nlsCompare: An R package for comparing nonlinear least squares packages in R

by Arkajyoti Bhattacharjee[1], John C. Nash[2], and Heather Turner[3]

Abstract Nonlinear least squares (nls) is extensively used in nonlinear modeling. This paper introduces the package **nlsCompare** that attempts to compare the best values of parameters and sum-of-squares in nls problems with existing tools for nls like `nls`(**base**), `nlxb` (**nlsr**), `nlsLM`(**minpack.lm**) and a new experimental package **nlsj** developed by the authors as a product of Google Summer of Code (GSoC), 2021.

Introduction

While working on the GSoC project **Improvements to nls()** under The R Project for Statistical Computing, the authors realized that to be able to observe improvements made via experimental packages such as `nlsj`, developers needed to quickly compare results with the best known reference values for each of a battery of problems.

nlsCompare:

- has a spreadsheet with pointers to test problem files along with reference solutions. These problem/cases are based on packages like **NRAIA** with data, models, constraints and starting values.
- has a spreadsheet containing the methods (solvers, algorithms, control settings) to be tested
- generates easy-to-read spreadsheets that compare results (sum of squared residuals and model parameters) with the best reference values yet observed. Since there can be multiple methods, these spreadsheets allow for fairly rapid comparisons. Moreover, the columns can be sorted to allow specific features to be compared more easily.
- gives appropriate messages to assist the understanding of errors or where to do improvements

nlscompare draws inspiration from R's testing library **testthat** and uses ideas from packages like **benchmarkme** to identify the computing environment. Currently, we have restricted our attention to the R computing environment.

Need for nlsCompare

Regression testing is useful not only to validate correctness of a program but often also for tracking the quality of its output. While working on improving `nls`, author[2] noted that to compare the improvements made over `nls` via **nlsj**, unit testing via **testthat** is not enough, largely because some desirable features such as bounds constraints within the default Gauss-Newton method is not provided. That is, we are trying to provide for a result currently unavailable in `nls()`. While we did for some time pursue the use of other nls functions, we eventually moved to using reference values from our problems spreadsheet. **nlsCompare** allows us to note which methods are supported by a particular function that are not in another and to check the consistency of supposedly equivalent results from different functions.

nlscompare attempts to achieve the above, outputting a simple informative spreadsheet that indicates whether an existing or new package's function provides consistent results with the current best obseved values. Further, it outputs an error log that indicates whether a solver (viz. `nls` or `nlsLM`, etc.) fail in a particular problem with particular methods (algorithms, starting points, etc.)

Workflow

The **nlscompare** needs to be run in the following hierarchical steps -

1. `setup_dir()`: choose and set the working directory save final csv outputs.
2. `machineId()`: create the machine ID to be used in the output csv files.
3. `create_db()`: create dataframe to store database of nls problems
4. `create_elog()`: create dataframe to store error-log in solving the nls problems
5. `csv_exists`: check if the csv files **nlsDatabase.csv** and **nlsErrorLog.csv** exist already in the directory chosen in 1. Create them if not present

6. `run()`: this uses the problem names - **Name** , the reference values - **nPars**, **Parameters** and **ssquares** from **problems.csv** , the methods from **methods.csv**, does the comparisons and error logging and edits the dataframes accordingly.
7. `write_csvs`: write the dataframes returned by `run` into **nlsDatabase.csv** and **nlsErrorLog.csv**
8. `rm_nls` : delete the global variables created due to running of the above functions

A detailed explanation on how to install and use the package is available in the package vignette - nlsCompare: How to Use it?

Package content details

methods.csv

This csv file contains three columns:

1. **solver:** This is the name of the function to be used to solve the nls problem. Currently, it supports `nlsj::nlsj`, `nls`, `nlsr::nlxb` and `minpack.lm::nlsLM`.
2. **algorithm:** This is the name of the algorithm used. Currently, supports `default`, `plinear`, `port`, `LM` and `marquardt`, but only when these are appropriate to a given solver.
3. **control:** This contains the package function and its associated control settings to control the iterations or other features of the solver and/or problem.

The **methods.csv** file is essentially a list of detailed methods for use inside the `run` function.

Users can add new solvers, algorithms and controls as in the existing csv file.

problems.csv

This csv file contains four sets of columns:

- **Name:** This contains the names of files present in the **./nlsCompare/inst/scripts** of the package and is used by the `run()` function to source the **Name**-ed files inside the function.
- **ssquares:** This contains the best (least) possible sum of squares achievable in the nls problem as present in the **Name**-ed file.
- **nPars:** The number of parameters in the nls problem present in a **Name**-ed file.
- **Pars:** This and the adjacent (**nPars** - 1) columns contain the best parameter values of the nls problem present in the **Name**-ed file.

The "best" values are based on the observations made by the authors on using different nls packages. These "best" values serve as a reference that is used inside the `run` function to compare with a **solver**'s output using the corresponding **algorithm** and **control**. In case a user finds a "better" (see below) value, the values inside this csv should be changed manually by the user.

./nlsCompare/inst/scripts

This directory contains the R scripts for numerous problems in nls that are sourced inside the `run` function. Each file has a naming structure: "problemName_x" where x is a integer number. "1" means that it is a parent test file of the "problemName" family of test files. The parent file mainly contains -

- `NLSdata`: A dataframe that contains the data
- `NLSformula`: nonlinear model formula
- `NLSstart`: Starting values for the nls problem
- `NLSlower`: An lower bound for the nls problem
- `NLSupper`: An upper bound for the nls problem
- `NLSweights`: weights for the nls problem
- `NLSsubset`: for subsetting the dataframe
- `NLSmethods`: a dataframe that stores the contents of **methods.csv**
- `NLSproblems`: a dataframe that stores the contents of **problems.csv**
- `refsol`: the package name that gives the best results to an nls problem listed in **problems.csv**
- `NLSssquares`: the sum of squares produced by **refsol**. This is derived from **problems.csv**'s **ssquares** column
- `NLSpars`: the parameters produced by **refsol**. This is derived from **problems.csv**'s **Pars** and its adjacent columns
- `NLStag`: a tag to classify the nls problem

- `NLSref`: nls object created using **refsol** to a problem

If x in "_x" is greater than 1, it means that it is a children test file belonging to the same "problemName family of test files. It contains a subset of the contents as mentioned above. For example, *NLSprob_2.R* may contain just a new `NLSstart` or may contain `NLSupper` and `NLSlower` and so on.

To add "children" scripts, the naming convention is compulsory. "NLSproblemName.x" must have the x value just succeeding the last x value for the same *problemName*.

Coverage

The idea of hierarchy in test files allows the diverse cases of:

- multiple starting values that may be acceptable or **infeasible** (i.e., outside constraints)
- bounded problems having a good or **inadmissible** (hence silly) bounds where one or more lower bounds is greater than the corresponding upper bound. For such problems all other inputs are irrelevant. The problem is not well-posed.
- multiple sets of weights for the same problem. Some of these may render a problem more or less difficult to solve.
- ability to work with subsets of the problem's data. At the time of writing we have not pursued subsets beyond a few trial cases.

The **methods.csv** allows the possibility for adding more solvers to the existing set:

- functions as **solver**s viz. `nlsj::nlsj`, `nlsr::nlxb` and `minpack.lm::nlsLM`
- existing and new **algorithm**s viz. "marquardt", "plinear", "port" and "Gauss-Newton" and its variants

The `run()` function currently only compares essential estimation quantities, that is the sum of squares and the parameters that generate this value.

Package Output Files

nlsDatabase.csv

This csv file stores a final output of the `run()` function. It has 11 columns -

- `DateTime`: the data and time (space-separated) the program runs a particular **FileName**
- `MachID`: a one-line machine summary useful for characterizing different machines
- `FileName`: the name of the file run inside the `run()` function. It is used from **Problems.csv**
- `Solver`: the nls function used from **methods.csv** to solve a nonlinear problem
- `Algorithm`: the algorithm used from **methods.csv** to solve a nonlinear problem
- `Control`: the control settings used from **methods.csv** to solve a nonlinear problem
- `Parameters`: result on comparing **Refsol** parameter values with that of **Solver**'s. It is `TRUE` if both are equal, a `NUMERIC` indicating a "Mean relative difference" and `NA` if the **solver** fails to run.
- `SSquares`: result on comparing **Refsol** sum of squares value with that of **Solver**'s. It is `TRUE` if both are equal, a `NUMERIC` indicating a "Mean relative difference" and `NA` if the **solver** fails to run.
- `Better`: comment based on **Parameters** and **SSquares**. It is `EQUAL` if both are `TRUE`, `BETTER` if *SSquares(Solver)* is less than *SSquares(refsol)* and `Worse` otherwise.
- `RefSol`: the package that gives the best parameter and sum of squares value for a problem. `NLStag` is used here.
- `Tags`: the tag used to classifiy the problem. `NLStag` value is used here.

nlsErrorLog.csv

This csv file stores a final output of the `run()` function and is essentially an error-log. Further, it is a subset of the **nlsDatabase.csv** which stores information of only those files which on which a particular *solver* does not run. It has 7 columns -

- `DateTime`: the data and time (space-separated) the program runs a particular **FileName**
- `MachID`: a one-line machine summary useful for characterizing different machines
- `FileName`: the name of the file run inside the `run()` function that observed an error.

- `Solver`: the nls function used from **methods.csv** that caused the error in solving **FileName** problem
- `Algorithm`: the algorithm used from **methods.csv** to solve the nonlinear problem
- `Control`: the control settings used from **methods.csv** to solve the nonlinear problem
- `Message`: the error message associated with the problem that comes from the `Solver`

Some observations

On using **nlsCompare**, we have made some observations that point towards the usefulness of a package.

- **nlsDatabase.csv** gave us an example (Tetra_1.R) where a solver indeed turned out to be better than our perceived reference solver. In this case, the problem has some quite extreme properties that were not obvious, and the output from **nlsCompare** is leading hopefully to some new functionality.
- Sometimes the reference values, reported in decimal notation are not sufficiently precise and a "better" sum of squares is tagged. As we discover such "better" instances, we will deal with them on a case by case basis by replacing the references values appropriately.
- **nlsDatabase.csv** shows which packages fail in a problem case and which succeed

Future work

A few things we would like to incorporate in **nlsCompare** -

- add other estimation quantities like **Convergence**, **Jacobian**, and some characteristics of the solutions related to uncertainty in the parameters.
- add `run_specific` that runs a particular file or a list of files rather than all files as is listed in **problems.csv**
- automatically update the reference values in the R scripts in ./nlsCompare/inst/scripts on encountering "better" values and use **pkgVersion** and**LastUpdated** columns in **problems.csv** to keep track of such changes.
- add more robust bound checking mechanisms like `optimx::bmchk`
- include more problems in the database; for example, from **NISTnls** or other collections in the literature.

Arkajyoti Bhattacharjee[1]
Mathematics and Statistics Department
Indian Institute of Technology, Kanpur
India
arkastat98@gmail.com

John C. Nash[2]
University of Ottawa

https://journal.r-project.org
ORCiD: 0000-0002-9079-593X
nashjc@uottawa.ca

Heather Turner[3]
Department of Statistics
University of Warwick
Coventry
United Kingdom
ORCiD: 0000-0002-1256-3375
ht@heatherturner.net