Refactoring the nls() function in R

John C Nash, retired professor, University of Ottawa
Arkajyoti Bhattacharjee, Indian Institute of Technology, Kanpur

2021-7-17

Contents

Abstract

This article reports the particular activities of our Google Summer of Code project "Improvements to nls()" that relate to R code for that function, which is intended for the estimation of models written as a formula that has at least one parameter that is not estimable via solving a set of linear equations. A companion document "Variety in Nonlinear Least Squares Codes" presents an overview of methods for the problem which takes a much wider view of the problem of minimizing a function that can be written as a sum of squared terms.

Our work has not fully addressed all the issues that we would like to see resolved, but we believe we have made sufficient progress to demonstrate that there are worthwhile improvements that can be made to the R function `nls()`.

The existing function and its shortcomings

`nls()` is the tool in base R (the distributed software package from https://cran.r-project.org) for estimating nonlinear statistical models. The function was developed mainly in the 1980s and 1990s by Doug Bates et al., initially for S (https://en.wikipedia.org/wiki/S_%28programming_language%29). The ideas spring primarily from the book by D. M. Bates and Watts (1988).

The `nls()` function has a remarkable and quite comprehensive set of capabilities for estimating nonlinear models that are expressed as formulas. In particular, we note that it - handles formulas that include R functions - allows data to be subset - permits parameters to be indexed over a set of related data - produces measures of variability (i.e., standard error estimates) for the estimated parameters - has related profiling capabilities

With such a range of features and a long history, it is not surprising that code has become untidy and overly patched. It is, to our mind, essentially unmaintainable. Moreover, it has deficiencies that can and should be fixed. Let us review some of the issues. We will then propose corrective actions, some of which we have carried out.

Issue: Convergence and termination tests

Within the standard documentation (**man**ual or ".Rd" file) `nls()` warns

> **The default settings of nls generally fail on artificial "zero-residual" data problems.**
>
> 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
>
> $$y = f(x, \theta) + eps$$
>
> (with var(eps) > 0). It fails to indicate convergence on data of the form
>
> $$y = f(x, \theta)$$
>
> because the criterion amounts to comparing two components of the round-off error. To avoid a zero-divide in computing the convergence testing value, a positive constant scaleOffset should be added to the denominator sum-of-squares; it is set in control; this does not yet apply to algorithm = "port."

It turns out that this issue can be quite easily resolved. The key "convergence test" – more properly a "termination test" for the **program** rather than testing for convergence of the underlying **algorithm** – is the Relative Offset Convergence Criterion (see Douglas M. Bates and Watts (1981)). This works by projecting the proposed step in the parameter vector on the gradient and estimating how much the sum of squares loss function will decrease. To avoid scale issues, we use the current size of the loss function as a measure and divide by it. When we have "converged," the estimated decrease is very small, as usually is its ratio to the sum of squares. However, in some cases we have the possibility of an exact fit and the sum of squares is (almost) zero and we get the possibility of a zero-divide failure.

The issue is easily resolved by adding a small quantity to the loss function. To preserve legacy behaviour, in 2021, one of us (JN) proposed that `nls.control()` have an additional parameter `scaleOffset` with a default value of zero for legacy behaviour. Setting it to a small number – 1.0 is a reasonable choice – allows small-residual problems (i.e., near-exact fits) to be dealt with easily. We call this the **safeguarded relative offset convergence criterion**.

Example of a small-residual problem

```
rm(list=ls())
t <- -10:10
```

```
y <- 100/(1+.1*exp(-0.51*t))
lform<-y~a/(1+b*exp(-c*t))
ldata<-data.frame(t=t, y=y)
plot(t,y)
lstartbad<-c(a=1, b=1, c=1)
lstart2<-c(a=100, b=10, c=1)
nlsr::nlxb(lform, data=ldata, start=lstart2)
nls(lform, data=ldata, start=lstart2, trace=TRUE)
# Fix with scaleOffset
nls(lform, data=ldata, start=lstart2, trace=TRUE, control=list(scaleOffset=1.0))
sessionInfo()
```

Edited output of running this function follows:

```
> rm(list=ls())
> t <- -10:10
> y <- 100/(1+.1*exp(-0.51*t))
> lform<-y~a/(1+b*exp(-c*t))
> ldata<-data.frame(t=t, y=y)
> plot(t,y)
> lstart2<-c(a=100, b=10, c=1)
> nlsr::nlxb(lform, data=ldata, start=lstart2)
nlsr object: x
residual sumsquares =  1.007e-19  on  21 observations
    after  13    Jacobian and  19 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
a                   100     2.679e-11  3.732e+12  1.863e-216  -6.425e-11       626.6
b                   0.1      3.78e-13  2.646e+11  9.125e-196  -3.393e-08       112.3
c                  0.51       6.9e-13  7.391e+11  8.494e-204   1.503e-08       2.791
# Note that this has succeeded. The test in nlsr recognizes small residual problems.
> nls(lform, data=ldata, start=lstart2, trace=TRUE)
40346.    (1.08e+00): par = (100 10 1)
11622.    (2.93e+00): par = (101.47 0.49449 0.71685)
5638.0    (1.08e+01): par = (102.23 0.38062 0.52792)
642.08    (1.04e+01): par = (102.16 0.22422 0.41935)
97.712    (1.79e+01): par = (100.7 0.14774 0.45239)
22.250    (1.78e+02): par = (99.803 0.093868 0.50492)
0.025789  (1.33e+03): par = (100.01 0.10017 0.50916)
6.0571e-08 (7.96e+05): par = (100 0.1 0.51)
4.7017e-19 (1.86e+04): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
```

```
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
1.2440e-27 (5.71e-01): par = (100 0.1 0.51)
Error in nls(lform, data = ldata, start = lstart2, trace = TRUE) :
  number of iterations exceeded maximum of 50

> nls(lform, data=ldata, start=lstart2, trace=TRUE, control=list(scaleOffset=1.0))
40346.      (1.08e+00): par = (100 10 1)
11622.      (2.91e+00): par = (101.47 0.49449 0.71685)
5638.0      (9.23e+00): par = (102.23 0.38062 0.52792)
642.08      (5.17e+00): par = (102.16 0.22422 0.41935)
97.712      (2.31e+00): par = (100.7 0.14774 0.45239)
22.250      (1.11e+00): par = (99.803 0.093868 0.50492)
0.025789    (3.79e-02): par = (100.01 0.10017 0.50916)
6.0571e-08  (5.80e-05): par = (100 0.1 0.51)
4.7017e-19  (1.62e-10): par = (100 0.1 0.51)
Nonlinear regression model
  model: y ~ a/(1 + b * exp(-c * t))
   data: ldata
     a      b      c
100.00   0.10   0.51
 residual sum-of-squares: 4.7e-19

Number of iterations to convergence: 8
Achieved convergence tolerance: 1.62e-10

> sessionInfo()
R version 4.1.0 (2021-05-18)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Linux Mint 20.2
```

Issue: more general termination tests

The single convergence criterion of `nls()` leaves out some possibilities that could be useful for some problems. The package `nlsr` (Nash and Murdoch (2019)) already offers both the safeguarded relative offset test (**roffset**) as well as a **small sum of squares** test (**smallsstest**) that compares the latest evaluated sum of squared (weighted) residuals to a very small multiple of the initial sum of squares. The multiple uses a control setting `offset` which defaults to 100.0 and we compute the 4th power of the machine epsilon times this offset.

```
epstol<-100*.Machine$double.eps
e4 <- epstol^4
e4
```

```
## [1] 2.430865e-55
```

We do note that `nls()` stops after `maxiter` "iterations." However, for almost all iterative algorithms, the meaning of "iteration" requires careful examination of the code. Instead, we prefer to record the number of times the residuals or the jacobian have been computed and put upper limits on these. Our codes exit (terminate) when these limits are reached. Generally we prefer larger limits than the default `maxiter=50` of `nls()`, but that may simply reflect our history of dealing with more difficult problems as we are the tool-makers users consult when things go wrong.

Issue: Failure when Jacobian is computationally singular

This is the infamous "singular gradient" termination. In some cases it is due to failure of the simple finite difference approximation of the Jacobian in the `numericDeriv()` function that is a part of `nls()`. `nlsr` has used analytic derivatives, and we can import this functionality to the `nls()` code.

However, the more common source of the issue is that the Jacobian is very close to singular for some values of the model parameters. In such cases we need to find an alternative algorithm to the Gauss-Newton iteration of `nls()`. The most common work-around is the Levenberg-Marquardt stabilization (Marquardt (1963), Levenberg1944, jn77ima). Versions of this have been implemented in packages `minpack.lm` and `nlsr`.

Issue: Jacobian computation

analytic in nlsr, `numericDeriv()`, others

Issue: Subsetting

`nls()` accepts an argument `subset`. Unfortunately, this acts through the mediation of `model.frame` and is not clearly obvious in the source code files `/src/library/stats/R/nls.R` and `/src/library/stats/src/nls.C`.

- implementation via weights
- implementation via model.frame
- other concerns

Issue: na.action

`na.action` is an argument to the `nls()` function, but it does not appear in obviously in the source code ...

Issue: model

`model` is an argument to the `nls()` function, but it does not appear in obviously in the source code ...

Issue: sources of data

`nls()` can be called without specifying the `data` argument. In this case, it will search in the available environments (i.e., workspaces) for suitable data objects. We do NOT like this approach. R allows users to

leave many objects in the default (.GlobalEnv) workspace. Moreover, users have to actively suppress saving this workspace (`.RData`) on exit, and any such file in the path when R is launched will be loaded.

Nevertheless, to provide compatible behaviour with `nls()`, we will need to ensure that equivalent behaviour is guaranteed.

Issue: missing start vector and self-starting models

Nonlinear estimation algorithms are almost all iterative and need a set of starting parameters. `nls()` offers a special class of modeling formulae called **selfStart** models. There are a number of these in base R (??list) and others in R packages such as (?? list or examples). Unfortunately, the structure of the programming of these is such that the methods by which initial parameters are computed is entangled with the particularities of the `nls()` code. Though there is a `getInitial()` function, this is not easy to use to simply compute the initial parameter estimates.

?? weird output in testing

```
> ls()
[1] "ldata"     "lform"     "lstart2"   "lstartbad" "t"         "y"
> apar <- getInitial(y~SSlogis(t, Asym, xmid, scal), data=ldata)
Error in nls(y ~ 1/(1 + exp((xmid - x)/scal)), data = xy, start = list(xmid = aux[[1L]],  :
  number of iterations exceeded maximum of 50
```

In the event that a selfStart model is not available, `nls()` sets all the starting parameters to 1. This is, in our view, tolerable, but could possibly be improved by using a set of values that are slightly different e.g., in the case of a model

$$y\tilde{a} * exp(-b * x) + c * exp(-d * x)$$

it would be useful to have b and d values different so the Jacobian is not singular. Thus some sort of sequence like 1.0, 1.1, 1.2, 1.3 for the four parameters might be better and it can be provided quite simply instead of all 1s.

Issue: results of running nls()

The output of nls() is an object of class "nls" which has the following structure:

?? put in an example and document it.

Concerns with content of the nls object

The nls object contains some elements that are awkward to produce by other algorithms. Moreover, some information that would be useful is not presented obviously (??examples - convergence/termination info, Jsingvals)

Issue: code structure

The `nls()` code is structured in a way that inhibits both maintenance and improvement. In particular, the iterative setup is such that introduction of Marquardt stabilization is not easily available.

To obtain performance, a lot of the code is in C with consequent calls and returns that complicate the code. Over time, R has become much more efficient on modern computers, and the need to use compiled C and Fortran is less critical. Moreover, the burden for maintenance could be much reduced by moving code entirely to R.

Issue: code documentation

`setPars()` – explain weaknesses. Only used by profile.nls()

The paucity of documentation is exacerbated by the mixed R/C/Fortran code base.

Following is an email to Dr. Heather Turner from John Nash.

I'm afraid that I don't know the purpose of the recursive call either. I know that I wrote the code to
for the response, covariates, etc., but I don't recall anything like a recursive call being necessary.

If the R sources were in a git repository I might try to use `git blame` to find out when and by whom t
but they are in an SVN repository, I think, and I haven't used it for a long, long time.

I don't think I will be of much help. My R skills have atrophied to the point where I wouldn't even kn
exploring what is happening in the first call as opposed to the recursive call.

On Tue, Jun 29, 2021 at 11:50 AM John Nash <Nashjc@uottawa.ca <mailto:Nashjc@uottawa.ca>> wrote:

> Thanks.
>
> https://gitlab.com/nashjc/improvenls/-/blob/master/Croucher-expandednlsnoc.R
> <https://gitlab.com/nashjc/improvenls/-/blob/master/Croucher-expandednlsnoc.R>
>
> This has the test problem and the expanded code. Around line 367 is where we are
> scratching our heads. The function code (from nlsModel()) is in the commented lines below
> the call. This is

```
# > setPars
# function(newPars) {
#   setPars(newPars)
#   resid <<- .swts * (lhs - (rhs <<- getRHS())) # envir = thisEnv {2 x}
#   dev   <<- sum(resid^2) # envir = thisEnv
#   if(length(gr <- attr(rhs, "gradient")) == 1L) gr <- c(gr)
#   QR <<- qr(.swts * gr) # envir = thisEnv
#   (QR$rank < min(dim(QR$qr))) # to catch the singular gradient matrix
# }
```

> I'm anticipating that we will be able to set up a (possibly inefficient) code
> with documentation that will be easier to follow and test, then gradually figure
> out how to make it more efficient.
>
> The equivalent from minpack.lm is

```
setPars = function(newPars) {
            setPars(newPars)
            assign("resid", .swts * (lhs - assign("rhs", getRHS(),
                envir = thisEnv)), envir = thisEnv)
            assign("dev", sum(resid^2), envir = thisEnv)
            assign("QR", qr(.swts * attr(rhs, "gradient")), envir = thisEnv)
            return(QR$rank < min(dim(QR$qr)))
        }
```

> In both there is the recursive call, which must have a purpose I don't understand.

```
Cheers, JN

On 2021-06-29 12:33 p.m., Douglas Bates wrote:
> *Attention : courriel externe | external email*
> Thanks for contacting me, John.  Can you point me to a file in the gitlab.com <http://gitlab.com>
<http://gitlab.com <http://gitlab.com>> repository that
> contains the definition of setPars?
>
> (By the way, it is probably best to use the email address dmbates@gmail.com <mailto:dmbates@gmail
<mailto:dmbates@gmail.com <mailto:dmbates@gmail.com>> for me.  If email
> goes to bates@stat.wisc.edu <mailto:bates@stat.wisc.edu> <mailto:bates@stat.wisc.edu <mailto:bates
it should get forwarded to the gmail.com <http://gmail.com> <http://gmail.com <http://gmail.com>>
> address but sometimes gmail decides that such mail looks suspicious and puts it in the spam folde
why.  For
> a long time I used bates@stat.wisc.edu <mailto:bates@stat.wisc.edu> <mailto:bates@stat.wisc.edu
<mailto:bates@stat.wisc.edu>> as my "From:" address because it had been my address
> since the 80's but even gmail got suspicious of mail from that address that did not appear to orig
wisc.edu <http://wisc.edu>
> <http://wisc.edu <http://wisc.edu>> domain.)
>
```

Goals of our effort

What we want to accomplish.

Code rationalization and documentation

We want

- to provide a packaged version of `nls()` (call it `nlsalt`) all in R that matches the version in base R and what is packaged in `nlspkg` as described in the "PkgFromRbase" document
- try to obtain cleaner structure for the overall `nls()` infrastructure. By this we mean a re-factoring of the routines so they are better suited to maintenance of both the existing `nls()` methods and features as well as the new features we would like to add.
- try to explain what we do, either in comments or separate maintainer documentation. Since we are complaining about the lack of explanatory material for the current code, we feel it incumbent on us to provide such material for our own work, and if possible for the existing code.

Provide tests

- to ensure work-alike properties
- to test individual functions to ensure they work across the range of calling mechanisms
- for "silly" inputs to try to see if exceptions are caught

A test runner program

??? this is for the quick testing of sets of problems — documented in TestsDoc.

Output of the project

?? see Working Doc etc.

- formal reports
- informal reports
- problem sets
- code and documentation

References

Bates, D. M., and D. G. Watts. 1988. *Nonlinear Regression Analysis and Its Applications*. Wiley.

Bates, Douglas M., and Donald G. Watts. 1981. "A Relative Offset Orthogonality Convergence Criterion for Nonlinear Least Squares." *Technometrics* 23 (2): 179–83.

Marquardt, Donald W. 1963. "An Algorithm for Least-Squares Estimation of Nonlinear Parameters." *SIAM Journal on Applied Mathematics* 11 (2): 431–41.

Nash, John C, and Duncan Murdoch. 2019. *Nlsr: Functions for Nonlinear Least Squares Solutions*.