|  |  |
| --- | --- |
| nls {stats} | R Documentation |

**Nonlinear Least Squares**

**Description**

Determine the nonlinear (weighted) least-squares estimates of the parameters of a nonlinear model.

**Usage**

nls(formula, data, start, control, algorithm,

trace, subset, weights, na.action, model,

lower, upper, ...)

**Arguments**

|  |  |
| --- | --- |
| formula | a nonlinear model [formula](http://127.0.0.1:14695/library/stats/help/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. When start is missing, a very cheap guess for start is tried (if algorithm != "plinear"). |
| control | an optional list of control settings. See [nls.control](http://127.0.0.1:14695/library/stats/help/nls.control) for the names of the settable control values and their effect. |
| algorithm | character string specifying the algorithm to use. The default algorithm is a Gauss-Newton algorithm. Other possible values are "plinear" for the Golub-Pereyra algorithm for partially linear least-squares models and "port" for the ‘nl2sol’ algorithm from the Port library – see the references. Can be abbreviated. |
| trace | logical value indicating if a trace of the iteration progress should be printed. Default is FALSE. If TRUE the residual (weighted) sum-of-squares and the parameter values are printed at the conclusion of each iteration. When the "plinear" algorithm is used, the conditional estimates of the linear parameters are printed after the nonlinear parameters. When the "port" algorithm is used the objective function value printed is half the residual (weighted) sum-of-squares. |
| 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 na.action setting of [options](http://127.0.0.1:14695/library/stats/help/options), and is [na.fail](http://127.0.0.1:14695/library/stats/help/na.fail) if that is unset. The ‘factory-fresh’ default is [na.omit](http://127.0.0.1:14695/library/stats/help/na.omit). Value [na.exclude](http://127.0.0.1:14695/library/stats/help/na.exclude) can be useful. |
| model | logical. If true, the model frame is returned as part of the object. Default is FALSE. |
| lower, upper | vectors of lower and upper bounds, replicated to be as long as start. If unspecified, all parameters are assumed to be unconstrained. Bounds can only be used with the "port" algorithm. They are ignored, with a warning, if given for other algorithms. |
| ... | Additional optional arguments. None are used at present. |

**Details**

An nls object is a type of fitted model object. It has methods for the generic functions [anova](http://127.0.0.1:14695/library/stats/help/anova), [coef](http://127.0.0.1:14695/library/stats/help/coef), [confint](http://127.0.0.1:14695/library/stats/help/confint), [deviance](http://127.0.0.1:14695/library/stats/help/deviance), [df.residual](http://127.0.0.1:14695/library/stats/help/df.residual), [fitted](http://127.0.0.1:14695/library/stats/help/fitted), [formula](http://127.0.0.1:14695/library/stats/help/formula), [logLik](http://127.0.0.1:14695/library/stats/help/logLik), [predict](http://127.0.0.1:14695/library/stats/help/predict), [print](http://127.0.0.1:14695/library/stats/help/print), [profile](http://127.0.0.1:14695/library/stats/help/profile), [residuals](http://127.0.0.1:14695/library/stats/help/residuals), [summary](http://127.0.0.1:14695/library/stats/help/summary), [vcov](http://127.0.0.1:14695/library/stats/help/vcov) and [weights](http://127.0.0.1:14695/library/stats/help/weights).

Variables in formula (and weights if not missing) are looked for first in data, then the environment of formula and finally along the search path. Functions in formula are searched for first in the environment of formula and then along the search path.

Arguments subset and na.action are supported only when all the variables in the formula taken from data are of the same length: other cases give a warning.

Note that the [anova](http://127.0.0.1:14695/library/stats/help/anova) method does not check that the models are nested: this cannot easily be done automatically, so use with care.

**Value**

A list of

|  |  |
| --- | --- |
| m | an nlsModel object incorporating the model. |
| data | the expression that was passed to nls as the data argument. The actual data values are present in the environment of the m component. |
| call | the matched call with several components, notably algorithm. |
| 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. |
| convInfo | a list with convergence information. |
| control | the control list used, see the control argument. |
| convergence, message | for an algorithm = "port" fit only, a convergence code (0 for convergence) and message.  To use these is *deprecated*, as they are available from convInfo now. |

**Warning**

**Do not use nls on artificial "zero-residual" data.**

The nls function uses a relative-offset convergence criterion that compares the numerical imprecision at the current parameter estimates to the residual sum-of-squares. This performs well on data of the form

*y = f(x, θ) + eps*

(with var(eps) > 0). It fails to indicate convergence on data of the form

*y = f(x, θ)*

because the criterion amounts to comparing two components of the round-off error. If you wish to test nls on artificial data please add a noise component, as shown in the example below.

The algorithm = "port" code appears unfinished, and does not even check that the starting value is within the bounds. Use with caution, especially where bounds are supplied.

**Note**

Setting warnOnly = TRUE in the control argument (see [nls.control](http://127.0.0.1:14695/library/stats/help/nls.control)) returns a non-converged object (since **R** version 2.5.0) which might be useful for further convergence analysis, *but* ***not*** *for inference*.

**Author(s)**

Douglas M. Bates and Saikat DebRoy: David M. Gay for the Fortran code used by algorithm = "port".

**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.

<http://www.netlib.org/port/> for the Port library documentation.

**See Also**

[summary.nls](http://127.0.0.1:14695/library/stats/help/summary.nls), [predict.nls](http://127.0.0.1:14695/library/stats/help/predict.nls), [profile.nls](http://127.0.0.1:14695/library/stats/help/profile.nls).

Self starting models (with ‘automatic initial values’): [selfStart](http://127.0.0.1:14695/library/stats/help/selfStart).

**Examples**

require(graphics)

DNase1 <- subset(DNase, Run == 1)

## using a selfStart model

fm1DNase1 <- nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)

summary(fm1DNase1)

## the coefficients only:

coef(fm1DNase1)

## including their SE, etc:

coef(summary(fm1DNase1))

## using conditional linearity

fm2DNase1 <- nls(density ~ 1/(1 + exp((xmid - log(conc))/scal)),

data = DNase1,

start = list(xmid = 0, scal = 1),

algorithm = "plinear")

summary(fm2DNase1)

## without conditional linearity

fm3DNase1 <- nls(density ~ Asym/(1 + exp((xmid - log(conc))/scal)),

data = DNase1,

start = list(Asym = 3, xmid = 0, scal = 1))

summary(fm3DNase1)

## using Port's nl2sol algorithm

fm4DNase1 <- nls(density ~ Asym/(1 + exp((xmid - log(conc))/scal)),

data = DNase1,

start = list(Asym = 3, xmid = 0, scal = 1),

algorithm = "port")

summary(fm4DNase1)

## weighted nonlinear regression

Treated <- Puromycin[Puromycin$state == "treated", ]

weighted.MM <- function(resp, conc, Vm, K)

{

## Purpose: exactly as white book p. 451 -- RHS for nls()

## Weighted version of Michaelis-Menten model

## ----------------------------------------------------------

## Arguments: 'y', 'x' and the two parameters (see book)

## ----------------------------------------------------------

## Author: Martin Maechler, Date: 23 Mar 2001

pred <- (Vm \* conc)/(K + conc)

(resp - pred) / sqrt(pred)

}

Pur.wt <- nls( ~ weighted.MM(rate, conc, Vm, K), data = Treated,

start = list(Vm = 200, K = 0.1))

summary(Pur.wt)

## Passing arguments using a list that can not be coerced to a data.frame

lisTreat <- with(Treated,

list(conc1 = conc[1], conc.1 = conc[-1], rate = rate))

weighted.MM1 <- function(resp, conc1, conc.1, Vm, K)

{

conc <- c(conc1, conc.1)

pred <- (Vm \* conc)/(K + conc)

(resp - pred) / sqrt(pred)

}

Pur.wt1 <- nls( ~ weighted.MM1(rate, conc1, conc.1, Vm, K),

data = lisTreat, start = list(Vm = 200, K = 0.1))

stopifnot(all.equal(coef(Pur.wt), coef(Pur.wt1)))

## Chambers and Hastie (1992) Statistical Models in S (p. 537):

## If the value of the right side [of formula] has an attribute called

## 'gradient' this should be a matrix with the number of rows equal

## to the length of the response and one column for each parameter.

weighted.MM.grad <- function(resp, conc1, conc.1, Vm, K)

{

conc <- c(conc1, conc.1)

K.conc <- K+conc

dy.dV <- conc/K.conc

dy.dK <- -Vm\*dy.dV/K.conc

pred <- Vm\*dy.dV

pred.5 <- sqrt(pred)

dev <- (resp - pred) / pred.5

Ddev <- -0.5\*(resp+pred)/(pred.5\*pred)

attr(dev, "gradient") <- Ddev \* cbind(Vm = dy.dV, K = dy.dK)

dev

}

Pur.wt.grad <- nls( ~ weighted.MM.grad(rate, conc1, conc.1, Vm, K),

data = lisTreat, start = list(Vm = 200, K = 0.1))

rbind(coef(Pur.wt), coef(Pur.wt1), coef(Pur.wt.grad))

## In this example, there seems no advantage to providing the gradient.

## In other cases, there might be.

## The two examples below show that you can fit a model to

## artificial data with noise but not to artificial data

## without noise.

x <- 1:10

y <- 2\*x + 3 # perfect fit

yeps <- y + rnorm(length(y), sd = 0.01) # added noise

nls(yeps ~ a + b\*x, start = list(a = 0.12345, b = 0.54321))

## terminates in an error, because convergence cannot be confirmed:

try(nls(y ~ a + b\*x, start = list(a = 0.12345, b = 0.54321)))

## the nls() internal cheap guess for starting values can be sufficient:

x <- -(1:100)/10

y <- 100 + 10 \* exp(x / 2) + rnorm(x)/10

nlmod <- nls(y ~ Const + A \* exp(B \* x))

plot(x,y, main = "nls(\*), data, true function and fit, n=100")

curve(100 + 10 \* exp(x / 2), col = 4, add = TRUE)

lines(x, predict(nlmod), col = 2)

## The muscle dataset in MASS is from an experiment on muscle

## contraction on 21 animals. The observed variables are Strip

## (identifier of muscle), Conc (Cacl concentration) and Length

## (resulting length of muscle section).

utils::data(muscle, package = "MASS")

## The non linear model considered is

## Length = alpha + beta\*exp(-Conc/theta) + error

## where theta is constant but alpha and beta may vary with Strip.

with(muscle, table(Strip)) # 2, 3 or 4 obs per strip

## We first use the plinear algorithm to fit an overall model,

## ignoring that alpha and beta might vary with Strip.

musc.1 <- nls(Length ~ cbind(1, exp(-Conc/th)), muscle,

start = list(th = 1), algorithm = "plinear")

summary(musc.1)

## Then we use nls' indexing feature for parameters in non-linear

## models to use the conventional algorithm to fit a model in which

## alpha and beta vary with Strip. The starting values are provided

## by the previously fitted model.

## Note that with indexed parameters, the starting values must be

## given in a list (with names):

b <- coef(musc.1)

musc.2 <- nls(Length ~ a[Strip] + b[Strip]\*exp(-Conc/th), muscle,

start = list(a = rep(b[2], 21), b = rep(b[3], 21), th = b[1]))

summary(musc.2)