Package ‘nls2’

February 14, 2012

Version 0.1-3

Date 2010-08-22

Title Non-linear regression with brute force

Author G. Grothendieck

Maintainer G. Grothendieck <ggrothendieck@gmail.com>

Description Adds algorithm=‘‘brute-force’’ and multiple starting values to nls.

Depends proto

Suggests nlstools

License GPL-2

URL http://nls2.googlecode.com

Repository CRAN

Date/Publication 2010-08-23 09:24:17

R topics documented:

as.lm.nls *Linearize an nls object.*

Description

Given an "nls" object, this method produces an "lm" object which approximates it.

Usage

```
## S3 method for class 'nls'
as.lm(object, ...)
```

Arguments

object

...

Details

Given an object of class "nls", as.lm.nls produces an "lm" object tangent to the "nls" object which approximates it.

Value

The returned value is an "lm" class object.

References

Bates, DM and DG Watts, Nonlinear Regression Analysis and its Applications, Wiley, 1988.

See Also

nls.

Examples

```
# data is from ?nls
DNase1 <- subset(DNase, Run == 1)
fm1DNase1 <- nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)

# these give same result
vcov(fm1DNase1)
vcov(as.lm(fm1DNase1))

# nls confidence and prediction intervals based on asymptotic approximation
# are same as as.lm confidence intervals.
predict(as.lm(fm1DNase1), interval = "confidence")
predict(as.lm(fm1DNase1), interval = "prediction")
```

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"), ..., all = FALSE)
```

Arguments

`formula`	same as `formula` parameter in `nls`.
`data`	same as `data` parameter in `nls`.
`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` object in which case the `coef` of the object will be used as the starting value.
`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"`) and `"random-search"` options.
`...`	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.

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.

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 gradient.
# Note that this cannot be done with nls since the singular gradient 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

# Use plinear algorithm to reduce a 4 parameter model to a model with
# 2 linear and 2 nonlinear parameters

## Not run:
data(Ratkowsky, package = "NISTnls") # Ratkowsky2 data set
# fo corresponds to the model on page 13 of Huet et al.
fo <- y ~ cbind(rep(1, 9), exp(- exp(p3+p4*log(x))))
st <- data.frame(p3 = c(-100,100), p4 = c(-100, 100))
rat.nls <- nls2(fo, Ratkwosky2, start = st,
control = nls.control(maxiter = 200), algorithm = "plinear")
rat.nls
rat2.nls <- nls2(fo, Ratkowsky2, start = rat.nls, algorithm = "plinear")
rat2.nls

## End(Not run)
```

Index