# Working Document for the Improvement to nls() GSOC project

Arkajyoti Bhattacharjee, Indian Institute of Technology, Kanpur
John C. Nash, University of Ottawa, Canada
Heather Turner, University of Warwick, UK

03 June, 2021

## Abstract

nls() is the primary nonlinear modeling tool in base R. It has a great many features, but it is about two decades old and has a number of weaknesses, as well as some gaps in documentation. This document is an ongoing record of work under the Google Summer of Code 2021 of the first author. As such it is NOT meant to be a finished academic report, but a form of extended diary of activity, issues and results.

## 2021-5-18

By email, a Google Meet was set up for May 19 at noon Ottawa time. JN downloaded the latest version of `R-devel.tar.gz` and unpacked it into a directory `~/vmshare/R-devel` which is shared with a VirtualBox VM of Kubuntu 20.04 (Focal Fossa), a long-term support version of Linux. Opening a terminal inside this VM, it was possible to build and run this version of R.

```
cd /media/sf_vmshare/R-devel/
./configure
make
sudo make install
[admin password]
R
```

This launched the development version of R correctly.

JN also set up this document.

### Agenda for May 19

- check Meet is working
- introductions
- start linux VM install if an iso is available. May want to check that VirtualBox Guest additions is available and that the shared directory works, as these sometimes require some attention to permissions and ownership etc.
- Consider early goals to for possible nls() changes. See below.
- Set objectives for next two weeks
- Set next online meeting

### Possible early goals

- Get a VM running under VirtualBox and install build tools. See https://support.rstudio.com/hc/en-us/articles/218004217-Building-R-from-source I did NOT need more than ./configure in my build, as I am not building a server version.
- Try the build. (Cheer loudly when it works!)

- Explore and document the R source for nls-related code. In particular, we want
  - to list the files that have such code or calls to it
  - to note, if possible, what each does
  - to note, in particular, where nls() solves the Gauss-Newton equations, as we will want to modify these sections to augment them to allow a Marquardt stabilization.
  - to note, in particular, where nls() computes the Jacobian and/or Hessian for the nonlinear least squares problem. In package `nlsr`, there are tools that allow an expression for the model to be parsed and processed to compute the Jacobian using symbolic or automatic derivatives. This may or may not be feasible for `nls()`. However, we may be able to improve the approximation used.

    [2021-5-27] first draft DerivsNLS.Rmd. Note discovered "central" control for numericDeriv() function. It does NOT appear that the 'dir' control is actually used. This would likely change from forward to backward differences for SOME components of the step to compute first order finite difference approximation to the Jacobian.
- Augment this document to record what has been done and results or problems.
- Add to the bibliography file specified. This is taken from another work, but as Rmarkdown only uses the references it needs, it will serve as a template.
- AB can ask for pointers to references to add to this document or to subsidiary documents we will create as necessary to describe parts of the work if required.

[2021-5-25] Meet session. AB got Linux Mint 20.1 VM going and verified. SOme questions sorted out. JN added some material re: linear least squares solutions to the Variety document.

## Some thoughts on how nls() code might be modified [JN: 2021-5-19]

- R-core will veto any changes that do not leave existing tests and examples as they are. Therefore we will be always thinking of ways to leave the default behaviour as it is now. In the relative offset convergence test that was causing small-residual problems to fail, JN suggested a small change to the test. In very simplified terms, the test is a ratio of `(new sum of squares)/(baseline sum of squares)` When both these quantities are small, we risk zero/zero situation. Therefore, a new control parameter `convTeestAdd` that has a default of 0 was introduced. This can be set in the `nls.control()` function. It is added to the denominator and allows the `nls()` function to complete satisfactorily when we have problems where the fit is nearly exact. Thus we are seeking similar modifications that are invoked by setting new control parameters to non-default settings to get new behaviour.
- In the material below on implementing nonlinear least squares, it should be possible to have a zero valued default for the Marquardt parameter $\lambda$. As long as this zero value is preserved, the legacy behaviour is used. Our decisions need to be careful in situations where we really do need the Marquardt stabilization. That is, should we simply fail (legacy behavior), should we override the legacy behaviour, since the user almost certainly does not want an avoidable failure, or should we seek an additional control that allows for more nuanced outcomes. Clearly thought is needed, and we will need to find some examples to illustrate our arguments.
- Note that nls() is a mix of several codes, using different solution tools. We will need to decide where our modifications apply and document carefully. JN opinion: we may want to postpone consideration of "port" and "plinear" algorithms, but it would be useful to document what they do. The current documentation suggests that bounds on parameters can only be used with "port."
- The derivative code to generate the Jacobian in nls() is (JN believes??) a simple forward-difference code. This may be in C rather than R. It is well-known that central differences, while they cost twice the computing effort, do a better job. Can we find a way to allow that easily?

- If there are bounds on parameters, the "step" taken in estimating differences may violate bounds. This is an issue well-known to workers in optimization, but not on the radar of most users. Clearly a "nasty." Can we do anything to at least warn users in a way that is useful?

**[2021-5-31] Notes on investigations**

1) AB managed to get R built from an R-devel expanded tarball. JN had managed this earlier. There appear to be some transient glitches (HT has noted also). These are likely due to R-core members making trial changes. MASS package in particular noted.

2) Investigations of using subversion or git repo of the code so we can track any changes in files relevant to nls(). JN tried from https://github.com/wch/r-source/wiki which has some instructions. These seemed to work, but we note the need for texinfo and git-svn packages to be installed. Also standard "git pull" cannot be used. "git sv rebase" followed by "git svn fetch" tried and seem to work, but JN does not feel he understands what is going on.

3) Started to look into development of replacement code for parts of nls(). Created ReplaceNLS.R and RestoreNLS.R with the intention that source() of these files would replace the functions. However, first attempt, which simply added a cat() output to numericDeriv() function got

```
Error in numericDeriv(rexpr, theta, rho = weedenv) :
  object 'C_numeric_deriv' not found
```

## [2021-6-1] nlspkg package created

Thanks to Duncan Murdoch who created nlspkg from the R source files. This is a package version of the files in the source tree that will let us work with the code and change it. When we issue

```
library(nlspkg)
```

once we have installed it, the objects in the package will replace (mask) those in base-R.

To build nlspkg, I needed the inconsolata.sty font that is in the `texlive-fonts-extra` package of Linux Mint. In other systems, this font may reside elsewhere.

As of June 1, the package builds and mostly checks. There are two warnings:

- a number of function objects are NOT documented. We may or may not wish to add `man` files (of type `.Rd`). The list of objects is quite long, but they seem not to be needed by the user. Note that nlminb is present, likely for the PORT option of the "algorithm" parameter of `nls()`.
- There is a non-standard license specification. Checking in the Writing R Extensions manual (file r-exts.pdf), suggests dropping one of the two licenses listed (GPL-2 and GPL-3). Dropping the latter removed one warning.

The package `nlspkg` is to provide the EXISTING nls() functionality, while the package `nlsalt` is for our replacement. I plan to make this all-R. If we want to use C (or something else) later, then we will create another "alternate" package.

Note that each of these packages needs its own "project." In Rstudio we choose to create a project using an existing directory (copy of that for `nlspkg`). This sets up the control files for the project and allows the package to be built and checked with the Rstudio tools. The umbrella ImproveNLS project handles the **git** version control.

## [2021-6-1] nlsalt package mods started

numericDeriv first cut at pure R. JN incorporated C code into the R code of the numericDeriv function as comments, then tried to provide R statements to do what it appears the C is doing. Some steps and checks likely left out.

?? Still to test – not working right away on June 2

## [2021-6-2] Converting codes to All-R – numericDeriv

In the `ExDerivs.R` script, JN tested a forward difference code in R against a call to numericDeriv(). On a single example, it tests OK, so next step is to modify nls.R in package `nlsalt`.

Note that `ExDerivs.R` is in flux. "Old" tests will be commented out. Look for the double question mark (??) for things that may need to be checked.

Some concerns:

- the code requires us to get() and assign() values in the environment `rho` that is used to evaluate the nonlinear residual functions. There may or may not be ways to make this more efficient.
- the C code uses explicit loops. A very preliminarly use of R's vectorization in the difference between two residual vectors (i.e., without subscripts or looping) worked fine, but there are almost certainly further optimizations. By contrast, we MAY want to leave either explicit loops or commented code that contains such loops as a way to provide explanation of what is happening.

## From `nlsr` vignette `nlsr-devdoc.Rmd` [2021-5-23 onwards]

### 5. Implementation of nonlinear least squares methods

This section is a review of approaches to solving the nonlinear least squares problem that underlies nonlinear regression modelling. In particular, we look at using a QR decomposition for the Levenberg-Marquardt stabilization of the solution of the Gauss-Newton equations.

#### Gauss-Newton variants

Nonlinear least squares methods are mostly founded on some or other variant of the Gauss-Newton algorithm. The function we wish to minimize is the sum of squares of the (nonlinear) residuals $r(x)$ where there are $m$ observations (elements of $r$) and $n$ parameters $x$. Hence the function is

$$f(x) = \sum_i ({r_i}^2)$$

Newton's method starts with an original set of parameters $x[0]$. At a given iteration, which could be the first, we want to solve

$$x[k+1] = x[k] - H^{-1} g$$

where $H$ is the Hessian and $g$ is the gradient at $x[k]$. We can rewrite this as a solution, at each iteration, of

$$H\delta = -g$$

with

$$x[k+1] = x[k] + \delta$$

For the particular sum of squares, the gradient is

$$g(x) = 2 * r[k]$$

and

$$H(x) = 2(J'J + \sum_i (r_i * Z_i))$$

where $J$ is the Jacobian (first derivatives of $r$ w.r.t. $x$) and $Z_i$ is the tensor of second derivatives of $r_i$ w.r.t. $x$). Note that $J'$ is the transpose of $J$.

The primary simplification of the Gauss-Newton method is to assume that the second term above is negligible. As there is a common factor of 2 on each side of the Newton iteration after the simplification of the Hessian, the Gauss-Newton iteration equation is

$$(J'J)\delta = -J'r$$

This iteration frequently fails. The approximation of the Hessian by the Jacobian inner-product is one reason, but there is also the possibility that the sum of squares function is not "quadratic" enough that the unit step reduces it. Hartley (1961) introduced a line search along delta, while Marquardt (1963) suggested replacing $J'J$ with $(J'J + \lambda * D)$ where $D$ is a diagonal matrix intended to partially approximate the omitted portion of the Hessian.

Marquardt suggested $D = I$ (a unit matrix) or $D =$ (diagonal part of $J'J$). The former approach, when $\lambda$ is large enough that the iteration is essentially

$$\delta = -g/\lambda$$

gives a version of the steepest descents algorithm. Using the diagonal of $J'J$, we have a scaled version of this (see https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm; Levenberg (1944) predated Marquardt, but the latter seems to have done the practical work that brought the approach to general attention.)

Nash (1977) found that on low precision machines, it was common for diagonal elements of $J'J$ to underflow. A very small modification to solve

$$(J'J + \lambda(D + \phi I))\delta = -g$$

where $\phi$ is a small number avoids most of these conditions. $\phi = 1$ seems to work quite well. We note that this modification would likely not have been recognized had I not been working on machines with mediochre floating-point arithmetic – a little more than 6 decimal digits of precision and no extended precision.

**Choices**

Both `nlsr::nlxb()` and `minpack.lm::nlsLM` use a Levenberg-Marquardt stabilization of the iteration described above, with `nlsr` using the modification involving the $\phi$ control parameter. The complexities of the code in `minpack.lm` are such that I have relied largely on the documentation to judge how the iteration is accomplished. `nls()` uses a straightforward Gauss-Newton iteration, but rather than form the sum of squares and cross-products, uses a QR decomposition of the matrix $J$ that has been found by a forward difference approximation.

(Nash (1979)), solving

$$(J^T J + \lambda D)\delta = -J^T r$$

where $D$ is some diagonal matrix and lambda is a number of modest size initially. Clearly for $\lambda = 0$ we have a Gauss-Newton method. Typically, the sum of squares of the residuals calculated at the "new" set of parameters is used as a criterion for keeping those parameter values. If so, the size of $\lambda$ is reduced. If not, we

increase the size of $\lambda$ and compute a new $\delta$. Note that a new $J$, the expensive step in each iteration, is NOT required.

As for Gauss-Newton methods, the details of how to start, adjust and terminate the iteration lead to many variants, increased by different possibilities for specifying $D$. See Nash (1979).

**Using matrix decompositions [2021-5-23 onwards]**

In Nash (1979), the iteration equation was solved as stated. However, this involves forming the sum of squares and cross products of $J$, a process that loses some numerical precision. A better way to solve the linear equations is to apply the QR decomposition to the matrix $J$ itself. However, we still need to incorporate the $\lambda I$ or $\lambda D$ adjustments. This is done by adding rows to $J$ that are the square roots of the "pieces." We add 1 row for each diagonal element of $I$ and each diagonal element of $D$.

In each iteration, we reduce the $\lambda$ parameter before solution. If the resulting sum of squares is not reduced, $\lambda$ is increased, otherwise we move to the next iteration. Various authors (including the present one) have suggested different strategies for this. My current opinion is that a "quick" increase, say a factor of 10, and a "slow" decrease, say a factor of 0.2, work quite well. However, it is important to check that $\lambda$ has not got too small or underflowed before applying the increase factor. On the other hand, it is useful to be able to set lambda = 0 in the code so that a pure Gauss-Newton method can be evaluated with the program(s). The current code `nlfb()` uses the line

```
if (lamda<1000*.Machine$double.eps) lamda<-1000*.Machine$double.eps
```

to ensure we get an increase. To force a Gauss-Newton algorithm, the controls `laminc` and `lamdec` are set to 0.

**The Levenberg-Marquardt adjustment to the Gauss-Newton approach is the second major improvement of `nlsr` (and also its predecessor `nlmrt` and the package `minpack-lm`) over `nls()`.**

`nls()` attempts to minimize a sum of squared residuals by a Gauss-Newton method. If we compute a Jacobian matrix $J$ and a vector of residuals $r$ from a vector of parameters $x$, then we can define a linearized problem

$$J^T J\delta = -J^T r$$

This leads to an iteration where, from a set of starting parameters $x_0$, we compute

$$x_{i+1} = x_i + \delta$$

This is commonly modified to use a step factor `step`

$$x_{i+1} = x_i + step * \delta$$

It is in the mechanisms to choose the size of *step* and to decide when to terminate the iteration that Gauss-Newton methods differ. Indeed, though I have tried several times, I find the very convoluted code behind `nls()` very difficult to decipher. Unfortunately, its authors have now (as far as I am aware) all ceased to maintain the code.

We could implement the methods using the equations above. However, the accumulation of inner products in $J^T J$ occasions some numerical error, and it is generally both safer and more efficient to use matrix decompositions. In particular, if we form the QR decomposition of $J$

$$QR = J$$

where Q is an orthonormal matrix and R is Right or Upper triangular, we can easily solve

$$R\delta = -Q^T r$$

for which the solution is also the solution of the Gauss-Newton equations. But how do we get the Marquardt stabilization?

If we augment $J$ with a square matrix $\sqrt{\lambda D}$ whose diagonal elements are the square roots of $\lambda$ times the diagonal elements of $D$, and augment the vector $r$ with $n$ zeros where $n$ is the column dimension of $J$ and $D$, we achieve our goal.

Typically we can use $D = I_n$ (the identity of order $n$), but Marquardt (1963) showed that using the diagonal elements of $J^T J$ for $D$ results in a useful implicit scaling of the parameters. Nash (1977) pointed out that on computers with limited arithmetic (which now are rare since the IEEE 754 standard appeared in 1985), underflow might cause a problem of very small elements in $D$ and proposed adding $\phi I_n$ to the diagonals of $J^T J$ before multiplying by $\lambda$ in the Marquardt stabilization. This avoids some awkward cases with very little extra overhead. It is accomplished with the QR approach by appending $\sqrt{\phi\lambda}$ times a unit matrix $I_n$ to the matrix already augmented matrix. We must also append a further $n$ zeros to the augmented $r$.

[2021-5-27] JN added some notes on LU and Cholesky to "Variety" document.

**======== End of extract on approaches to solving Gauss Newton equations =========**

## Early explorations and setup of project [2021-5-24]

This is essentially a diary entry. (?? AB you may want to add some notes too.)

- AB working to secure his computer, set up VirtualBox and prepare a linux guest VM
- AB has been looking at documents JN posted. Gitlab repo seems to be working OK. On comment from HT, JN discovered (very overdue!) that Rstudio support for git is very useful.
- JN set up nls-flowchart.txt for use by team. This is looking at R-devel.tar.gz downloaded May 18, 2021. It is likely we can use any recent version, though at some point we may need to be more precise. The purpose of this is to describe what is going on in the nls() operations.
- JN added document **VarietyInNonlinearLeastSquaresCodes.Rmd**. This is tangential to the project, but is an attempt to provide a panorama of the different ways in which nonlinear least squares software has been set up over the years. JN will respond to questions from other team members to try to make this explanatory document helpful in understanding the existing nls() code and proposed changes.
- Section added above "From `nlsr` vignette `nlsr-devdoc.Rmd`"
- In the "VarietyInNonlinearLeastSquaresCodes.Rmd" document, JN added a section on different approaches to solving LINEAR least squares problems, since the inner iteration of almost all nonlinear least squares methods involves such a sub-problem. AB has been looking at this. It is currently a messy DRAFT section at 2021-5-25.
- Meet scheduled for May 25;

## Attempts to understand numericDeriv() [2021-5-24 to 27]

JN has found it quite difficult to get a good understanding of the `numericDeriv()` function that appears in `nls.R` and calls material in `nls.c`. That this has proved difficult to use suggests the documentation is less than wonderful. However, the main reason is to set up tests and timings and compare to other approaches to computing the Jacobian in nonlinear least squares. ??Gut feeling: numericDeriv is really only set up to compute a Jacobian. ?? Note: there is a "central" option. Can this be used with nls()? Why or why not?

A file ExDerivs.R has been added to the repo.

[2021-5-26] JN sorted out one approach that allows numricDeriv, nlsr::model2rjfun and numDeriv::jacobian to be compared. [2021-5-27] DerivsNLS.Rmd first draft done. Problems with system() and system2() and calling bash from rmarkdown document led to email sent to Yihui Xie, as the output of a Linux command (inxi -F) to get machine properties was returned with extra characters.

## References


Hartley, H. O. 1961. "The Modified Gauss-Newton Method for Fitting of Nonlinear Regression Functions by Least Squares." *Technometrics* 3: 269–80.

Levenberg, Kenneth. 1944. "A Method for the Solution of Certain Non-Linear Problems in Least Squares." *Quarterly of Applied Mathematics* 2: 164--168.

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. 1977. "Minimizing a Nonlinear Sum of Squares Function on a Small Computer." *Journal of the Institute for Mathematics and Its Applications* 19: 231–37.

———. 1979. *Compact Numerical Methods for Computers : Linear Algebra and Function Minimisation*. Book. Hilger: Bristol.