Package: nls2 (via r-universe)

October 12, 2024

Version 0.3-4

Date 2024-07-15

Title Non-Linear Regression with Brute Force

Author G. Grothendieck, R Core Team (nls)

Maintainer G. Grothendieck <ggrothendieck@gmail.com>

Description Adds brute force and multiple starting values to nls.

Depends proto, stats

Suggests nlstools, lhs, CPoptim

License GPL-2

BugReports https://github.com/ggrothendieck/nls2/issues

URL https://github.com/ggrothendieck/nls2

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

RemoteUrl https://github.com/ggrothendieck/nls2

RemoteRef HEAD

RemoteSha b8022a0cfad1ab16f06f225b3614aa4ac74e4d32

Contents

nls2 *Nonlinear Least Squares with Brute Force*

Description

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

Usage

```
nls2(formula, data = parent.frame(), start, control = nls.control(),
 algorithm = c("default", "plinear", "port", "brute-force",
 "grid-search", "random-search", "lhs", "CPoptim",
 "plinear-brute-force", "plinear-random", "plinear-lhs"),
 trace = FALSE, weights, subset, ..., all = FALSE)
```

Arguments

formula	same as formula parameter in nls.
data	same as data parameter in nls except that if subset is specified then data is not optional and must be specified as a data.frame.
start	same as start parameter in nls except that it may alternately be (1) a two row data frame in which case nls2 will start at each point on a grid chosen to have maxiter iterations if "algorithm" is "brute-force" or "grid-search" or will start at maxiter random points within the defined rectangle, (2) a data frame with more than two rows in which case an optimization will be run with the starting value defined by each row successively or (3) it may be an nls or other object having a coef method in which case the coef of the object will be used as the starting value. The list of vectors format supported in nls for grouped data is not supported.
control	same as control parameter in nls.
algorithm	same as algorithm parameter in nls with the addition of the "brute-force" (alternately called "grid-search"), "random-search", "lhs" (Latin Hypercube Sampling), "plinear-brute", "plinear-random" and "plinear-lhs" options.
trace	If TRUE certain intermediate results shown.
weights	For weighted regression.
subset	Subset argument as in nls
...	other arguments passed to nls.
all	if all is true then a list of nls objects is returned, one for each row in start; otherwise, only the one with least residual sum of squares is returned.

Details

Similar to nls except that start and algorithm have expanded values and there is a new all argument.

nls2 generates a grid or random set of starting values and then optionally performs an nls optimization starting at each one.

If algorithm is "brute-force" (or its synonym "grid-search") then (1) if start is a two row data frame then a grid is created from the rectangle defined by the two rows such that the grid has at most maxiter points with the residuals sum of squares being calculated at each generated point. (2) If start is a data frame with more than two rows then the residual sum of squares is evaluated at each row.

If algorithm is "random-search" then (1) if start is a two row data frame then maxiter points are uniformly sampled from the rectangle it defines or (2) if start is a data frame with more than two rows then the "maxiter" rows are sampled without replacement.

"plinear-brute" and "plinear-random" are like "brute-force" and "random-search" except that the formula is a plinear-style formula and only starting values for the non-linear parameters are given.

If algorithm is neither of the above two values then if start has more than one row a two phase procedure is undertaken. (1) if start is a two row data frame then a random set of points is generated and then the optimization is carried out starting from each of those points.

(2) If start is a data frame with more than two rows then the optimization is carried out starting from each row.

In any of the above cases if all=FALSE, the default, then an "nls" object at the value with the least residual sum of squares returned; otherwise, if all=TRUE then a list of "nls" objects is returned with one component per starting value.

If the starting value is an "nls" object then the coef of that object will be used as the starting value.

If the algorithm argument is "CPoptim" then the convex partition algorithm from the CPoptim package is used. In that case start must be a 2 row data frame with the first row being the lower bounds and second row the upper bounds and the column names must be the names of the corresponding parameters. The maximum number of function evaluations (FEmax) defaults to 5000 times the number of parameters and the sample size (sampleSize defaults to 1000. Other values can be passed via the control list but normally the defaults should be adequate.

Since nls is used to produce starting values rather than the final estimates confint cannot be used on objects produced by nls.

See Also

nls.

Examples

```
y <- c(44,36,31,39,38,26,37,33,34,48,25,22,44,5,9,13,17,15,21,10,16,22,
13,20,9,15,14,21,23,23,32,29,20,26,31,4,20,25,24,32,23,33,34,23,28,30,10,29,
40,10,8,12,13,14,56,47,44,37,27,17,32,31,26,23,31,34,37,32,26,37,28,38,35,27,
34,35,32,27,22,23,13,28,13,22,45,33,46,37,21,28,38,21,18,21,18,24,18,23,22,
38,40,52,31,38,15,21)
```

```
x <- c(26.22,20.45,128.68,117.24,19.61,295.21,31.83,30.36,13.57,60.47,
205.30,40.21,7.99,1.18,5.40,13.37,4.51,36.61,7.56,10.30,7.29,9.54,6.93,12.60,
2.43,18.89,15.03,14.49,28.46,36.03,38.52,45.16,58.27,67.13,92.33,1.17,
29.52,84.38,87.57,109.08,72.28,66.15,142.27,76.41,105.76,73.47,1.71,305.75,
325.78,3.71,6.48,19.26,3.69,6.27,1689.67,95.23,13.47,8.60,96.00,436.97,
472.78,441.01,467.24,1169.11,1309.10,1905.16,135.92,438.25,526.68,88.88,31.43,
21.22,640.88,14.09,28.91,103.38,178.99,120.76,161.15,137.38,158.31,179.36,
214.36,187.05,140.92,258.42,85.86,47.70,44.09,18.04,127.84,1694.32,34.27,
75.19,54.39,79.88,63.84,82.24,88.23,202.66,148.93,641.76,20.45,145.31,
27.52,30.70)

## Example 1
## brute force followed by nls optimization

fo <- y ~ Const + B * (x ^ A)

# pass our own set of starting values
# returning result of brute force search as nls object
st1 <- expand.grid(Const = seq(-100, 100, len = 4),
 B = seq(-100, 100, len = 4), A = seq(-1, 1, len = 4))
mod1 <- nls2(fo, start = st1, algorithm = "brute-force")
mod1
# use nls object mod1 just calculated as starting value for
# nls optimization.  Same as: nls(fo, start = coef(mod1))
nls2(fo, start = mod1)

## Example 2

# pass a 2-row data frame and let nls2 calculate grid
st2 <- data.frame(Const = c(-100, 100), B = c(-100, 100), A = c(-1, 1))
mod2 <- nls2(fo, start = st2, algorithm = "brute-force")
mod2
# use nls object mod1 just calculated as starting value for
# nls optimization.  Same as: nls(fo, start = coef(mod2))
nls2(fo, start = mod2)

## Example 3

# Create same starting values as in Example 2
# running an nls optimization from each one and picking best.
# This one does an nls optimization for every random point
# generated whereas Example 2 only does a single nls optimization
nls2(fo, start = st2, control = nls.control(warnOnly = TRUE))

## Example 4

# Investigate singular jacobian at the start value
# Note that this cannot be done with nls since the singular jacobian at
#  the initial conditions would stop it with an error.

DF1 <- data.frame(y=1:9, one=rep(1,9))
xx <- nls2(y~(a+2*b)*one, DF1, start = c(a=1, b=1), algorithm = "brute-force")
svd(xx$m$Rmat())[-2]
```

```
## Example 5

# plinear-lhs example
# Thanks to John Nash for suggesting this truncation of the
# Ratkowsky2 dataset. Full dataset: data(Ratkowsky2, package = "NISTnls")
# Use plinear-lhs to get starting values and then run nls via nls2 for
# final answer.

pastured <- data.frame(
  time=c(9, 14, 21, 28, 42, 57, 63, 70, 79),
  yield= c(8.93, 10.8, 18.59, 22.33, 39.35, 56.11, 61.73, 64.62, 67.08))
fo <- yield ~ cbind(1, - exp(-exp(t3+t4*log(time))))

gstart <- data.frame(t3 = c(-10, 10), t4 = c(1, 8))
set.seed(123)
junk <- capture.output(fm0 <- nls2(fo, data = pastured, start = gstart, alg = "plinear-lhs",
   control = nls.control(maxiter = 1000)), type = "message")
nls2(fo, pastured, start = fm0, alg = "plinear")

## Example 6

# CPoptim example
nls2(demand ~ a + b * Time, data = BOD, start =
   data.frame(a = c(-10, 10), b = c(-10, 10)), alg = "CPoptim")
```

Index