Package 'nlsr'

October 13, 2022

Type Package

Title Functions for Nonlinear Least Squares Solutions - Updated 2022

Version 2022.8.16

Date 2022-08-16

Maintainer John C Nash <nashjc@uottawa.ca>

Description Provides tools for working with nonlinear least squares problems. For the estimation of models reliable and robust tools than nls(), where the the Gauss-Newton method frequently stops with 'singular gradient' messages. This is accomplished by using, where possible, analytic derivatives to compute the matrix of derivatives and a stabilization of the solution of the estimation equations. Tools for approximate or externally supplied derivative matrices are included. Bounds and masks on parameters are handled properly.

License GPL-2

Encoding UTF-8

Depends R (>= 3.5)

Imports digest

Suggests minpack.lm, optimx, numDeriv, knitr, rmarkdown, markdown, Ryacas, Deriv, microbenchmark, MASS, ggplot2, nlraa

VignetteBuilder knitr

RoxygenNote 7.2.1

NeedsCompilation no

Author John C Nash [aut, cre], Duncan Murdoch [aut]

Repository CRAN

Date/Publication 2022-08-16 21:10:02 UTC

R topics documented:

`coef.nlsr` *coef.nlsr*

Description

prepare and display result of `nlsr` computations

Usage

```
## S3 method for class 'nlsr'
coef(object, ...)
```

Arguments

object	an object of class nlsr
...	additional data needed to evaluate the modeling functions Default FALSE

Details

The set of possible controls to set is as follows

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

---

dex *dex*

Description

Calculate expression for derivative calculations. Converts input to an expression suitable for use in nlsDeriv and related functions.

Usage

```
dex(x, do_substitute = NA, verbose = FALSE)
```

Arguments

x	An expression represented in a variety of ways. See Details.
do_substitute	Whether to use the expression passed as x, or to evaluate it and use its value.
verbose	Print messages describing the process.

Details

If do_substitute is NA, the following rules are used:

An attempt is made to evaluate x. If that fails, the expression is used.

If the evaluation succeeds and the value is a character vector, it is parsed.

If the value is not a character vector and the expression is a single name, the value is used.

Otherwise, the expression is used.

Once the expression is determined it may be simplified, by extracting the language object from a length-one expression vector, or the right-hand-side from a formula.

Normally a warning will be issued if x is a formula containing a left-hand side. To suppress this, wrap the formula in expression(), or pass it as a character string to be parsed.

value

An expression or language object suitable as input to `nlsDeriv` and related functions.

Author(s)

Duncan Murdoch

Examples

```
aa <- dex(~ x^2)
aa
str(aa)
bb <- dex(expression(x^2))
bb
str(bb)
cc <- dex("x^2")
cc
str(cc)
```

---

findSubexprs *findSubexprs*

Description

Try to find the sub-expressions in `expr` ??

Usage

```
findSubexprs(expr, simplify = FALSE, tag = ".expr", verbose = FALSE, ...)
```

Arguments

`expr`	An expression represented in a variety of ways. See Details.
`simplify`	The environment in which simplifications are stored.
`tag`	to be attached to the returned object(s)??
`verbose`	If `TRUE`, then diagnostic output will be printed as derivatives and simplifications are recognized.
`...`	Additional arguments

`fitted.nlsr` *fitted.nlsr*

Description

prepare and display fits of `nlsr` computations

Usage

```
## S3 method for class 'nlsr'
fitted(object = NULL, data = parent.frame(), ...)
```

Arguments

`object`	an object of class `nlsr`
`data`	a data frame with the date for which fits are wanted ?? how do we guarantee it is the data used to fit the model Or do we need a different approach?
`...`	additional data needed to evaluate the modeling functions Default FALSE

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

`isCALL` *isCALL Test if argument is a call*

Description

isCALL

Test if argument is a call

Usage

```
isCALL(x, name)
```

Arguments

`x`	object to be tested
`name`	??need to document better – is it a character string?

isMINUSONE *isMINUSONE*

Description

isMINUSONE

Usage

```
isMINUSONE(x)
```

Arguments

x	object to be tested

isONE *isONE*

Description

Test if argument is one

Usage

```
isONE(x)
```

Arguments

x	object to be tested

isZERO *isZERO*

Description

Test if argument is zero

Usage

```
isZERO(x)
```

Arguments

x	object to be tested

jaback *jaback*

Description

approximate Jacobian via forward differences

Usage

```
jaback(pars, resfn = NULL, bdmsk = NULL, resbest = NULL, ndstep = 1e-07, ...)
```

Arguments

pars	a named numeric vector of parameters to the model
resfn	a function to compute a vector of residuals
bdmsk	?? do we need it? Default is NULL
resbest	If supplied, a vector of the residuals at the parameters pars to save re-evaluation.
ndstep	A tolerance used to alter parameters to compute numerical approximations to derivatives. Default 1e-7.
...	Extra information needed to compute the residuals

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

jacentral *jacentral*

Description

approximate Jacobian via central differences. Note this needs two evaluations per parameter, but generally gives much better approximation of the derivatives.

Usage

```
jacentral(
  pars,
  resfn = NULL,
  bdmsk = NULL,
  resbest = NULL,
  ndstep = 1e-07,
  ...
)
```

Arguments

pars	a named numeric vector of parameters to the model
resfn	a function to compute a vector of residuals
bdmsk	?? do we need it? Default is NULL
resbest	If supplied, a vector of the residuals at the parameters pars to save re-evaluation.
ndstep	A tolerance used to alter parameters to compute numerical approximations to derivatives. Default 1e-7.
...	Extra information needed to compute the residuals

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

---

jafwd *jafwd*

Description

approximate Jacobian via forward differences

Usage

```
jafwd(pars, resfn = NULL, bdmsk = NULL, resbest = NULL, ndstep = 1e-07, ...)
```

Arguments

pars	a named numeric vector of parameters to the model
resfn	a function to compute a vector of residuals
bdmsk	?? do we need it? Default is NULL
resbest	If supplied, a vector of the residuals at the parameters pars to save re-evaluation.
ndstep	A tolerance used to alter parameters to compute numerical approximations to derivatives. Default 1e-7.
...	Extra information needed to compute the residuals

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

jand *jand*

Description

approximate Jacobian via numDeriv::jacobian

Usage

```
jand(pars, resfn = NULL, bdmsk = NULL, resbest = NULL, ndstep = 1e-07, ...)
```

Arguments

pars	a named numeric vector of parameters to the model
resfn	a function to compute a vector of residuals
bdmsk	?? do we need it? Default is NULL
resbest	If supplied, a vector of the residuals at the parameters pars to save re-evaluation.
ndstep	A tolerance used to alter parameters to compute numerical approximations to derivatives. Default 1e-7.
...	Extra information needed to compute the residuals

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

---

model2rjfun *model2rjfun*

Description

These functions create functions to evaluate residuals or sums of squares at particular parameter locations.

Usage

```
model2rjfun(modelformula, pvec, data = NULL, jacobian = TRUE, testresult = TRUE, ...)
SSmod2rjfun(modelformula, pvec, data = NULL, jacobian = TRUE, testresult = TRUE, ...)
model2ssgrfun(modelformula, pvec, data = NULL, gradient = TRUE,
              testresult = TRUE, ...)
modelexpr(fun)
```

Arguments

modelformula	A formula describing a nonlinear regression model.
pvec	A vector of parameters.
data	A dataframe, list or environment holding data used in the calculation.
jacobian	Whether to compute the Jacobian matrix.
testresult	Whether to test the function by evaluating it at pvec.
...	Dot arguments, that is, arguments that may be supplied by name = value to supply information needed to compute specific quantities in the model. @details If pvec does not have names, the parameters will have names generated in the form 'p_<n>', e.g. p_1, p_2. Names that appear in pvec will be taken to be parameters of the model. The data argument may be a dataframe, list or environment, or NULL. If it is not an environment, one will be constructed using the components of data with parent environment set to be the environment of modelformula. SSmod2rjfun returns a function with header function(prm), which evaluates the residuals (and if jacobian is TRUE the Jacobian matrix) of the selfStart model (the rhs is used) at prm. The residuals are defined to be the right hand side of modelformula minus the left hand side. Note that the selfStart model used in the model formula must be available (i.e., loaded). If this function is called from nlxb() then the control element japprox must be @section value model2rjfun returns a function with header function(prm), which evaluates the residuals (and if jacobian is TRUE the Jacobian matrix) of the model at prm. The residuals are defined to be the right hand side of modelformula minus the left hand side. model2ssgrfun returns a function with header function(prm), which evaluates the sum of squared residuals (and if gradient is TRUE the gradient vector) of the model at prm. modelexpr returns the expression used to calculate the vector of residuals (and possibly the Jacobian) used in the previous functions.
gradient	Whether to compute the gradient vector.
fun	A function produced by one of model2rjfun or model2ssgrfun.

keyword

nonlinear

Author(s)

John Nash and Duncan Murdoch

See Also

nls

Examples

```
y <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443, 38.558,
        50.156, 62.948, 75.995, 91.972)

tt <- seq_along(y)  # for testing

mydata <- data.frame(y = y, tt = tt)
f <- y ~ b1/(1 + b2 * exp(-1 * b3 * tt))
p <- c(b1 = 1, b2 = 1, b3 = 1)
rjfn <- model2rjfun(f, p, data = mydata)
rjfn(p)
myexp <- modelexpr(rjfn)
cat("myexp:"); print(myexp)

ssgrfn <- model2ssgrfun(f, p, data = mydata)
ssgrfn(p)
```

`newDeriv` *newDeriv*

Description

Define a new derivative expression

Usage

```
newDeriv (expr, deriv, derivEnv = sysDerivs)
```

Arguments

`expr`	An expression represented in a variety of ways.
`deriv`	An expression giving the derivative of the function call in `expr`.
`derivEnv`	The environment in which to evaluate (??) these

`newSimplification` *newSimplification*

Description

Define a new simplification expression

Usage

```
newSimplification(expr, test, simplification, do_eval = FALSE,
                        simpEnv = sysSimplifications)
```

Arguments

expr	An expression represented in a variety of ways.
test	?? what is this
simplification	?? what is this
do_eval	??evaluate the result
simpEnv	the environment in which the simplification is carried out

---

nlfb *nlfb: nonlinear least squares modeling by functions*

Description

A simplified and hopefully robust alternative to finding the nonlinear least squares minimizer that causes 'formula' to give a minimal residual sum of squares.

Usage

```
nlfb(
  start,
  resfn,
  jacfn = NULL,
  trace = FALSE,
  lower = -Inf,
  upper = Inf,
  weights = NULL,
  data = NULL,
  ctrlcopy = FALSE,
  control = list(),
  ...
)
```

Arguments

start	a numeric vector with all elements present e.g., start=c(b1=200, b2=50, b3=0.3) ?? does it need names?
resfn	A function that evaluates the residual vector for computing the elements of the sum of squares function at the set of parameters `start`. Where this function is created by actions on a formula or expression in `nlxb`, this residual vector will be created by evaluation of the 'model - data', rather than the conventional 'data - model' approach. The sum of squares is the same.
jacfn	A function that evaluates the Jacobian of the sum of squares function, that is, the matrix of partial derivatives of the residuals with respect to each of the parameters. If NULL (default), uses an approximation. The Jacobian MUST be returned as the attribute "gradient" of this function, allowing `jacfn` to have the same name and be the same code block as `resfn`, which may permit some efficiencies of computation.

trace	TRUE for console output during execution
lower	a vector of lower bounds on the parameters. If a single number, this will be applied to all. Default -Inf.
upper	a vector of upper bounds on the parameters. If a single number, this will be applied to all parameters. Default Inf.
weights	A vector of fixed weights. The objective function that will be minimized is the sum of squares where each residual is multiplied by the square root of the corresponding weight. Default NULL implies unit weights.
data	a data frame of variables used by resfn and jacfn to compute the required residuals and Jacobian.
ctrlcopy	If TRUE use control supplied as is. Saves reprocessing controls.
control	a list of control parameters. See nlsr.control().
...	additional data needed to evaluate the modeling functions

Details

nlfb is particularly intended to allow for the resolution of very ill-conditioned or else near zero-residual problems for which the regular nls() function is ill-suited.

This variant uses a qr solution without forming the sum of squares and cross products t(J)

Neither this function nor nlxb have provision for parameter scaling (as in the parscale control of optim and package optimx). This would be more tedious than difficult to introduce, but does not seem to be a priority feature to add.

Value

A list of the following items:

coefficients A named vector giving the parameter values at the supposed solution.

ssquares The sum of squared residuals at this set of parameters.

resid The residual vector at the returned parameters.

jacobian The jacobian matrix (partial derivatives of residuals w.r.t. the parameters) at the returned parameters.

feval The number of residual evaluations (sum of squares computations) used.

jeval The number of Jacobian evaluations used.

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

Examples

```
library(nlsr)
# Scaled Hobbs problem
shobbs.res  <-  function(x){ # scaled Hobbs weeds problem -- residual
 # This variant uses looping
 if(length(x) != 3) stop("shobbs.res -- parameter vector n!=3")
```

```
 y  <-  c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
          38.558, 50.156, 62.948, 75.995, 91.972)
 tt  <-  1:12
 res  <-  100.0*x[1]/(1+x[2]*10.*exp(-0.1*x[3]*tt)) - y
}
shobbs.jac  <-  function(x) { # scaled Hobbs weeds problem -- Jacobian
  jj  <-  matrix(0.0, 12, 3)
  tt  <-  1:12
  yy  <-  exp(-0.1*x[3]*tt)
  zz  <-  100.0/(1+10.*x[2]*yy)
  jj[tt,1]   <-   zz
  jj[tt,2]   <-   -0.1*x[1]*zz*zz*yy
  jj[tt,3]   <-   0.01*x[1]*zz*zz*yy*x[2]*tt
  attr(jj, "gradient") <- jj
  jj
}
st <- c(b1=2, b2=1, b3=1) # a default starting vector (named!)
anlf1bm <- nlfb(start=st, resfn=shobbs.res, jacfn=shobbs.jac, lower=c(2,0,0),
upper=c(2,6,3))
anlf1bm
## Short output
pshort(anlf1bm)
anlf2bm <- nlfb(start=st, resfn=shobbs.res, jacfn=shobbs.jac, lower=c(2,0,0),
upper=c(2,6,9))
anlf2bm
## Short output
pshort(anlf2bm)
```

`nlsDeriv` *nlsDeriv Functions to take symbolic derivatives.*

Description

Compute derivatives of simple expressions symbolically, allowing user-specified derivatives.

Usage

```
nlsDeriv(expr, name, derivEnv = sysDerivs, do_substitute = FALSE, verbose = FALSE, ...)

codeDeriv(expr, namevec, hessian = FALSE, derivEnv = sysDerivs,
          do_substitute = FALSE, verbose = FALSE, ...)

fnDeriv(expr, namevec, args = all.vars(expr), env = environment(expr),
          do_substitute = FALSE, verbose = FALSE, ...)
```

Arguments

expr	An expression represented in a variety of ways. See Details.
name	The name of the variable with respect to which the derivative will be computed.
derivEnv	The environment in which derivatives are stored.
do_substitute	If TRUE, use substitute to get the expression passed as expr, otherwise evaluate it.
verbose	If TRUE, then diagnostic output will be printed as derivatives and simplifications are recognized.
...	Additional parameters which will be passed to codeDeriv from fnDeriv, and to nlsSimplify from nlsDeriv and codeDeriv.
namevec	Character vector giving the variable names with respect to which the derivatives will be taken.
hessian	Logical indicator of whether the 2nd derivatives should also be computed.
args	Desired arguments for the function. See Details below.
env	The environment to be attached to the created function. If NULL, the caller's frame is used.

Details

Functions nlsDeriv and codeDeriv are designed as replacements for the **stats** package functions D and deriv respectively, though the argument lists do not match exactly.

The nlsDeriv function computes a symbolic derivative of an expression or language object. Known derivatives are stored in derivEnv; the default sysDerivs contains expressions for all of the derivatives recognized by deriv, but in addition allows differentiation with respect to any parameter where it makes sense. It also allows the derivative of abs and sign, using an arbitrary choice of 0 at the discontinuities.

The codeDeriv function computes an expression for efficient calculation of the expression value together with its gradient and optionally the Hessian matrix.

The fnDeriv function wraps the codeDeriv result in a function. If the args are given as a character vector (the default), the arguments will have those names, with no default values. Alternatively, a custom argument list with default values can be created using alist; see the example below.

The expr argument will be converted to a language object using dex (but note the different default for do_substitute). Normally it should be a formula with no left hand side, e.g. ~ x^2 , or an expression vector e.g. expression(x, x^2, x^3) , or a language object e.g. quote(x^2). In codeDeriv and fnDeriv the expression vector must be of length 1.

The newDeriv function is used to define a new derivative. The expr argument should match the header of the function as a call to it (e.g. as in the help pages), and the deriv argument should be an expression giving the derivative, including calls to D(arg), which will not be evaluated, but will be substituted with partial derivatives of that argument with respect to name. See the examples below.

If expr or deriv is missing in a call to newDeriv(), it will return the currently saved derivative record from derivEnv. If name is missing in a call to nlsDeriv with a function call, it will print a message describing the derivative formula and return NULL.

To handle functions which act differently if a parameter is missing, code the default value of that parameter to `.MissingVal`, and give a derivative that is conditional on `missing()` applied to that parameter. See the derivatives of `"-"` and `"+"` in the file `derivs.R` for an example.

@section value

If `expr` is an expression vector, `nlsDeriv` and `nlsSimplify` return expression vectors containing the response. For formulas or language objects, a language object is returned.

`codeDeriv` always returns a language object.

`fnDeriv` returns a closure (i.e. a function).

`nlsDeriv` returns the symbolic derivative of the expression.

`newDeriv` with `expr` and `deriv` specified is called for the side effect of recording the derivative in `derivEnv`. If `expr` is missing, it will return the list of names of functions for which derivatives are recorded. If `deriv` is missing, it will return its record for the specified function.

@section note #' `newDeriv(expr, deriv, ...)` will issue a warning if a different definition for the derivative exists in the derivative table.

@author Duncan Murdoch

@seealso `deriv`

@examples newDeriv() newDeriv(sin(x)) nlsDeriv(~ sin(x+y), "x")

f <- function(x) x^2 newDeriv(f(x), 2*x*D(x)) nlsDeriv(~ f(abs(x)), "x")

nlsDeriv(~ pnorm(x, sd=2, log = TRUE), "x") fnDeriv(~ pnorm(x, sd = sd, log = TRUE), "x") f <- fnDeriv(~ pnorm(x, sd = sd, log = TRUE), "x", args = alist(x =, sd = 2)) f f(1) 100*(f(1.01) - f(1)) # Should be close to the gradient

The attached gradient attribute (from f(1.01)) is # meaningless after the subtraction.

Multiple point example xvals <- c(1, 3, 4.123) print(f(xvals)) # Getting a hessian matrix f2 <- ~ (x-2)^3*y - y^2 mydf2 <- fnDeriv(f2, c("x","y"), hessian=TRUE) # display the resulting function print(mydf2) x <- c(1, 2) y <- c(0.5, 0.1) evalmydf2 <- mydf2(x, y) print(evalmydf2) # the first index of the hessian attribute is the point at which we want the hessian hmat1 <- as.matrix(attr(evalmydf2,"hessian")[1,,]) print(hmat1) hmat2 <- as.matrix(attr(evalmydf2,"hessian")[2,,]) print(hmat2)

@section keyword math, nonlinear

`nlsr` *nlsr-package*

Description

nlsr-package

Usage

```
nlsr()
```

Details

This package includes methods for solving nonlinear least squares problems specified by a modeling expression and given a starting vector of named paramters. Note: You must provide an expression of the form lhs ~ rhsexpression so that the residual expression rhsexpression - lhs can be computed. The expression can be enclosed in quotes, and this seems to give fewer difficulties with R. Data variables must already be defined, either within the parent environment or else in the dot-arguments. Other symbolic elements in the modeling expression must be standard functions or else parameters that are named in the start vector.

The main functions in `nlsr` are:

nlfb Nash variant of the Marquardt procedure for nonlinear least squares, with bounds constraints, using a residual and optionally Jacobian described as `R` functions.

nlxb Nash variant of the Marquardt procedure for nonlinear least squares, with bounds constraints, using an expression to describe the residual via an `R` modeling expression. The Jacobian is computed via symbolic differentiation.

wrapnlsr Uses `nlxb` to solve nonlinear least squares then calls `nls()` to create an object of type nls.

model2rjfun returns a function with header `function(prm)`, which evaluates the residuals (and if jacobian is TRUE the Jacobian matrix) of the model at `prm`. The residuals are defined to be the right hand side of `modelformula` minus the left hand side.

model2ssgrfun returns a function with header `function(prm)`, which evaluates the sum of squared residuals (and if gradient is `TRUE` the gradient vector) of the model at `prm`.

modelexpr returns the expression used to calculate the vector of residuals (and possibly the Jacobian) used in the previous functions.

Author(s)

John C Nash and Duncan Murdoch

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

Nash, J. C. (2014) _Nonlinear Parameter Optimization Using R Tools._ Wiley

---

`nlsr.control` *nlsr.control*

Description

set and provide defaults of controls for package `nlsr`

Usage

```
nlsr.control(control)
```

Arguments

control	A list of controls. If missing, the defaults are provided. See below. If a named control is provided, e.g., via a call ctrllist<- nlsr.control(japprox="jand"), then that value is substituted for the default of the control in the FULL list of controls that is returned. NOTE: at 2022-6-17 there is NO CHECK FOR VALIDITY The set of possible controls to set is as follows, and is returned.

Value

femax INTEGER limit on the number of evaluations of residual function Default 10000.

japprox CHARACTER name of the Jacobian approximation to use Default NULL, since we try to use analytic gradient

jemax INTEGER limit on the number of evaluations of the Jacobian Default 5000

lamda REAL initial value of the Marquardt parameter Default 0.0001 Note: mis-spelling as in JNWMS, kept as historical serendipity.

lamdec REAL multiplier used to REDUCE `lambda` (0 < `lamdec` < `laminc`) Default 4, so `lamda <- lamda * (lamdec/laminc)`

laminc REAL multiplier to INCREASE `lambda` (1 < `laminc` Default 10

ndstep REAL stepsize for numerical Jacobian approximation Default 1e-7

numjac LOGICAL If TRUE use a numerical approximation to Jacobian. Default FALSE

offset REAL A value used to test for numerical equality, i.e. `a` and `b` are taken equal if `(a + offset) == (b + offset)` Default 100.

phi REAL Factor used to add unit Marquardt stabilization matrix in Nash modification of LM method. Default 1. choices

psi REAL Factor used to add scaled Marquardt stabilization matrix in Nash modification of LM method. Default 0.

rofftest LOGICAL If TRUE, perform (safeguarded) relative offset convergence test Default TRUE

smallsstest LOGICAL If TRUE tests sum of squares and terminates if very small ??describe Default TRUE

stepredn REAL Factor used to reduce the stepsize in a Gauss-Newton algorithm (Hartley's method). 0 means NO backtrack. Default 0.

watch LOGICAL to provide a pause at the end of each iteration for user to monitor progress. Default FALSE

prtlvl INTEGER The higher the value, the more intermediate output is provided. Default 0

scaleOffset a positive constant to be added to the denominator sum-of-squares in the relative offset convergence criteria. Default 0.

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

nlsSimplify *nlsSimplify*

Description

Try to simplify an expression re: nonlinear least squares

Usage

```
nlsSimplify(expr, simpEnv = sysSimplifications, verbose = FALSE)
```

Arguments

expr	An expression represented in a variety of ways. See Details.
simpEnv	The environment in which simplifications are stored.
verbose	If TRUE, then diagnostic output will be printed as derivatives and simplifications are recognized.

nlxb *nlxb: nonlinear least squares modeling by formula*

Description

A simplified and hopefully robust alternative to finding the nonlinear least squares minimizer that causes 'formula' to give a minimal residual sum of squares.

Usage

```
nlxb(
  formula,
  data = parent.frame(),
  start,
  trace = FALSE,
  lower = NULL,
  upper = NULL,
  weights = NULL,
  control = list(),
  ...
)
```

Arguments

formula	The modeling formula. Looks like 'y~b1/(1+b2*exp(-b3*T))'
data	a data frame containing data for variables used in the formula that are NOT the parameters. This data may also be defined in the parent frame i.e., 'global' to this function
start	MUST be a named vector with all elements present e.g., start=c(b1=200, b2=50, b3=0.3)
trace	TRUE for console output during execution
lower	a vector of lower bounds on the parameters. If a single number, this will be applied to all parameters Default NULL.
upper	a vector of upper bounds on the parameters. If a single number, this will be applied to all parameters. Default NULL.
weights	A vector of fixed weights. The objective function that will be minimized is the sum of squares where each residual is multiplied by the square root of the corresponding weight. Default NULL implies unit weights.
control	a list of control parameters. See nlsr.control().
...	additional data needed to evaluate the modeling functions

Details

nlxb is particularly intended to allow for the resolution of very ill-conditioned or else near zero-residual problems for which the regular nls() function is ill-suited.

This variant uses a qr solution without forming the sum of squares and cross products t(J)

Neither this function nor nlfb have provision for parameter scaling (as in the parscale control of optim and package optimx). This would be more tedious than difficult to introduce, but does not seem to be a priority feature to add.

There are many controls, and some of them are important for nlxb. In particular, if the derivatives needed for developing the Jacobian are NOT in the derivatives table, then we must supply code elsewhere as specified by the control japprox. This was originally just for numerical approximations, with the character strings "jafwd", "jaback", "jacentral" and "jand" leading to the use of a forward, backward, central or package numDeriv approximation. However, it is also possible to use code embedded in the residual function created using the formula. This is particularly useful for selfStart models, and we use the character string "SSJac" to point to such Jacobian code. Note how the starting parameter vector is found using the getInitial function from the stats package as in an example below.

Value

list of solution elements

resid = weighted residuals at the proposed solution jacobian = Jacobian matrix at the proposed solution feval = residual function evaluations used to reach solution from starting parameters jeval = Jacobian function (or approximation) evaluations used coefficients = a named vector of proposed solution parameters ssquares = weighted sum of squared residuals (often the deviance) lower = lower bounds on parameters upper = upper bounds on parameters maskidx = vector if indices of fixed (masked) parameters weights = specified weights on observations formula = the modeling formula resfn = the residual function (unweighted) based on the formula

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

Examples

```
library(nlsr)
weed <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
        38.558, 50.156, 62.948, 75.995, 91.972)
tt <- 1:12
weeddf <- data.frame(tt, weed)
frm <-
wmodu <- weed ~ b1/(1+b2*exp(-b3*tt)) # Unscaled
## nls from unit start FAILS
start1<-c(b1=1, b2=1, b3=1)
hunls1 <- try(nls(wmodu, data=weeddf, start=start1, trace=FALSE))
if (! inherits(hunls1, "try-error")) print(hunls1) ## else cat("Failure -- try-error\n")
## nlxb from unit start
hunlx1 <- try(nlxb(wmodu, data=weeddf, start=c(b1=1, b2=1, b3=1), trace=FALSE))
if (! inherits(hunlx1, "try-error")) print(hunlx1)

st2h<-c(b1=185, b2=10, b3=.3)
#' hunls2 <- try(nls(wmodu, data=weeddf, start=st2h, trace=FALSE))
if (! inherits(hunls1, "try-error")) print(hunls1) ## else cat("Failure -- try-error\n")
## nlxb from unit start
hunlx1 <- try(nlxb(wmodu, data=weeddf, start=st2h, trace=FALSE))
if (! inherits(hunlx1, "try-error")) print(hunlx1)

# Functional models need to use a Jacobian approximation or external calculation.
# For example, the SSlogis() selfStart model from \code{stats} package.

# nls() needs NO starting value
hSSnls<-try(nls(weed~SSlogis(tt, Asym, xmid, scal), data=weeddf))
summary(hSSnls)
# We need to get the start for nlxb explicitly
stSS <- getInitial(weed ~ SSlogis(tt, Asym, xmid, scal), data=weeddf)
hSSnlx<-try(nlxb(weed~SSlogis(tt, Asym, xmid, scal), data=weeddf, start=stSS))
hSSnlx

# nls() can only bound parameters with algorithm="port"
#   and minpack.lm is unreliable in imposing bounds, but nlsr copes fine.
lo<-c(0, 0, 0)
up<-c(190, 10, 2) # Note: start must be admissible.
bnls0<-try(nls(wmodu, data=weeddf, start=st2h,
         lower=lo, upper=up)) # should complain and fail

bnls<-try(nls(wmodu, data=weeddf, start=st2h,
         lower=lo, upper=up, algorith="port"))
summary(bnls)
bnlx<-try(nlxb(wmodu, data=weeddf, start=st2h, lower=lo, upper=up))
bnlx

# nlxb() can also MASK (fix) parameters. The mechanism of maskidx from nls
```

```
# is NO LONGER USED. Instead we set upper and lower parameters equal for
# the masked parameters. The start value MUST be equal to this fixed value.
lo<-c(190, 0, 0) # mask first parameter
up<-c(190, 10, 2)
strt <- c(b1=190, b2=1, b3=1)
mnlx<-try(nlxb(wmodu, start=strt, data=weeddf,
         lower=lo, upper=up))
mnlx
mnls<-try(nls(wmodu, data=weeddf, start=strt,
         lower=lo, upper=up, algorith="port"))
summary(mnls)

# Try first parameter masked and see if we get SEs
lo<-c(200, 0, 0) # mask first parameter
up<-c(100, 10, 5)
strt <- c(b1=200, b2=1, b3=1)
mnlx<-try(nlxb(wmodu, start=strt, data=weeddf,
         lower=lo, upper=up))
mnlx
mnls<-try(nls(wmodu, data=weeddf, start=strt,
         lower=lo, upper=up, algorith="port"))
summary(mnls)
```

`numericDerivR` *numericDerivR: numerically evaluates the gradient of an expression. All in R*

Description

This version is all in R to replace the C version in package stats

Usage

```
numericDerivR(
  expr,
  theta,
  rho = parent.frame(),
  dir = 1,
  eps = .Machine$double.eps^(1/if (central) 3 else 2),
  central = FALSE
)
```

Arguments

`expr`	expression or call to be differentiated. Should evaluate to a numeric vector.
`theta`	character vector of names of numeric variables used in expr.
`rho`	environment containing all the variables needed to evaluate expr.

dir	numeric vector of directions, typically with values in -1, 1 to use for the finite differences; will be recycled to the length of theta.
eps	a positive number, to be used as unit step size hh for the approximate numerical derivative (f(x+h)-f(x))/h (f(x+h)-f(x))/h or the central version, see central.
central	logical indicating if central divided differences should be computed, i.e., (f(x+h) - f(x-h)) / 2h (f(x+h)-f(x-h))/2h. These are typically more accurate but need more evaluations of f()f().

Value

The value of eval(expr, envir = rho) plus a matrix attribute "gradient". The columns of this matrix are the derivatives of the value with respect to the variables listed in theta.

Examples

```
ex <- expression(a/(1+b*exp(-tt*c)) - weed)
weed <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
             38.558, 50.156, 62.948, 75.995, 91.972)
tt <- 1:12
a <- 200; b <- 50; c <- 0.3
dhobb <- numericDerivR(ex, theta=c("a", "b", "c"))
print(dhobb)
# exf <- ~ a/(1+b*exp(-tt*c)) - weed
# Note that a formula doesn't work
# dh1 <-  try(numericDerivR(exf, theta=c("a", "b", "c")))
```

---

nvec *nvec*

Description

Compact display of specified nvec named vector

Usage

```
nvec(vec)
```

Arguments

vec	a named vector of parameters

Value

none (??may want to change that)

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

pctrl *pctrl*

Description

Compact display of specified `control` vector

Usage

```
pctrl(control)
```

Arguments

`control`	a control list

Value

none (??may want to change that)

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

pnlm0 *pnlm0*

Description

Compact display of specified `minpack.lm` object `x`

Usage

```
pnlm0(x)
```

Arguments

`x`	a minpack.lm nlsLM() or nls.lm() result object

Value

none

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

pnls0 *pnls0*

Description

Compact display of specified nls object x

Usage

```
pnls0(x)
```

Arguments

x	an nls() result object

Value

none

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

predict.nlsr *predict.nlsr*

Description

prepare and display predictions from an nlsr model

Usage

```
## S3 method for class 'nlsr'
predict(object = NULL, newdata = list(), ...)
```

Arguments

object	an object of class nlsr
newdata	a dataframe containing columns that match the original dataframe used to estimate the nonlinear model in the nlsr object
...	additional data needed to evaluate the modeling functions Default FALSE

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

print.nlsr *print.nlsr*

Description

prepare and display result of nlsr computations

Usage

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

Arguments

x	an object of class nlsr
...	additional data needed to evaluate the modeling functions Default FALSE

Details

The set of possible controls to set is as follows

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

pshort *pshort*

Description

1 line result display of nlsr computations

Usage

```
pshort(x)
```

Arguments

x	an object of class nlsr

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

`rawres.nlsr` *rawres.nlsr*

Description

prepare and display raw residuals of `nlsr` computations NOTE: we use model - data i.e., rhs - lhs

Usage

```
rawres.nlsr(object = NULL, data = parent.frame(), ...)
```

Arguments

`object`	an object of class `nlsr`
`data`	a data frame with the date for which fits are wanted ?? how do we guarantee it is the data used to fit the model Or do we need a different approach?
`...`	additional data needed to evaluate the modeling functions Default FALSE

Value

A vector of the raw residuals

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

`resgr` *resgr*

Description

Computes the gradient of the sum of squares function for nonlinear least squares where `resfn` and `jacfn` supply the residuals and Jacobian

Usage

```
resgr(prm, resfn, jacfn, ...)
```

Arguments

`prm`	a numeric vector of parameters to the model
`resfn`	a function to compute a vector of residuals
`jacfn`	a function to compute the Jacobian of the sum of squares
`...`	Extra information needed to compute the residuals

Details

?? does it work with approximate Jacobian functions

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

`resid.nlsr` *resid.nlsr*

Description

prepare and display result of `nlsr` computations

Usage

```
## S3 method for class 'nlsr'
resid(object, ...)
```

Arguments

`object`	an object of class `nlsr`
`...`	additional data needed to evaluate the modeling functions

Author(s)

J C Nash nashjc _at_ uottawa.ca

remove _at_export to try to overcome NAMESPACE issue

`residuals.nlsr` *residuals.nlsr*

Description

prepare and display result of `nlsr` computations

Usage

```
## S3 method for class 'nlsr'
residuals(object, ...)
```

Arguments

`object`	an object of class `nlsr`
`...`	additional data needed to evaluate the modeling functions

Author(s)

J C Nash nashjc _at_ uottawa.ca

`resss` *resss*

Description

compute the sum of squares from `resfn` at parameters `prm`

Usage

```
resss(prm, resfn, ...)
```

Arguments

`prm`	a named numeric vector of parameters to the model
`resfn`	a function to compute a vector of residuals
`...`	Extra information needed to compute the residuals

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

`summary.nlsr` *summary.nlsr*

Description

prepare display result of `nlsr` computations - NOT compact output

Usage

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

Arguments

`object`	an object of class `nlsr`
`...`	additional data needed to evaluate the modeling functions

Details

The set of possible controls to set is as follows

Author(s)

J C Nash 2014-7-16 nashjc _at_ uottawa.ca

sysDerivs *sysDerivs*

Description

creates a new environment whose parent is emptyenv

Usage

```
sysDerivs()
```

Format

An object of class `environment` of length 27.

sysSimplifications *sysSimplifications*

Description

creates a new environment whose parent is emptyenv

Usage

```
sysSimplifications
```

Format

An object of class `environment` of length 13.

wrapnlsr *wrapnlsr*

Description

Provides class nls solution to a nonlinear least squares solution using the Nash Marquardt tools.

Usage

```
wrapnlsr(
  formula = NULL,
  data = NULL,
  start = NULL,
  control = NULL,
  trace = FALSE,
  subset = NULL,
  lower = -Inf,
  upper = Inf,
  weights = NULL,
  ...
)
```

Arguments

formula	The modeling formula. Looks like 'y~b1/(1+b2*exp(-b3*T))'
data	a data frame containing data for variables used in the formula that are NOT the parameters. This data may also be defined in the parent frame i.e., 'global' to this function
start	MUST be a named vector with all elements present e.g., start=c(b1=200, b2=50, b3=0.3)
control	a list of control parameters. See nlsr.control().
trace	TRUE for console output during execution (default FALSE)
subset	an optional vector specifying a subset of observations to be used in the fitting process. NOT used currently by nlxb() or nlfb() and will throw an error if present and not NULL.
lower	a vector of lower bounds on the parameters. If a single number, this will be applied to all parameters Default -Inf.
upper	a vector of upper bounds on the parameters. If a single number, this will be applied to all parameters. Default Inf.
weights	A vector of fixed weights. The objective function that will be minimized is the sum of squares where each residual is multiplied by the square root of the corresponding weight. Default NULL implies unit weights. ???
...	additional data needed to evaluate the modeling functions

Value

A solution object of type nls

Examples

```
library(nlsr)
cat("kvanderpoel.R test of wrapnlsr\n")
x<-c(1,3,5,7)
```

```
y<-c(37.98,11.68,3.65,3.93)
pks28<-data.frame(x=x,y=y)
fit0<-try(nls(y~(a+b*exp(1)^(-c*x)), data=pks28, start=c(a=0,b=1,c=1),
              trace=TRUE))
print(fit0)
fit1<-nlxb(y~(a+b*exp(-c*x)), data=pks28, start=c(a=0,b=1,c=1), trace = TRUE)
print(fit1)
cat("\n\n or better\n")
fit2<-wrapnlsr(y~(a+b*exp(-c*x)), data=pks28, start=c(a=0,b=1,c=1),
               lower=-Inf, upper=Inf, trace = TRUE)
fit2

weed <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
          38.558, 50.156, 62.948, 75.995, 91.972)
tt <- 1:12
weeddf <- data.frame(tt, weed)
hobbsu <- weed ~ b1/(1+b2*exp(-b3*tt))
st2 <- c(b1=200, b2=50, b3=0.3)
wts <- 0.5^tt # a straight scaling comes via wts <- rep(0.01, 12)
lo <- c(200, 0, 0)
up <- c(1000, 1000, 1000)
whuw2 <-  try(wrapnlsr(start=st2, formula=hobbsu, data=weeddf, subset=2:11,
                  weights=wts, trace=TRUE, lower=lo, upper=up))
summary(whuw2)
deviance(whuw2)
```

Index