

Package ‘minpack.lm’

May 8, 2012

Version 1.1-6

Title R interface to the Levenberg-Marquardt nonlinear least-squares algorithm found in MINPACK, plus support for bounds

Author Timur V. Elzhov, Katharine M. Mullen, Andrej-Nikolai Spiess, Ben Bolker

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

Description The `nls.lm` function provides an R interface to `lmdr` and `lmdif` from the MINPACK library, for solving nonlinear least-squares problems by a modification of the Levenberg-Marquardt algorithm, with support for lower and upper parameter bounds. The implementation can be used via nls-like calls using the `nlsLM` function.

License file LICENSE

Repository CRAN

Date/Publication 2012-05-08 04:07:02

R topics documented:

<code>nls.lm</code>	1
<code>nls.lm.control</code>	6
<code>nlsLM</code>	8

Index	13
--------------	-----------

<code>nls.lm</code>	<i>Addresses NLS problems with the Levenberg-Marquardt algorithm</i>
---------------------	--

Description

The purpose of `nls.lm` is to minimize the sum square of the vector returned by the function `fn`, by a modification of the Levenberg-Marquardt algorithm. The user may also provide a function `jac` which calculates the Jacobian.

Usage

```
nls.lm(par, lower=NULL, upper=NULL, fn, jac = NULL,
       control = nls.lm.control(), ...)
```

Arguments

par	A list or numeric vector of starting estimates. If par is a list, then each element must be of length 1.
lower	A numeric vector of lower bounds on each parameter. If not given, the default lower bound for each parameter is set to $-\text{Inf}$.
upper	A numeric vector of upper bounds on each parameter. If not given, the default upper bound for each parameter is set to Inf .
fn	A function that returns a vector of residuals, the sum square of which is to be minimized. The first argument of fn must be par.
jac	A function to return the Jacobian for the fn function.
control	An optional list of control settings. See nls.lm.control for the names of the settable control values and their effect.
...	Further arguments to be passed to fn and jac.

Details

Both functions fn and jac (if provided) must return numeric vectors. Length of the vector returned by fn must not be lower than the length of par. The vector returned by jac must have length equal to $\text{length}(\text{fn}(\text{par}, \dots)) \cdot \text{length}(\text{par})$.

The control argument is a list; see [nls.lm.control](#) for details.

Successful completion.

The accuracy of `nls.lm` is controlled by the convergence parameters `ftol`, `ptol`, and `gtol`. These parameters are used in tests which make three types of comparisons between the approximation *par* and a solution *par*₀. `nls.lm` terminates when any of the tests is satisfied. If any of the convergence parameters is less than the machine precision, then `nls.lm` only attempts to satisfy the test defined by the machine precision. Further progress is not usually possible.

The tests assume that fn as well as jac are reasonably well behaved. If this condition is not satisfied, then `nls.lm` may incorrectly indicate convergence. The validity of the answer can be checked, for example, by rerunning `nls.lm` with tighter tolerances.

First convergence test.

If $|z|$ denotes the Euclidean norm of a vector *z*, then this test attempts to guarantee that

$$|fvec| < (1 + \text{ftol}) |fvec_0|,$$

where *fvec*₀ denotes the result of fn function evaluated at *par*₀. If this condition is satisfied with $\text{ftol} \simeq 10^{-k}$, then the final residual norm $|fvec|$ has *k* significant decimal digits and `info` is set to 1 (or to 3 if the second test is also satisfied). Unless high precision solutions are required, the recommended value for `ftol` is the square root of the machine precision.

Second convergence test.

If D is the diagonal matrix whose entries are defined by the array `diag`, then this test attempt to guarantee that

$$|D(par - par_0)| < ptol |D par_0|,$$

If this condition is satisfied with $ptol \simeq 10^{-k}$, then the larger components of $(D par)$ have k significant decimal digits and `info` is set to 2 (or to 3 if the first test is also satisfied). There is a danger that the smaller components of $(D par)$ may have large relative errors, but if `diag` is internally set, then the accuracy of the components of par is usually related to their sensitivity. Unless high precision solutions are required, the recommended value for `ptol` is the square root of the machine precision.

Third convergence test.

This test is satisfied when the cosine of the angle between the result of `fn` evaluation $fvec$ and any column of the Jacobian at par is at most `gtol` in absolute value. There is no clear relationship between this test and the accuracy of `nls.lm`, and furthermore, the test is equally well satisfied at other critical points, namely maximizers and saddle points. Therefore, termination caused by this test (`info = 4`) should be examined carefully. The recommended value for `gtol` is zero.

Unsuccessful completion.

Unsuccessful termination of `nls.lm` can be due to improper input parameters, arithmetic interrupts, an excessive number of function evaluations, or an excessive number of iterations.

Improper input parameters.

`info` is set to 0 if $length(par) = 0$, or $length(fvec) < length(par)$, or $ftol < 0$, or $ptol < 0$, or $gtol < 0$, or $maxfev \leq 0$, or $factor \leq 0$.

Arithmetic interrupts.

If these interrupts occur in the `fn` function during an early stage of the computation, they may be caused by an unacceptable choice of par by `nls.lm`. In this case, it may be possible to remedy the situation by rerunning `nls.lm` with a smaller value of `factor`.

Excessive number of function evaluations.

A reasonable value for `maxfev` is $100 \cdot (length(par) + 1)$. If the number of calls to `fn` reaches `maxfev`, then this indicates that the routine is converging very slowly as measured by the progress of $fvec$ and `info` is set to 5. In this case, it may be helpful to force `diag` to be internally set.

Excessive number of function iterations.

The allowed number of iterations defaults to 50, can be increased if desired.

The list returned by `nls.lm` has methods for the generic functions `coef`, `deviance`, `df.residual`, `print.residuals`, `summary`, `confint`, and `vcov`.

Value

A list with components:

<code>par</code>	The best set of parameters found.
<code>hessian</code>	A symmetric matrix giving an estimate of the Hessian at the solution found.

fvec	The result of the last fn evaluation; that is, the residuals.
info	info is an integer code indicating the reason for termination. <ol style="list-style-type: none"> 0 Improper input parameters. 1 Both actual and predicted relative reductions in the sum of squares are at most ftol. 2 Relative error between two consecutive iterates is at most ptol. 3 Conditions for info = 1 and info = 2 both hold. 4 The cosine of the angle between fvec and any column of the Jacobian is at most gtol in absolute value. 5 Number of calls to fn has reached maxfev. 6 ftol is too small. No further reduction in the sum of squares is possible. 7 ptol is too small. No further improvement in the approximate solution par is possible. 8 gtol is too small. fvec is orthogonal to the columns of the Jacobian to machine precision. 9 The number of iterations has reached maxiter.
message	character string indicating reason for termination
.	.
diag	The result list of diag. See Details .
niter	The number of iterations completed before termination.
rsstrace	The residual sum of squares at each iteration. Can be used to check the progress each iteration.
deviance	The sum of the squared residual vector.

Note

The public domain FORTRAN sources of MINPACK package by J.J. Moré, implementing the Levenberg-Marquardt algorithm were downloaded from <http://ftp.netlib.org/minpack>, and left unchanged. The contents of this manual page are largely extracted from the comments of MINPACK sources.

References

J.J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," in *Lecture Notes in Mathematics* **630**: Numerical Analysis, G.A. Watson (Ed.), Springer-Verlag: Berlin, 1978, pp. 105-116.

See Also

[optim](#), [nls](#), [nls.lm.control](#)

Examples

```
##### example 1

## values over which to simulate data
x <- seq(0,5,length=100)

## model based on a list of parameters
getPred <- function(parS, xx) parS$a * exp(xx * parS$b) + parS$c

## parameter values used to simulate data
pp <- list(a=9,b=-1, c=6)

## simulated data, with noise
simDNoisy <- getPred(pp,x) + rnorm(length(x),sd=.1)

## plot data
plot(x,simDNoisy, main="data")

## residual function
residFun <- function(p, observed, xx) observed - getPred(p,xx)

## starting values for parameters
parStart <- list(a=3,b=-.001, c=1)

## perform fit
nls.out <- nls.lm(par=parStart, fn = residFun, observed = simDNoisy,
xx = x, control = nls.lm.control(nprint=1))

## plot model evaluated at final parameter estimates
lines(x,getPred(as.list(coef(nls.out)), x), col=2, lwd=2)

## summary information on parameter estimates
summary(nls.out)

##### example 2

## function to simulate data
f <- function(TT, tau, N0, a, f0) {
  expr <- expression(N0*exp(-TT/tau)*(1 + a*cos(f0*TT)))
  eval(expr)
}

## helper function for an analytical gradient
j <- function(TT, tau, N0, a, f0) {
  expr <- expression(N0*exp(-TT/tau)*(1 + a*cos(f0*TT)))
  c(eval(D(expr, "tau")), eval(D(expr, "N0" )),
    eval(D(expr, "a" )), eval(D(expr, "f0" )))
}

## values over which to simulate data
TT <- seq(0, 8, length=501)
```

```

## parameter values underlying simulated data
p <- c(tau = 2.2, N0 = 1000, a = 0.25, f0 = 8)

## get data
Ndet <- do.call("f", c(list(TT = TT), as.list(p)))
## with noise
N <- Ndet + rnorm(length(Ndet), mean=Ndet, sd=.01*max(Ndet))

## plot the data to fit
par(mfrow=c(2,1), mar = c(3,5,2,1))
plot(TT, N, bg = "black", cex = 0.5, main="data")

## define a residual function
fcn <- function(p, TT, N, fcall, jcall)
  (N - do.call("fcall", c(list(TT = TT), as.list(p))))

## define analytical expression for the gradient
fcn.jac <- function(p, TT, N, fcall, jcall)
  -do.call("jcall", c(list(TT = TT), as.list(p)))

## starting values
guess <- c(tau = 2.2, N0 = 1500, a = 0.25, f0 = 10)

## to use an analytical expression for the gradient found in fcn.jac
## uncomment jac = fcn.jac
out <- nls.lm(par = guess, fn = fcn, jac = fcn.jac,
             fcall = f, jcall = j,
             TT = TT, N = N, control = nls.lm.control(nprint=1))

## get the fitted values
N1 <- do.call("f", c(list(TT = TT), out$par))

## add a blue line representing the fitting values to the plot of data
lines(TT, N1, col="blue", lwd=2)

## add a plot of the log residual sum of squares as it is made to
## decrease each iteration; note that the RSS at the starting parameter
## values is also stored
plot(1:(out$niter+1), log(out$rsstrace), type="b",
     main="log residual sum of squares vs. iteration number",
     xlab="iteration", ylab="log residual sum of squares", pch=21,bg=2)

## get information regarding standard errors
summary(out)

```

Description

Allow the user to set some characteristics Levenberg-Marquardt nonlinear least squares algorithm implemented in `nls.lm`.

Usage

```
nls.lm.control(ftol = sqrt(.Machine$double.eps),
ptol = sqrt(.Machine$double.eps), gtol = 0, diag = list(), epsfcn = 0,
factor = 100, maxfev = integer(), maxiter = 50, nprint = 0)
```

Arguments

<code>ftol</code>	non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most <code>ftol</code> . Therefore, <code>ftol</code> measures the relative error desired in the sum of squares.
<code>ptol</code>	non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most <code>ptol</code> . Therefore, <code>ptol</code> measures the relative error desired in the approximate solution.
<code>gtol</code>	non-negative numeric. Termination occurs when the cosine of the angle between result of <code>fn</code> evaluation <i>fvec</i> and any column of the Jacobian is at most <code>gtol</code> in absolute value. Therefore, <code>gtol</code> measures the orthogonality desired between the function vector and the columns of the Jacobian.
<code>diag</code>	a list or numeric vector containing positive entries that serve as multiplicative scale factors for the parameters. Length of <code>diag</code> should be equal to that of <code>par</code> . If not, user-provided <code>diag</code> is ignored and <code>diag</code> is internally set.
<code>epsfcn</code>	(used if <code>jac</code> is not provided) is a numeric used in determining a suitable step for the forward-difference approximation. This approximation assumes that the relative errors in the functions are of the order of <code>epsfcn</code> . If <code>epsfcn</code> is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.
<code>factor</code>	positive numeric, used in determining the initial step bound. This bound is set to the product of <code>factor</code> and the $ \text{diag} * \text{par} $ if nonzero, or else to <code>factor</code> itself. In most cases <code>factor</code> should lie in the interval (0.1,100). 100 is a generally recommended value.
<code>maxfev</code>	integer; termination occurs when the number of calls to <code>fn</code> has reached <code>maxfev</code> . Note that <code>nls.lm</code> sets the value of <code>maxfev</code> to $100 * (\text{length}(\text{par}) + 1)$ if <code>maxfev = integer()</code> , where <code>par</code> is the list or vector of parameters to be optimized.
<code>maxiter</code>	positive integer. Termination occurs when the number of iterations reaches <code>maxiter</code> .
<code>nprint</code>	is an integer; set <code>nprint</code> to be positive to enable printing of iterates

Value

A list with exactly nine components:

`ftol`

ptol
gtol
diag
epsfcn
factor
maxfev
nprint

with meanings as explained under ‘Arguments’.

References

J.J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," in *Lecture Notes in Mathematics* **630**: Numerical Analysis, G.A. Watson (Ed.), Springer-Verlag: Berlin, 1978, pp. 105-116.

See Also

[nls.lm](#)

Examples

```
nls.lm.control(maxiter = 4)
```

nlsLM

Standard 'nls' framework that uses 'nls.lm' for fitting

Description

nlsLM is a modified version of [nls](#) that uses [nls.lm](#) for fitting. Since an object of class 'nls' is returned, all generic functions such as [anova](#), [coef](#), [confint](#), [deviance](#), [df.residual](#), [fitted](#), [formula](#), [logLik](#), [predict](#), [print](#), [profile](#), [residuals](#), [summary](#), [update](#), [vcov](#) and [weights](#) are applicable.

Usage

```
nlsLM(formula, data = parent.frame(), start, jac = NULL,  
      algorithm = "LM", control = nls.lm.control(),  
      lower = NULL, upper = NULL, trace = FALSE, subset,  
      weights, na.action, model = FALSE, ...)
```

Arguments

formula	a nonlinear model formula including variables and parameters. Will be coerced to a formula if necessary.
data	an optional data frame in which to evaluate the variables in formula and weights . Can also be a list or an environment, but not a matrix.
start	a named list or named numeric vector of starting estimates.
jac	A function to return the Jacobian.
algorithm	only method "LM" (Levenberg-Marquardt) is implemented.
control	an optional list of control settings. See nls.lm.control for the names of the settable control values and their effect.
lower	A numeric vector of lower bounds on each parameter. If not given, the default lower bound for each parameter is set to <code>-Inf</code> .
upper	A numeric vector of upper bounds on each parameter. If not given, the default upper bound for each parameter is set to <code>Inf</code> .
trace	logical value indicating if a trace of the iteration progress should be printed. Default is <code>FALSE</code> . If <code>TRUE</code> , the residual (weighted) sum-of-squares and the parameter values are printed at the conclusion of each iteration.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
weights	an optional numeric vector of (fixed) weights. When present, the objective function is weighted least squares.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the <code>na.action</code> setting of options , and is <code>na.fail</code> if that is unset. The 'factory-fresh' default is <code>na.omit</code> . Value <code>na.exclude</code> can be useful.
model	logical. If true, the model frame is returned as part of the object. Default is <code>FALSE</code> .
...	Additional optional arguments. None are used at present.

Details

The standard [nls](#) function was modified in several ways to incorporate the Levenberg-Marquardt type [nls.lm](#) fitting algorithm. The `formula` is transformed into a function that returns a vector of (weighted) residuals whose sum square is minimized by [nls.lm](#). The optimized parameters are then transferred to `stats:::nlsModel` in order to obtain an object of class 'nlsModel'. The internal C function `C_nls_iter` and `stats:::nls_port_fit` were removed to avoid subsequent "Gauss-Newton", "port" or "plinear" types of optimization of `nlsModel`. Several other small modifications were made in order to make all generic functions work on the output.

Value

A list of

m	an <code>nlsModel</code> object incorporating the model.
data	the expression that was passed to <code>nls</code> as the data argument. The actual data values are present in the environment of the <code>m</code> component.

call	the matched call.
convInfo	a list with convergence information.
control	the control list used, see the control argument.
na.action	the "na.action" attribute (if any) of the model frame.
dataClasses	the "dataClasses" attribute (if any) of the "terms" attribute of the model frame.
model	if model = TRUE, the model frame.
weights	if weights is supplied, the weights.

Author(s)

Andrej-Nikolai Spiess and Katharine M. Mullen

References

- Bates, D. M. and Watts, D. G. (1988) *Nonlinear Regression Analysis and Its Applications*, Wiley
- Bates, D. M. and Chambers, J. M. (1992) *Nonlinear models*. Chapter 10 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.
- J.J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," in *Lecture Notes in Mathematics* **630**: Numerical Analysis, G.A. Watson (Ed.), Springer-Verlag: Berlin, 1978, pp. 105-116.

See Also

[nls.lm](#), [nls](#), [nls.lm.control](#), [optim](#)

Examples

```
### Examples from 'nls' doc ###
DNase1 <- subset(DNase, Run == 1)
## using a selfStart model
fm1DNase1 <- nlsLM(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)
## using logistic formula
fm2DNase1 <- nlsLM(density ~ Asym/(1 + exp((xmid - log(conc))/scal)),
                  data = DNase1,
                  start = list(Asym = 3, xmid = 0, scal = 1))

## all generics are applicable
coef(fm1DNase1)
confint(fm1DNase1)
deviance(fm1DNase1)
df.residual(fm1DNase1)
fitted(fm1DNase1)
formula(fm1DNase1)
logLik(fm1DNase1)
predict(fm1DNase1)
print(fm1DNase1)
profile(fm1DNase1)
residuals(fm1DNase1)
```

```

summary(fm1DNase1)
update(fm1DNase1)
vcov(fm1DNase1)
weights(fm1DNase1)

## weighted nonlinear regression using
## inverse squared variance of the response
## gives same results as original 'nls' function
Treated <- Puromycin[Puromycin$state == "treated", ]
var.Treated <- tapply(Treated$rate, Treated$conc, var)
var.Treated <- rep(var.Treated, each = 2)
Pur.wt1 <- nls(rate ~ (Vm * conc)/(K + conc), data = Treated,
              start = list(Vm = 200, K = 0.1), weights = 1/var.Treated^2)
Pur.wt2 <- nlsLM(rate ~ (Vm * conc)/(K + conc), data = Treated,
                start = list(Vm = 200, K = 0.1), weights = 1/var.Treated^2)
all.equal(coef(Pur.wt1), coef(Pur.wt2))

## 'nlsLM' can fit zero-noise data
## in contrast to 'nls'
x <- 1:10
y <- 2*x + 3
## Not run:
nls(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321))

## End(Not run)
nlsLM(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321))

### Examples from 'nls.lm' doc
## values over which to simulate data
x <- seq(0,5, length = 100)
## model based on a list of parameters
getPred <- function(parS, xx) parS$a * exp(xx * parS$b) + parS$c
## parameter values used to simulate data
pp <- list(a = 9, b = -1, c = 6)
## simulated data with noise
simDNoisy <- getPred(pp, x) + rnorm(length(x), sd = .1)
## make model
mod <- nlsLM(simDNoisy ~ a * exp(b * x) + c,
            start = c(a = 3, b = -0.001, c = 1),
            trace = TRUE)

## plot data
plot(x, simDNoisy, main = "data")
## plot fitted values
lines(x, fitted(mod), col = 2, lwd = 2)

## create declining cosine
## with noise
TT <- seq(0, 8, length = 501)
tau <- 2.2
N0 <- 1000
a <- 0.25
f0 <- 8
Ndet <- N0 * exp(-TT/tau) * (1 + a * cos(f0 * TT))

```

```
N <- Ndet + rnorm(length(Ndet), mean = Ndet, sd = .01 * max(Ndet))
## make model
mod <- nlsLM(N ~ N0 * exp(-TT/tau) * (1 + a * cos(f0 * TT)),
             start = c(tau = 2.2, N0 = 1500, a = 0.25, f0 = 10),
             trace = TRUE)

## plot data
plot(TT, N, main = "data")
## plot fitted values
lines(TT, fitted(mod), col = 2, lwd = 2)
```

Index

- *Topic **nonlinear**
 - nls.lm, 1
 - nls.lm.control, 6
 - nlsLM, 8
- *Topic **optimize**
 - nls.lm, 1
 - nls.lm.control, 6
 - nlsLM, 8
- *Topic **regression**
 - nls.lm, 1
 - nls.lm.control, 6
 - nlsLM, 8
- summary, 3, 8
- update, 8
- vcov, 3, 8
- weights, 8
- anova, 8
- coef, 3, 8
- confint, 3, 8
- deviance, 3, 8
- df.residual, 3, 8
- fitted, 8
- formula, 8, 9
- logLik, 8
- na.exclude, 9
- na.fail, 9
- na.omit, 9
- nls, 4, 8–10
- nls.lm, 1, 8–10
- nls.lm.control, 2, 4, 6, 9, 10
- nlsLM, 8
- optim, 4, 10
- options, 9
- predict, 8
- print, 3, 8
- profile, 8
- residuals, 3, 8