Referee report: A Comparison of R Tools for Nonlinear Least Squares Modeling

Id: 2023-15

2023-06-28

General

This article provides a vignette-style overview of the `nlsr`-package and highlights its design choices and behavior in comparison to base R's `nls()` function as well as the nonlinear least squares solvers provided by the R-packages `minpack.lm` and `gslnls`. The article's main contribution is to make R users (more) aware of the historic shortcomings and pitfalls in using R's `nls()` function and how the `nlsr`-package is able to serve as a more reliable alternative. Overall, the article is well-written and structured into coherent subthemes and sections. Given that `nls()` is the de facto standard for solving nonlinear least squares problems in R, I believe this article provides valuable insights and is well-suited for publication in the R Journal after some revisions and minor clarifications.

My main advice is to revise the title and introduction of the article. As it currently stands, it is suggested that the article provides an (objective) comparison of nonlinear least squares solvers available in R. The body of the article reads more like a vignette for the `nlsr`-package, explaining its design choices and comparing its behavior to `nls()` and other NLS solvers based on a few selected examples. As such, a more appropriate title could be, for instance, *The current state of R tools for nonlinear least squares modeling* or similar.

Detailed comments

1. The Hobbs weed example as an illustration of the performance of different NLS solvers is lacking a thorough comparison. I understand that the aim of the authors is to provide a data example that demonstrates the reliability of `nlsr` in favor of `nls()`, but `nls()` converges to the same solution as `nlsr` for all model setups when using the Port algorithm. This goes unnoticed in the text. Also, the model setup (Logistic3T) for which `gsl_nls()` fails can easily be made to converge correctly by changing either the algorithm or the scaling method, see the **Illustrating code** section. My advice is to provide a more comprehensive comparison of the available NLS solvers, or to pick a different data example that better highlights the benefits of `nlsr`. For instance, trying to solve the BoxBOD regression problem (https://www.itl.nist.gov/div898/strd/nls/data/boxbod.shtml), `nls()` fails with starting values $\theta_1 = 1, \theta_2 = 1$ using both the Gauss-Newton and Port algorithm, whereas `nlsr::nlxb()` converges correctly.

2. The section *Returned results of nls() and other tools* discusses the complexity of S3-objects of class `nls`, illustrated by the retrieval of the input data from an `nls`-object. It seems ill-advised to run `eval(parse(text=result$data))`, since it evaluates the name of the data object in the user's environment, which may not exist. Instead, one should use `model = TRUE` to store the input data in the returned `nls`-object. This behavior is consistent with `lm()`, so perhaps less confusing than currently presented in the manuscript. The authors mention that `nlxb()` returns a much simpler structure of 11 items in one level, but I did not manage to find the input data object in the `nlsr`-object returned by `nlxb()`.

3. The section *Functional specification of problems* mentions the use of `nlsr::nlfb()` to solve nonlinear least squares problems using a function to define the residuals. Since `nlsr::nlfb()` only has a notion of the residuals, how is it able to disentangle model and response, for the purpose of e.g. generating

model predictions? The authors mention that the function `wrapnlsr()` can be used to return an object of class `nls`, but this does not seem to work for problems defined through a residual function as in `nlsr::nlfb()`? If indeed the case, how does one evaluate e.g. model predictions, parameter confidence intervals, etc. when using `nlsr::nlfb()`? This remains unclear to me.

4. The section *Philosophical considerations* hints at the challenge of performing inference when parameter bound-constraints are involved in the optimization. In case of physical or model constraints on the parameters, a common approach is to reparametrize the model to be unconstrained in the parameters. Perhaps the authors can highlight an example scenario where imposing parameter constraints is preferable to model reparametrization (the included data examples with parameter constraints appear somewhat artificial), or elaborate further on the possible downsides of reparametrizing the model, if any.

5. The section *Programming language* makes the argument that keeping to a single programming language can allow for easier maintenance and upgrades, and that the performance penalty for using code entirely in R has been much reduced in recent years. I agree that there is not much of a performance penalty when solving small problems, such as the data examples included in the article. However, when solving problems using a large amount of data or a large number of parameters, I have experienced that the run-time (as well as memory usage) can become an actual bottleneck. As an example, consider the *Penalty function I* problem (Moré, Garbow, and Hillstrom (1981)) included in the **Illustrating code** section.

Minor comments

- Avoid emphasizing words by writing in all capitals. Examples: NOT (pg. 14, 15), WEIGHTED (pg. 13) and more.
- The link https://towardsdatascience.com/unit-testing-in-r-68ab9cc8d211 (as well as the links to github) may be subject to change or become unavailable in the future.

Illustrating code

```
## Hobbs weed example
weeddf <- data.frame(
  tt = 1:12,
  weed = c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192,
           31.443, 38.558, 50.156, 62.948, 75.995, 91.972)
)

### nls-port all setups
nls(
  formula = weed ~ b1 / (1 + b2 * exp(-b3 * tt)),
  data = weeddf,
  start = c(b1 = 1, b2 = 1, b3 = 1),
  algorithm = "port"
)
#> Nonlinear regression model
#>   model: weed ~ b1/(1 + b2 * exp(-b3 * tt))
#>    data: weeddf
#>       b1       b2       b3
#> 196.1863  49.0916   0.3136
#>  residual sum-of-squares: 2.587
#>
#> Algorithm "port", convergence message: relative convergence (4)

nls(
  formula = weed ~ 100 * c1 / (1 + 10 * c2 * exp(-0.1 * c3 * tt)),
  data = weeddf,
  start = c(c1 = 1, c2 = 1, c3 = 1),
  algorithm = "port"
)
#> Nonlinear regression model
#>   model: weed ~ 100 * c1/(1 + 10 * c2 * exp(-0.1 * c3 * tt))
#>    data: weeddf
#>    c1    c2    c3
#> 1.962 4.909 3.136
#>  residual sum-of-squares: 2.587
#>
#> Algorithm "port", convergence message: relative convergence (4)

nls(
  formula = weed ~ Asym /(1 + exp((xmid - tt) / scal)),
  data = weeddf,
  start = c(Asym = 1, xmid = 1, scal = 1),
  algorithm = "port"
)
#> Nonlinear regression model
#>   model: weed ~ Asym/(1 + exp((xmid - tt)/scal))
#>    data: weeddf
#>    Asym    xmid    scal
#> 196.186  12.417   3.189
#>  residual sum-of-squares: 2.587
#>
#> Algorithm "port", convergence message: relative convergence (4)

### gsl_nls Logistic3T setup
gslnls::gsl_nls(
  fn = weed ~ Asym / (1 + exp((xmid - tt) / scal)),
  data = weeddf,
  start = c(Asym = 1, xmid = 1, scal = 1),
  algorithm = "lmaccel"
)
#> Nonlinear regression model
#>   model: weed ~ Asym/(1 + exp((xmid - tt)/scal))
#>    data: weeddf
#>    Asym    xmid    scal
#> 196.186  12.417   3.189
#>  residual sum-of-squares: 2.587
#>
#> Algorithm: multifit/levenberg-marquardt+accel, (scaling: more, solver: qr)
#>
#> Number of iterations to convergence: 22
#> Achieved convergence tolerance: 1.18e-11

gslnls::gsl_nls(
  fn = weed ~ Asym / (1 + exp((xmid - tt) / scal)),
  data = weeddf,
  start = c(Asym = 1, xmid = 1, scal = 1),
  algorithm = "dogleg"
)
```

```
#> Nonlinear regression model
#>   model: weed ~ Asym/(1 + exp((xmid - tt)/scal))
#>    data: weeddf
#>    Asym    xmid    scal
#> 196.186  12.417   3.189
#>  residual sum-of-squares: 2.587
#>
#> Algorithm: multifit/dogleg, (scaling: more, solver: qr)
#>
#> Number of iterations to convergence: 16
#> Achieved convergence tolerance: 1.599e-13

gslnls::gsl_nls(
  fn = weed ~ Asym / (1 + exp((xmid - tt) / scal)),
  data = weeddf,
  start = c(Asym = 1, xmid = 1, scal = 1),
  algorithm = "ddogleg"
)
#> Nonlinear regression model
#>   model: weed ~ Asym/(1 + exp((xmid - tt)/scal))
#>    data: weeddf
#>    Asym    xmid    scal
#> 196.186  12.417   3.189
#>  residual sum-of-squares: 2.587
#>
#> Algorithm: multifit/double-dogleg, (scaling: more, solver: qr)
#>
#> Number of iterations to convergence: 16
#> Achieved convergence tolerance: 1.14e-11

gslnls::gsl_nls(
  fn = weed ~ Asym / (1 + exp((xmid - tt) / scal)),
  data = weeddf,
  start = c(Asym = 1, xmid = 1, scal = 1),
  algorithm = "subspace2D"
)
#> Nonlinear regression model
#>   model: weed ~ Asym/(1 + exp((xmid - tt)/scal))
#>    data: weeddf
#>    Asym    xmid    scal
#> 196.186  12.417   3.189
#>  residual sum-of-squares: 2.587
#>
#> Algorithm: multifit/2D-subspace, (scaling: more, solver: qr)
#>
#> Number of iterations to convergence: 19
#> Achieved convergence tolerance: 9.783e-13

gslnls::gsl_nls(
  fn = weed ~ Asym / (1 + exp((xmid - tt) / scal)),
  data = weeddf,
  start = c(Asym = 1, xmid = 1, scal = 1),
  control = list(scale = "levenberg")
)
#> Nonlinear regression model
#>   model: weed ~ Asym/(1 + exp((xmid - tt)/scal))
#>    data: weeddf
#>    Asym    xmid    scal
#> 196.186  12.417   3.189
#>  residual sum-of-squares: 2.587
#>
#> Algorithm: multifit/levenberg-marquardt, (scaling: levenberg, solver: qr)
#>
#> Number of iterations to convergence: 24
#> Achieved convergence tolerance: 7.065e-13
```

```
## NOT RUN

## Penalty function I [More, Garbow, Hillstrom 1981]
alpha <- 1e-5
n <- 1000
### residuals
fres  <-  function(x) {
  c(sqrt(alpha) * (x - 1), sum(x^2) - 0.25)
}
### jacobian
fjac <- function(x) {
  jj <- rbind(diag(sqrt(alpha), nrow = length(x)), 2 * t(x))
  attr(jj, "gradient") <- jj
  jj
}

bench::mark(
  "large problem" = nlsr::nlfb(start = 1:n, resfn = fres, jacfn = fjac),
  iterations = 1
)
# # A tibble: 1 × 13
#   expression         min   median `itr/sec` mem_alloc `gc/sec`
#   <bch:expr>    <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
# 1 large problem    3.24m    3.24m   0.00515    9.23GB    0.226
```

References

Moré, Jorge J., Burton S. Garbow, and Kenneth E. Hillstrom. 1981. "Testing Unconstrained Optimization Software." *ACM Transactions on Mathematical Software* 7: 17–41.