jacobian. The default is to assume that the Jacobian is *not* banded. Must be specified if `dsub` has been specified and must be larger than zero when `dsub` is zero.

`allowSingular` A logical value indicating if a small correction to the Jacobian when it is singular or too ill-conditioned is allowed. If the correction is less than `100*.Machine$double.eps` the correction cannot be applied and an unusable Jacobian will be reported. The method used is similar to a Levenberg-Marquardt correction and is explained in Dennis and Schnabel (1996) on page 151. It may be necessary to choose a higher value for `cndtol` to enforce the correction. The default value is `FALSE`.

Value

A list containing components

`x` final values for x

`fvec` function values

`termcd` termination code as integer. The values returned are

- `1` Function criterion is near zero. Convergence of function values has been achieved.
- `2` x-values within tolerance. This means that the relative distance between two consecutive x-values is smaller than `xtol` but that the function value criterion is still larger than `ftol`. *Function values may not be near zero; therefore the user must check if function values are acceptably small.*
- `3` No better point found. This means that the algorithm has stalled and cannot find an acceptable new point. This may or may not indicate acceptably small function values.

4 Iteration limit `maxit` exceeded.

5 Jacobian is too ill-conditioned.

6 Jacobian is singular.

7 Jacobian is unusable.

-10 User supplied Jacobian is most likely incorrect.

`message` a string describing the termination code

`scalex` a vector containing the scaling factors, which will be the final values when automatic scaling was selected

`njcnt` number of Jacobian evaluations

`nfcnt` number of function evaluations, excluding those required for calculating a Jacobian and excluding the initial function evaluation (at iteration 0)

`iter` number of outer iterations used by the algorithm. This excludes the initial iteration. The number of backtracks can be calculated as the difference between the `nfcnt` and `iter` components.

`jac` the final Jacobian or the Broyden approximation if `jacobian` was set to `TRUE`. If no iterations were executed, as may happen when the initial guess are sufficiently close the solution, there is no Broyden approximation and the returned matrix will always be the actual Jacobian. If the matrix is singular or too ill-conditioned the returned matrix is of no value.

Warning

You cannot use this function recursively. Thus function `fn` should not in its turn call `nleqslv`.

References

Bouaricha, A. and Schnabel, R.B. (1997), Algorithm 768: TENSOLVE: A Software Package for Solving Systems of Nonlinear Equations and Nonlinear Least-squares Problems Using Tensor Methods, *Transactions on Mathematical Software*, **23**, 2, pp. 174–195.

Dennis, J.E. Jr and Schnabel, R.B. (1996), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Siam.

MoréMore, J.J. (1978), The Levenberg-Marquardt Algorithm, Implementation and Theory, In *Numerical Analysis*, G.A. Watson (Ed.), Lecture Notes in Mathematics 630, Springer-Verlag, pp. 105–116.

Golub, G.H and C.F. Van Loan (1996), Matrix Computations (3rd edition), The John Hopkins University Press.

Higham, N.J. (2002), Accuracy and Stability of Numerical Algorithms, 2nd ed., SIAM, pp. 10–11.

Nocedal, J. and Wright, S.J. (2006), *Numerical Optimization*, Springer.

Powell, M.J.D. (1970), A hybrid method for nonlinear algebraic equations, In *Numerical Methods for Nonlinear Algebraic Equations*, P. Rabinowitz (Ed.), Gordon & Breach.

Powell, M.J.D. (1970), A Fortran subroutine for solving systems nonlinear equations, In *Numerical Methods for Nonlinear Algebraic Equations*, P. Rabinowitz (Ed.), Gordon & Breach.

decompositions,

Reichel, L. and W.B. Gragg (1990), Algorithm 686: FORTRAN subroutines for updating the QR decomposition, *ACM Trans. Math. Softw.*, **16**, 4, pp. 369–377.

See Also

If this function cannot solve the supplied function then it is a good idea to try the function testnslv in this package. For detecting multiple solutions see searchZeros.

Examples

```
# Dennis Schnabel example 6.5.1 page 149
dslnex <- function(x) {
    y <- numeric(2)
    y[1] <- x[1]^2 + x[2]^2 - 2
    y[2] <- exp(x[1]-1) + x[2]^3 - 2
    y
}

jacdsln <- function(x) {
    n <- length(x)
    Df <- matrix(numeric(n*n),n,n)
    Df[1,1] <- 2*x[1]
    Df[1,2] <- 2*x[2]
    Df[2,1] <- exp(x[1]-1)
    Df[2,2] <- 3*x[2]^2

    Df
}

BADjacdsln <- function(x) {
    n <- length(x)
    Df <- matrix(numeric(n*n),n,n)
    Df[1,1] <- 4*x[1]
    Df[1,2] <- 2*x[2]
    Df[2,1] <- exp(x[1]-1)
    Df[2,2] <- 5*x[2]^2

    Df
}

xstart <- c(2,0.5)
fstart <- dslnex(xstart)
xstart
fstart

# a solution is c(1,1)

nleqslv(xstart, dslnex, control=list(btol=.01))

# Cauchy start
nleqslv(xstart, dslnex, control=list(trace=1,btol=.01,delta="cauchy"))

# Newton start
nleqslv(xstart, dslnex, control=list(trace=1,btol=.01,delta="newton"))

# final Broyden approximation of Jacobian (quite good)
```

```
z <- nleqslv(xstart, dslnex, jacobian=TRUE,control=list(btol=.01))
z$x
z$jac
jacdsln(z$x)

# different initial start; not a very good final approximation
xstart <- c(0.5,2)
z <- nleqslv(xstart, dslnex, jacobian=TRUE,control=list(btol=.01))
z$x
z$jac
jacdsln(z$x)

## Not run:
# no global strategy but limit stepsize
# but look carefully: a different solution is found
nleqslv(xstart, dslnex, method="Newton", global="none", control=list(trace=1,stepmax=5))

# but if the stepsize is limited even more the c(1,1) solution is found
nleqslv(xstart, dslnex, method="Newton", global="none", control=list(trace=1,stepmax=2))

# Broyden also finds the c(1,1) solution when the stepsize is limited
nleqslv(xstart, dslnex, jacdsln, method="Broyden", global="none", control=list(trace=1,stepmax=2))

## End(Not run)

# example with a singular jacobian in the initial guess
f <- function(x) {
    y <- numeric(3)
    y[1] <- x[1] + x[2] - x[1]*x[2] - 2
    y[2] <- x[1] + x[3] - x[1]*x[3] - 3
    y[3] <- x[2] + x[3] - 4
    return(y)
}

Jac <- function(x) {
    J <- matrix(0,nrow=3,ncol=3)
    J[,1] <- c(1-x[2],1-x[3],0)
    J[,2] <- c(1-x[1],0,1)
    J[,3] <- c(0,1-x[1],1)
    J
}

# exact solution
xsol <- c(-.5, 5/3 , 7/3)
xsol

xstart <- c(1,2,3)
J <- Jac(xstart)
J
rcond(J)

z <- nleqslv(xstart,f,Jac, method="Newton",control=list(trace=1,allowSingular=TRUE))
all.equal(z$x,xsol)
```

`nleqslv-iterationreport`

Detailed iteration report of nleqslv

Description

The format of the iteration report provided by nleqslv when the `trace` component of the `control` argument has been set to 1.

Details

All iteration reports consist of a series of columns with a header summarising the contents. Common column headers are

`Iter` Iteration counter

`Jac` Jacobian type. The Jacobian type is indicated by `N` for a Newton Jacobian or `B` for a Broyden updated matrix; optionally followed by the letter `s` indicating a totally singular matrix or the letter `i` indicating an ill-conditioned matrix. Unless the Jacobian is singular, the Jacobian type is followed by an estimate of the inverse condition number of the Jacobian in parentheses as computed by Lapack. This column will be blank when backtracking is active.

`Fnorm` square of the euclidean norm of function values / 2

`Largest |f|` infinity norm of $f(x)$ at the current point

Report for linesearch methods

A sample iteration report for the linesearch global methods (`cline`, `qline` and `gline`) is (some intercolumn space has been removed to make the table fit)

Iter	Jac	Lambda	Ftarg	Fnorm	Largest |f|
0				2.886812e+00	2.250000e+00
1	N(9.6e-03)	1.0000	2.886235e+00	5.787362e+05	1.070841e+03
1		0.1000	2.886754e+00	9.857947e+00	3.214799e+00
1		0.0100	2.886806e+00	2.866321e+00	2.237878e+00
2	B(2.2e-02)	1.0000	2.865748e+00	4.541965e+03	9.341610e+01
2		0.1000	2.866264e+00	3.253536e+00	2.242344e+00
2		0.0298	2.866304e+00	2.805872e+00	2.200544e+00
3	B(5.5e-02)	1.0000	2.805311e+00	2.919089e+01	7.073082e+00
3		0.1000	2.805816e+00	2.437606e+00	2.027297e+00
4	B(1.0e-01)	1.0000	2.437118e+00	9.839802e-01	1.142529e+00
5	B(1.9e-01)	1.0000	9.837834e-01	7.263320e-02	3.785249e-01
6	B(2.2e-01)	1.0000	7.261868e-02	1.581364e-02	1.774419e-01
7	B(1.5e-01)	1.0000	1.581047e-02	9.328284e-03	1.213052e-01
8	B(1.7e-01)	1.0000	9.326419e-03	1.003283e-04	1.400491e-02
9	B(1.9e-01)	1.0000	1.003082e-04	3.072159e-06	2.206943e-03

```
10  B(1.5e-01)   1.0000   3.071544e-06  1.143217e-07  4.757203e-04
11  B(1.3e-01)   1.0000   1.142989e-07  1.144686e-09  4.783197e-05
12  B(1.2e-01)   1.0000   1.144457e-09  4.515245e-13  9.502884e-07
13  B(1.2e-01)   1.0000   4.514342e-13  1.404463e-17  5.299876e-09
```

The column headed by Lambda shows the value of the line search parameter. The column headed by Ftarg follows from a sufficient decrease requirement and is the value below which Fnorm must drop if the current step is to be accepted.

The value of Lambda may not be lower than a threshold determined by the setting of control parameter xtol to avoid reporting false convergence. When no acceptable Lambda is possible and the Broyden method is being used, a new Jacobian will be computed.

Report for the pure method

The iteration report for the global method none is almost the same as the above report, except that the column labelled Ftarg is omitted. The column Lambda gives the ratio of the maximum stepsize and the length of the full Newton step. It is either exactly 1.0, indicating that the Newton step is smaller than the maximum stepsize and therefore used unmodified, or smaller than 1.0, indicating that the Newton step is larger than the maximum stepsize and therefore truncated.

Report for the double dogleg global method

A sample iteration report for the global method dbldog is (some intercolumn space has been removed to make the table fit)

```
Iter        Jac     Lambda      Eta     Dlt0     Dltn        Fnorm   Largest |f|
   0                                                  2.886812e+00  2.250000e+00
   1  N(9.6e-03) C          0.9544   0.4671   0.9343* 1.699715e-01  5.421673e-01
   1            W   0.0833   0.9544   0.9343   0.4671  1.699715e-01  5.421673e-01
   2  B(1.1e-02) W   0.1154   0.4851   0.4671   0.4671  1.277667e-01  5.043571e-01
   3  B(7.3e-02) W   0.7879   0.7289   0.4671   0.0759  5.067893e-01  7.973542e-01
   3             C           0.7289   0.0759   0.1519  5.440250e-02  2.726084e-01
   4  B(8.3e-02) W   0.5307   0.3271   0.1519   0.3037  3.576547e-02  2.657553e-01
   5  B(1.8e-01) N          0.6674   0.2191   0.4383  6.566182e-03  8.555110e-02
   6  B(1.8e-01) N          0.9801   0.0376   0.0752  4.921645e-04  3.094104e-02
   7  B(1.9e-01) N          0.7981   0.0157   0.0313  4.960629e-06  2.826064e-03
   8  B(1.6e-01) N          0.3942   0.0029   0.0058  1.545503e-08  1.757498e-04
   9  B(1.5e-01) N          0.6536   0.0001   0.0003  2.968676e-11  5.983765e-06
  10  B(1.5e-01) N          0.4730   0.0000   0.0000  4.741792e-14  2.198380e-07
  11  B(1.5e-01) N          0.9787   0.0000   0.0000  6.451792e-19  8.118586e-10
```

After the column for the Jacobian the letters indicate the following

- C a fraction (≤ 1.0) of the Cauchy or steepest descent step is taken where the fraction is the ratio of the trust region radius and the Cauchy steplength.
- W a convex combination of the Cauchy and eta*(Newton step) is taken. The number in the column headed by Lambda is the weight of the partial Newton step.
- P a fraction (≥ 1.0) of the partial Newton step, equal to eta*(Newton step), is taken where the fraction is the ratio of the trust region radius and the partial Newton steplength.

`N` a normal full Newton step is taken.

The number in the column headed by `Eta` is calculated from an upper limit on the ratio of the length of the steepest descent direction and the length of the Newton step. See Dennis and Schnabel (1996) pp.139ff for the details. The column headed by `Dlt0` gives the trust region size at the start of the current iteration. The column headed by `Dltn` gives the trust region size when the current step has been accepted by the dogleg algorithm.

The trust region size is decreased when the actual reduction of the function value norm does not agree well with the predicted reduction from the linear approximation of the function or does not exhibit sufficient decrease. And increased when the actual and predicted reduction are sufficiently close. The trust region size is not allowed to decrease beyond a threshold determined by the setting of control parameter `xtol`; when that happens the backtracking is regarded as a failure and is terminated. In that case a new Jacobian will be calculated if the Broyden method is being used.

The current trust region step is continued with a doubled trust region size when the actual and predicted reduction agree extremely well. This is indicated by `*` immediately following the value in the column headed by `Dltn`. This could save gradient calculations. However, a trial step is never taken if the current step is a Newton step. If the trial step does not succeed then the previous trust region size is restored by halving the trial size. The exception is when a trial step takes a Newton step. In that case the trust region size is immediately set to the size of the Newton step which implies that a halving of the new size leads to a smaller size for the trust region than before.

Normally the initial trust region radius is the same as the final trust region radius of the previous iteration but the trust region size is restricted by the size of the current Newton step. So when full Newton steps are being taken, the trust region radius at the start of an iteration may be less than the final value of the previous iteration. The double dogleg method and the trust region updating procedure are fully explained in sections 6.4.2 and 6.4.3 of Dennis and Schnabel (1996).

Report for the single (Powell) dogleg global method

A sample iteration report for the global method `pwldog` is (some intercolumn space has been removed to make the table fit)

Iter	Jac		Lambda	Dlt0	Dltn	Fnorm	Largest |f|
0						2.886812e+00	2.250000e+00
1	N(9.6e-03)	C		0.4671	0.9343*	1.699715e-01	5.421673e-01
1		W	0.0794	0.9343	0.4671	1.699715e-01	5.421673e-01
2	B(1.1e-02)	W	0.0559	0.4671	0.4671	1.205661e-01	4.890487e-01
3	B(7.3e-02)	W	0.5662	0.4671	0.0960	4.119560e-01	7.254441e-01
3		W	0.0237	0.0960	0.1921	4.426507e-02	2.139252e-01
4	B(8.8e-02)	W	0.2306	0.1921	0.3842*	2.303135e-02	2.143943e-01
4		W	0.4769	0.3842	0.1921	2.303135e-02	2.143943e-01
5	B(1.9e-01)	N		0.1375	0.2750	8.014508e-04	3.681498e-02
6	B(1.7e-01)	N		0.0162	0.0325	1.355741e-05	5.084627e-03
7	B(1.3e-01)	N		0.0070	0.0035	1.282776e-05	4.920962e-03
8	B(1.8e-01)	N		0.0028	0.0056	3.678140e-08	2.643592e-04
9	B(1.9e-01)	N		0.0001	0.0003	1.689182e-12	1.747622e-06
10	B(1.9e-01)	N		0.0000	0.0000	9.568768e-16	4.288618e-08
11	B(1.9e-01)	N		0.0000	0.0000	1.051356e-18	1.422035e-09

This is much simpler than the double dogleg report, since the single dogleg takes either a steepest descent step, a convex combination of the steepest descent and Newton steps or a full Newton step. The number in the column `Lambda` is the weight of the Newton step. The single dogleg method is a special case of the double dogleg method with `eta` equal to 1. It uses the same method of updating the trust region size as the double dogleg method.

Report for the hook step global method

A sample iteration report for the global method `hook` is (some intercolumn space has been removed to make the table fit)

Iter	Jac	mu	dnorm	Dlt0	Dltn	Fnorm	Largest |f|
0						2.886812e+00	2.250000e+00
1	N(9.6e-03) H	0.1968	0.4909	0.4671	0.9343*	1.806293e-01	5.749418e-01
1	H	0.0366	0.9381	0.9343	0.4671	1.806293e-01	5.749418e-01
2	B(2.5e-02) H	0.1101	0.4745	0.4671	0.2336	1.797759e-01	5.635028e-01
3	B(1.4e-01) H	0.0264	0.2341	0.2336	0.4671	3.768809e-02	2.063234e-01
4	B(1.6e-01) N		0.0819	0.0819	0.1637	3.002274e-03	7.736213e-02
5	B(1.8e-01) N		0.0513	0.0513	0.1025	5.355533e-05	1.018879e-02
6	B(1.5e-01) N		0.0090	0.0090	0.0179	1.357039e-06	1.224357e-03
7	B(1.5e-01) N		0.0004	0.0004	0.0008	1.846111e-09	6.070166e-05
8	B(1.4e-01) N		0.0000	0.0000	0.0001	3.292896e-12	2.555851e-06
9	B(1.5e-01) N		0.0000	0.0000	0.0000	7.281583e-18	3.800552e-09

The column headed by `mu` shows the Levenberg-Marquardt parameter when the Newton step is larger than the trust region radius. The column headed by `dnorm` gives the Euclidean norm of the step (adjustment of the current `x`) taken by the algorithm. The absolute value of the difference with `Dlt0` is less than 0.1 times the trust region radius.

After the column for the Jacobian the letters indicate the following

`H` a Levenberg-Marquardt restricted step is taken.

`N` a normal full Newton step is taken.

The meaning of the columns headed by `Dlt0` and `Dltn` is identical to that of the same columns for the double dogleg method. The method of updating the trust region size is the same as in the double dogleg method.

See Also

`nleqslv`

print.test.nleqslv *Printing the result of* testnslv

Description

Print method for test.nleqslv objects.

Usage

```
## S3 method for class 'test.nleqslv'
print(x, digits = 4, width.cutoff = 45L, ...)
```

Arguments

x	a test.nleqslv object
digits	specifies the minimum number of significant digits to be printed in values.
width.cutoff	integer passed to deparse which sets the cutoff at which line-breaking is tried.
...	additional arguments to print.

Details

This is the print method for objects inheriting from class test.nleqslv. It prints the call to testnslv followed by the description of the experiment (if the title argument was specified in the call to testnslv) and the dataframe containing the results of testnslv.

Value

It returns the object x invisibly.

Examples

```
dslnex <- function(x) {
    y <- numeric(2)
    y[1] <- x[1]^2 + x[2]^2 - 2
    y[2] <- exp(x[1]-1) + x[2]^3 - 2
    y
}
xstart <- c(1.5,0.5)
fstart <- dslnex(xstart)
z <- testnslv(xstart,dslnex)
print(z)
```

`searchZeros` *Solve a nonlinear equation system with multiple roots from multiple initial estimates*

Description

This function solves a system of nonlinear equations with `nleqlsv` for multiple initial estimates of the roots.

Usage

```
searchZeros(x, fn, digits = 4L, ...)
```

Arguments

`x`	A matrix with each row containing an initial guess of the roots.
`fn`	A function of `x` returning a vector of function values with the same length as the vector `x`.
`digits`	integer passed to `round` for locating and removing duplicate rounded solutions.
`...`	Further arguments to be passed to `nleqslv`, `fn` and `jac`.

Details

Each row of `x` is a vector of initial estimates for the argument `x` of `nleqslv`. The function runs `nleqslv` for each row of the matrix `x`. The first initial value is treated separately and slightly differently from the other initial estimates. It is used to check if all arguments in `...` are valid arguments for `nleqslv` and the function to be solved. This is done by running `nleqslv` with no condition handling. If an error is then detected an error message is issued and the function stops. For the remaining initial estimates `nleqslv` is executed silently. Only solutions for which the `nleqslv` termination code `tcode` equals `1` are regarded as valid solutions. The rounded solutions (after removal of duplicates) are used to order the solutions in increasing order. These rounded solutions are not included in the return value of the function.

Value

If no solutions are found `NULL` is returned. Otherwise a list containing the following components is returned

`x` a matrix with each row containing a unique solution (unrounded)

`xfnorm` a vector of the function criterion associated with each row of the solution matrix `x`.

`fnorm` a vector containing the function criterion for every converged result

`idxcvg` a vector containing the row indices of the matrix of initial estimates for which function value convergence was achieved

`idxxtol` a vector containing the row indices of the matrix of initial estimates for which x-value convergence was achieved

idxnocvg a vector containing the row indices of the matrix of initial estimates which lead to an nleqslv termination code > 2

idxfatal a vector containing the row indices of the matrix of initial estimates for which a fatal error occurred that made nleqslv stop

xstart a matrix of the initial estimates corresponding to the solution matrix

cvgstart a matrix of all initial estimates for which convergence was achieved

Examples

```
# Dennis Schnabel example 6.5.1 page 149 (two solutions)
set.seed(123)
dslnex <- function(x) {
    y <- numeric(2)
    y[1] <- x[1]^2 + x[2]^2 - 2
    y[2] <- exp(x[1]-1) + x[2]^3 - 2
    y
}
xstart <- matrix(runif(50, min=-2, max=2),ncol=2)
ans <- searchZeros(xstart,dslnex, method="Broyden",global="dbldog")
ans

# more complicated example
# R. Baker Kearfott, Some tests of Generalized Bisection,
# ACM Transactions on Methematical Software, Vol. 13, No. 3, 1987, pp 197-220

# A high-degree polynomial system (section 4.3 Problem 12)
# There are 12 real roots (and 126 complex roots to this system!)

hdp <- function(x) {
    f <- numeric(length(x))
    f[1] <- 5 * x[1]^9 - 6 * x[1]^5 * x[2]^2 + x[1] * x[2]^4 + 2 * x[1] * x[3]
    f[2] <- -2 * x[1]^6 * x[2] + 2 * x[1]^2 * x[2]^3 + 2 * x[2] * x[3]
    f[3] <- x[1]^2 + x[2]^2 - 0.265625
    f
}


N <- 40 # at least to find all 12 roots
set.seed(123)
xstart <- matrix(runif(3*N,min=-1,max=1), N, 3)  # N initial guesses, each of length 3
ans <- searchZeros(xstart,hdp, method="Broyden",global="dbldog")
ans$x
```

---

testnslv *Test different methods for solving with* nleqslv

Description

The function tests different methods and global strategies for solving a system of nonlinear equations with `nleqslv`

Usage

```
testnslv(
  x,
  fn,
  jac = NULL,
  ...,
  method = c("Newton", "Broyden"),
  global = c("cline", "qline", "gline", "pwldog", "dbldog", "hook", "none"),
  Nrep = 0L,
  title = NULL
)
```

Arguments

`x`	A numeric vector with an initial guess of the root.
`fn`	A function of `x` returning the function values.
`jac`	A function to return the Jacobian for the `fn` function. For a vector valued function `fn` the Jacobian must be a numeric matrix of the correct dimensions. For a scalar valued function `fn` the `jac` function may return a scalar. If not supplied numerical derivatives will be used.
`...`	Further arguments to be passed to `fn` and `jac` and `nleqslv`.
`method`	The methods to use for finding a solution.
`global`	The global strategies to test. The argument may consist of several possibly abbreviated items.
`Nrep`	Number of repetitions to apply. Default is no repetitions.
`title`	a description of this experiment.

Details

The function solves the function `fn` with `nleqslv` for the specified methods and global strategies. When argument `Nrep` has been set to a number greater than or equal to 1, repetitions of the solving process are performed and the used CPU time in seconds is recorded.

If checking a user supplied jacobian is enabled, then `testnslv` will stop immediately when a possibly incorrect jacobian is detected.

Value

`testnslv` returns an object of class `"test.nleqslv"` which is a list containing the following elements

`call` the matched call

`out` a dataframe containing the results with the following columns

- `Method` method used.
- `Global` global strategy used.
- `termcd` termination code of `nleqslv`.
- `Fcnt` number of function evaluations used by the method and global strategy. This excludes function evaluations made when computing a numerical Jacobian.
- `Jcnt` number of Jacobian evaluations.
- `Iter` number of outer iterations used by the algorithm.
- `Message` a string describing the termination code in an abbreviated form.
- `Fnorm` square of the euclidean norm of the vector of function results divided by 2.
- `cpusecs` CPU seconds used by the requested number of repetitions (only present when argument `Nrep` is not 0).

`title` the description if specified

The abbreviated strings are

`Fcrit` Convergence of function values has been achieved.

`Xcrit` This means that the relative distance between two consecutive x-values is smaller than `xtol`.

`Stalled` The algorithm cannot find an acceptable new point.

`Maxiter` Iteration limit `maxit` exceeded.

`Illcond` Jacobian is too ill-conditioned.

`Singular` Jacobian is singular.

`BadJac` Jacobian is unusable.

`ERROR` `nleqslv` stopped because of a fatal error.

Warning

Any `nleqslv` error message will be displayed immediately and an error for the particular combination of method and global strategy will be recorded in the final dataframe.

Examples

```
dslnex <- function(x) {
    y <- numeric(2)
    y[1] <- x[1]^2 + x[2]^2 - 2
    y[2] <- exp(x[1]-1) + x[2]^3 - 2
    y
}
xstart <- c(0.5,0.5)
fstart <- dslnex(xstart)
testnslv(xstart,dslnex)
# this will encounter an error
xstart <- c(2.0,0.5)
fstart <- dslnex(xstart)
testnslv(xstart,dslnex)
```

Index

∗ **nonlinear**
    nleqslv, 2
    searchZeros, 14
    testnslv, 15
∗ **optimize**
    nleqslv, 2
    searchZeros, 14
    testnslv, 15
∗ **print**
    print.test.nleqslv, 13

deparse, *13*

Iteration-report
        *(*nleqslv-iterationreport*)*, 9

nleqslv, 2, *12*, *14*, *16*
nleqslv-iterationreport, 9

print *(*print.test.nleqslv*)*, 13
print.test.nleqslv, 13

round, *14*

searchZeros, *7*, 14

testnslv, *7*, 15