Journal of Statistical Software

September 2014, Volume 60, Issue 2. *http://www.jstatsoft.org/*

On Best Practice Optimization Methods in R

John C. Nash
University of Ottawa

Abstract

R (R Core Team 2014) provides a powerful and flexible system for statistical computations. It has a default-install set of functionality that can be expanded by the use of several thousand add-in packages as well as user-written scripts. While R is itself a programming language, it has proven relatively easy to incorporate programs in other languages, particularly Fortran and C. Success, however, can lead to its own costs:

- Users face a confusion of choice when trying to select packages in approaching a problem.
- A need to maintain workable examples using early methods may mean some tools offered as a default may be dated.
- In an open-source project like R, how to decide what tools offer "best practice" choices, and how to implement such a policy, present a serious challenge.

We discuss these issues with reference to the tools in R for nonlinear parameter estimation (NLPE) and optimization, though for the present article 'optimization' will be limited to function minimization of essentially smooth functions with at most bounds constraints on the parameters. We will abbreviate this class of problems as NLPE. We believe that the concepts proposed are transferable to other classes of problems seen by R users.

Keywords: R, optimization methods, best practice.

1. Background

In the 1970s, I developed several optimization tools for small computers – machines with 8K bytes for programs (usually in BASIC) and data. These were included in a rather idiosynchratic book (Nash 1979) that is in print more than 30 years later. Three of the algorithms from that book are the Nelder-Mead, BFGS (Broyden-Fletcher-Goldfarb-Shanno) and CG (conjugate gradients) methods of the R `optim()` function, which Brian Ripley converted from their Pascal versions with my permission. They are good programs, but better tools from similar classes

of method are now available, as discussed below.

Furthermore, at this time, a method somewhere intermediate between the conjugate-gradient (CG) and variable metric (BFGS) has not been identified as best-practice. The L-BFGS-B tool in `optim()` is apparently (see Wikipedia 2014) based on version 2.3 of this code by Zhu, Byrd, Lu, and Nocedal (1997). However, the authors of L-BFGS-B have recently (February 2011) released version 3.0 of the code, and there is a variant of this method in package **nloptr** (Ypma 2014) as well as a truncated Newton method, a class of method that offers similar storage advantages to L-BFGS approaches.

Outside `optim()`, but still in the default-install collection, `nlm()` and `nlminb()` offer optimization capabilities. That there are these tools separate from `optim()` openly inspires confusion in R novices. The wrapper **optimx** (Nash and Varadhan 2011) is an attempt to bring such tools under one umbrella. It has been evolving since it was introduced, with some ideas tried and discarded while others have been streamlined. It is anticipated that this evolution will continue as it is written entirely in R and the structure is such that it is relatively easy to add new optimizers and other features.

optimx has been found especially useful as a way to compare the performance of more than a dozen optimizers on particular problems. It also standardizes the call to these optimizers, so allows simpler changing from one method to another. However, it could and should be modified to allow much more guidance about methods to users unfamiliar with NLPE methods.

Ordinary users may also be unaware that some tools are intended to be computationally robust – the nonlinear least squares solvers in package **nlmrt** (Nash 2014b) for example – rather than trying to be very efficient on most problems. A wrong answer computed quickly is not acceptable, but neither is it reasonable to do unnecessary work on a problem that is known to be straightforward and is carried out frequently. The user is generally not party to these design decisions, and may become interested only when workflow is affected by bad solutions or what appears to be excessive run-time.

2. Inertia

A large obstacle to changing methods is that many of R's capabilities are built up from different building blocks. Maximum likelihood tools call optimizations, as do some of the time series, structural equation modeling and neural network functions. An example is **bbmle** (Bolker and R Core Team 2014), which suggests **optimx**. There is a natural reluctance to risk breaking these tools because of these dependencies.

This inertia is also driven by costs. The R Core Team has done a great job in building a highly successful software system. Any request such as that made here potentially implies more work. It is, therefore, important that we offer ideas how the overall work can actually be reduced or at least shared by others.

3. Principles for progress

To balance the need for progress with the requirement to maintain existing capabilities, consider the following principles.

- Users should be able to get reasonable results with default settings for any methods. For example, novice users who invoke a command such as `optim()` with the minimum required elements should get the best default method we can offer them for their problem.
- Developers should be able to experiment, and to test and measure performance, under the most flexible of conditions.
- Existing functionality must remain working, including the use of legacy methods if necessary, though the use of such functionality may be outside the defaults.
- Methods for a given class of problem, e.g., NLPE, should evolve with the accepted wisdom in the sub-domain of expertise.

Recognizing that R is a volunteer-driven open project, we should add the requirement that any changes should impose the minimum possible burden on members of the R Core Team. However, failure to evolve the base methods in R would, it can be argued, imperil the future of the project as users migrate to tools that offer more capable resources.

While the examples in this article are from the NLPE problem domain, it is also likely that there are other statistical and graphical domains of broad utility that are equally affected.

4. The scope of the issue

Fully exercising the principles above presents a challenge with respect to the R Project for Statistical Computing. Even restricting our attention to the tools for NLPE within the Comprehensive R Archive Network (CRAN) and **Bioconductor** repositories requires that we review many packages. There are still more tools on the developmental collections R-Forge and Rforge, and almost certainly more in private collections on personal or academic servers (for example, the **nls2** package from INRA, Huet *et al.* 1996) or services like GitHub.

Using the Task View on *Optimization and Mathematical Programming* (Theussl 2014), which will be abbreviated TVoO, we can list the functions that perform optimization tasks and the packages they belong to. Where there are duplicate names, we will need to provide extra information to identify the individual functions or packages. Besides the more than 66 R packages listed by the TVoO at 2014-08-08, there is also a set of interface functions to AD Model Builder, which uses Automatic Differentiation to obtain derivatives for its optimization tools. There is also **nlmrt** that is intended to handle difficult nonlinear least squares problems as well as those with small or zero residuals, which `nls` specifically excludes.

This large number of NLPE tools speaks to the capabilities of R, but users new to the area need guidance or else strong default R capabilities. Even listing the number of packages, their ongoing modification, and the variety of methods and options they offer is a serious challenge to dedicated volunteers who try to maintain the TVoO.

It is also important that optimizers used within other R packages are as good as possible. Unfortunately, some packages call the internal parts of optimization tools. For example, `nnet` directly calls the internal C routine `vmmin.c` which is the principal computational engine of the `optim()` method `"BFGS"`. This adds a layer of investigation and work compared to packages which use a more standard call to optimizers. It was encouraging while preparing this article

that several package developers I talked to were clearly interested in making their packages capable of using different optimizers as they became available.

5. Current structures

NLPE tools currently in the default R facilities address the following needs:

- A relatively robust method that does not require derivatives (`optim:Nelder-Mead`). This is a derivative-free method using heuristics to search the domain.
- A low-memory optimizer for unconstrained problems with large numbers of parameters (`optim:CG`). This conjugate-gradient code uses just a few vectors but attempts to generate search directions that are relatively efficient in minimizing the objective function. Unfortunately, this code, in any programming language, has given me the least satisfaction of any that I have implemented. Much better implementations of similar ideas are available, e.g., **Rcgmin** (Nash 2013b), and there is some recent literature suggesting further improvements may be possible.
- A simple unconstrained variable metric/quasi-Newton method (`optim:BFGS`). This algorithm solves an approximate version of the Newton equations (to seek a zero of the gradient) by using an approximation of the inverse Hessian and multiplying this times the gradient. A line search is applied to the resulting search direction, and a new trial solution found. The inverse Hessian is updated at each iteration. In this simplified variant, a 'backtrack to acceptable point' line search is used. The inverse Hessian is updated by the Broyden-Fletcher-Goldfarb-Shanno formula, giving the method its acronym.
- A modest-memory optimizer for bounds constrained problems (`optim:L-BFGS-B`). The inverse Hessian in `optim:BFGS` need not be stored explicitly, and this method keeps only the vectors needed to create it as needed. There are, however, many bells and whistles on this code, which also allows for bounds constraints. Both the author and a reviewer have observed disconcerting failures of this method, but it often performs very well. The variant in R is not, however, the most recent one released by the original developers.
- A Newton-like method for unconstrained problems with at least first derivatives (`nlm`). This is a variant of a code by Dennis and Schnabel (1983) that uses only first derivative information to approximate the Hessian (but will use it if supplied) and thereby solve the Newton equations for a zero of the gradient. This complicated code often performs well.
- A bounds constrained quasi-Newton method (`nlminb`). This is a complicated code by David Gay in the Bell Labs **PORT** library collection (Fox, Hall, and Schryer 1978; Fox 1997). It allows bounds constraints and uses a quasi-Newton method.
- A stochastic method that does not require derivatives (`optim:SANN`). This method is unlike most others in that it does not have a termination test, but always evaluates the objective function the specified `maxit` number of times.
- A nonlinear least squares solver (`nls`). This suite of tools solves the Gauss-Newton equations (an approximation to the Hessian for sums-of-squares is used). This is very efficient when it works, but often fails.

- A linear inequality-constrained optimization tool (`constrOptim`).
- A one-parameter minimizer (`optimizer`).
- A one-parameter root-finder (`uniroot`).

The default tools do not appear to contain facilities for

- Mathematical programming – linear, quadratic, integer, mixed-integer, or nonlinear, apart from `constrOptim`.
- Solutions of nonlinear equations, except for one-parameter root-finding.
- Convex optimization, e.g., Boyd and Vandenberghe (2004).
- Constraint satisfaction.
- A method for global optimization that includes some facility for measuring success.
- Optimization of objectives that cannot be measured or evaluated precisely, sometimes referred to as noisy or fuzzy optimization problems, for example, as in Moré and Wild (2009) or Joe and Nash (2003).

Methods capable of using parallel evaluation of function or gradient information appear to be of interest from mailing list or similar postings, but at the time of writing do not seem to be publicly available in R packages.

The lists above are not a judgement on whether facilities should or should not be in the default collection, as that requires a consideration of user needs. They do, however, raise the question of how decisions are made to include or – possibly more contentiously – remove facilities. Who decides for R is clear – the R Core Team. Most large open-source projects have a similar structure, though R is possibly less democratic than some others in that membership of the group is decided by that group and not by the R Foundation. That there has been very little complaint suggests that so far the R Core Team has made mostly acceptable choices.

Unfortunately, in NLPE and likely in other sub-domains, the default tools are not best practice, and the model of an aging default collection and an unstructured, largely un-mapped host of contributed packages is at best unattractive. It is certainly an invitation to misapplication of tools. Sometimes this results in ill-informed complaints on the R mailing lists, wasting time of those who try to help out.

The history of open-source projects is that they either adapt or else are forked. Users rapidly adopt a well-executed fork or repackaging – for example, Ubuntu from Debian Linux. However, forks and variants represent a division of energies, and for a scientific project like R, adaptation is surely preferred to forking, and the rest of this article suggests some pathways to do this adaptation.

6. Classification-driven sets of methods

One of the most important aspects of software is that one can rename objects so that improved routines can be called easily. For NLPE methods, it is often relatively straightforward to

identify which general family of optimizer is indicated for a particular end-use. Thus, if a function currently calls `optim:CG`, substitution to call, say, **Rcgmin** does not take much work. While there are exceptions, most optimization methods likely to be of general applicability can be classified as in the *Current structures* section above. As a reviewer has pointed out, particular statistical applications may lead to very specialized methods being better suited for production usage. However, reliable general-purpose optimizers can be very helpful when there are variations or constraints required, or in reproducing and testing solutions from specialized tools.

A proof of concept package **optreplace** (Nash 2013a) has been created to test the idea of substitution of method. Given its experimental purpose, **optreplace** is not intended to become a regular CRAN package. In it the `nmkb` routine from **dfoptim** (Varadhan and Borchers 2011) is used instead of the legacy `"Nelder-Mead"` method, **Rvmmin** (Nash 2013c) is used for `"BFGS"` and `"L-BFGS-B"` (and also whenever bounds are given with no method specified), and **Rcgmin** replaces `"CG"`. Specifying `"SANN"` as the method simply stops the execution of `optreplace()` at the moment; that is, **optreplace** offers no similar capability. Thus we include a general purpose derivative-free method, a quasi-Newton method and a conjugate-gradient method, all of which support bounds constraints. Using their storage needs, the last two methods might be more generally classified as a 'Hessian approximation method' and a 'vector-only gradient method'. We have (temporarily) avoided the need to have two quasi-Newton methods (`"BFGS"` and `"L-BFGS-B"`) by using the bounds constraint capability of **Rvmmin**, but note that this requires a full sized matrix for the Hessian inverse approximation. We have made no attempt yet to suggest replacements for `nlm`, where it would be very useful to add bounds-constraints capability to the Newton-like method. Nor have we so far delineated when `nlminb` is preferred to other quasi-Newton methods.

6.1. What is improvement?

We note that `nmkb()` seems to offer advantages beyond simply adding bounds constraints to the Nelder-Mead option of `optim()`. My experience and some simple test scripts suggest that it offers a modest advantage in number of function evaluations to termination, though the test function, starting rules, and convergence tolerances will vary the results. This can be illustrated with the simple function

```
R> qfn <- function(x) {
+   d <- 2
+   ff <- 1
+   for (i in seq_along(x)) ff <- ff + (x[[i]] - i)^d
+   ff
+ }
```

Using 1000 starting values generated by `runif()` between 1 and 6, we use `optim()` with `"Nelder-Mead"` and `nmk()` and compare results (Figure 1).

```
Average function evaluations (NM vs NMk):  115    105.674

Number of cases (of 1000 ) where NM ends with lower fn than NMk is  309
```

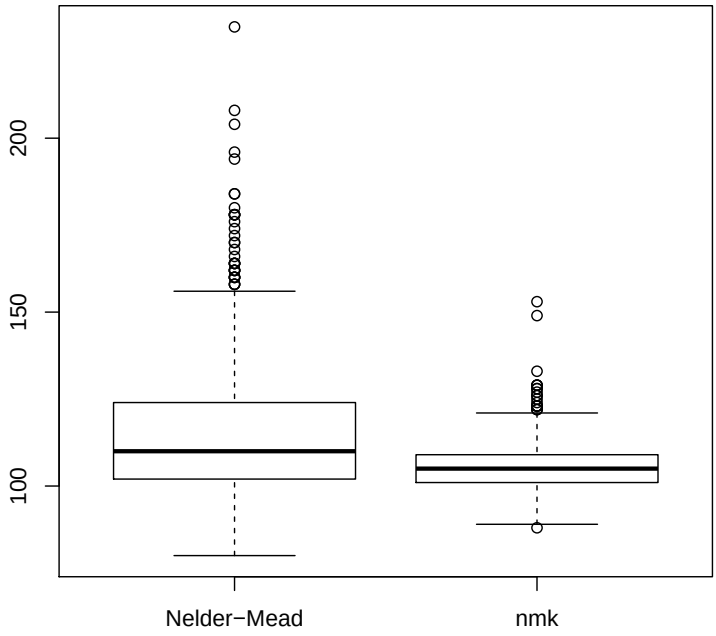

Figure 1: Function evaluations for two Nelder-Mead methods

The main purpose in the example above is to show how some of the experience may be gathered for comparing approaches. Different test functions or functional forms may, of course, alter the advantages seen. In particular, for work outside this paper I have compared methods on three forms of the logistic growth curve estimated for data given in (Nash 1979, Table 12.1), which presents weed density (y) at each year (t). The three forms are:

$$\begin{aligned} (\texttt{unsc}) \quad & y = b_1/(1 + b_2 \cdot \exp(-b_3 \cdot t)) \\ (\texttt{scal}) \quad & y = 100 \cdot x_1/(1 + x_2 \cdot 10 \cdot \exp(-0.1 \cdot x_3 \cdot t)) \\ (\texttt{bates}) \quad & y = q_1/(1 + \exp((q_2 - t)/q_3)) \end{aligned}$$

Using 100 starts (rather than 1000 as above to avoid excessive run time), but this time on the interval 0 to 5, we find many attempts that do not result in a satisfactory solution. Some starting points with some methods lead to points where the functional surface is essentially "flat" but nowhere near the global minimum sum of squares at the value approximately 2.587. Many problems share similar pathologies, and the details take too long to include here, but it is useful to compare how many solutions are found with a sum of squares not exceeding 2.6.

```
             model
meth          bates scal unsc
  Nelder-Mead    58   93   60
  nmkb          100  100   71
```

Here we see that `nmkb()` does better than the legacy Nelder-Mead. However, the story is nuanced by the function evaluation count, given in Figure 2, where `nmkb()` apparently uses more function evaluations. Note, however, that the function evaluation graphs are only for

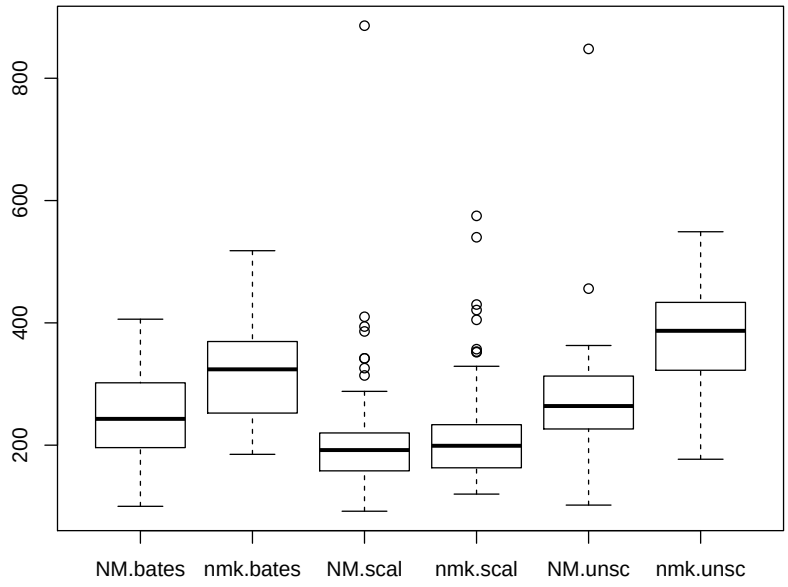

Figure 2: Function evaluations for logistic fits.

the "successful" runs where the sum of squares does not exceed 2.6. The legacy Nelder-Mead has got to the answer quicker when it succeeds, but we have not counted the cases where it did not get to the answer.

6.2. Issues of detail

Clearly different choices for optimizers can be made to substitute for the legacy methods via mechanisms like **optreplace**. The main detail work in doing this is identifying the calls and providing the appropriate substitutions, but there are parallel developments that would simplify the structure and avoid confusion.

- Methods written in C or Fortran should be called via an R language handle. This allows easier substitution of the optimizer. For example, `nnet` now directly calls `vmmin.c` rather than specifying `"BFGS"` as the `optim()` algorithm. It would be useful, of course, to measure the performance cost of such a change.

- The structure of some of the internal routines such as `optim.c` or `nls.c` is complicated, and the human knowledge needed to maintain or adjust them is a relatively scarce resource. They were written when the speed of C was required to make R viable. Even if performance dictates that codes remain in these languages, for reliability and long-term stability of the R infrastructure it is useful to have reference implementations of methods in R. **Rvmmin** is an implementation of the algorithm used for `optim:BFGS`. At the time of writing this article, iterates for equivalent invocations are slightly different, possibly due to minor differences in the implementation. Reference implementations are a tool that has been used with success to avoid gross errors and provide a fall-back capability, for example, with the Basic Linear Algebra Subroutines (BLAS, Lawson, Hanson, Kincaid, and Krogh 1979). While R implementations of optimizers are somewhat slower than C or Fortran variants, my experience is that they are generally fast enough for everyday

usage, but using compiled languages for the objective function and gradient may give the most useful speed improvement.

- As some codes become less commonly used, some storage and transmission overhead could be saved by dropping them from the base or recommended distribution files. However, discussions with various R users and developers in preparing this article suggest that it is important that at least some default NLPE methods be kept available in the basic distribution of R. In any software system, removal of deprecated features is a difficult task.

7. Providing guidance on best practice

Up to now, the R Core Team has decided on the defaults. **optreplace** shows one way to redefine the defaults that does not require the involvement of the R Core Team, though of course any cooperation would be welcome. More importantly, if some form of classification-driven selection of methods is implemented, then it is organizationally feasible to obtain opinions on the default choices for NLPE methods, and by extension for other domains, from knowledgeable community members. How this is accomplished is less important here than the concept that practitioners could guide, test and evaluate different choices. The default set of optimizers could then be chosen by a committee of experts formally established to provide the input, or by a choice made by the R Core Team from a list of candidate methods suggested through the TVoO or other discussions. User groups could also develop their own sets of choices.

A default set of methods is a silent but powerful indication to novices that leads to method selection, possibly without thought. However, many users do seek information. The current documentary guidance for users of optimization tools is essentially contributed material in the form of the TVoO, mailing list contributions, journal articles, vignettes or blog entries. Much of this is of value and some of high quality. However, it is a large and not well-structured body of material. How can users be more efficiently educated on their choices?

It is worth underlining that the current role of the TVoO in listing available packages is very important. It provides a starting point for evaluating options, and the author is of the opinion that it is useful to keep that role of the TVoO separate from the selection of 'best practice' methods and from the 'wish list' of suggested needs for new packages. These latter activities are and should be matters of debate. However, even though I am active as a developer of NLPE tools, I have found items in the TVoO of which I was previously unaware.

One needed tool for guidance of users is a way to view the CRAN repository from different perspectives. Currently only package lists by date and name are provided, along with a set of task views. With several thousand packages, the human search time is becoming onerous. A consequence may be that users do not investigate new tools that could be important for their work.

For NLPE problems, structured listings of packages suitable to various specializations would be helpful in reducing the volume of material the user must peruse. Moreover, like the Task Views, the development of different perspectives on the collection does not require direct involvement of the team maintaining CRAN. Of course, such efforts could be assisted by package developers providing appropriate descriptors and keywords. Two reviewers noted that `RSiteSearch()` (`http://finzi.psych.upenn.edu/search.html`) aided by **sos** (Graves,

Dorai-Raj, and François 2013) could be useful in searching for tools that satisfy particular needs. Ultimately, the usefulness of such information must be tested by actual trials. Feeding such outcomes back into documentation to help users remains, of course, a challenge.

8. Gathering evidence to support best practice decisions

Guidance on computational tools should be based on evidence. Far too many mailing list postings, and even a few published articles, claim 'method X outperforms method Y' based on the evidence of one or a few example problems. Worse, there may be many problems where 'method Y' does very well but 'method X' has not been tested. Performance of NLPE tools is extremely sensitive to starting parameters, termination tests, and to the setup, type and structure of problems. We have as an example one problem where two formulations, both reasonable, show an 800-fold difference in the time to compute the objective function. See, for example, `https://stat.ethz.ch/pipermail/r-devel/2011-August/061796.html` and related postings; the difference was reduced to a factor of 80 by Michael Lachmann by programming changes. With such performance variations, it is important to be very careful how comparisons are made.

Unfortunately, benchmarking is a tedious and often poorly-rewarded task, made more difficult if the software to be compared uses different calling sequences and outputs. Experience with the **optimx** wrapper suggests that standardization does help. **optimx** allows users to choose a list of methods, or else all applicable methods. It would be fairly easy to add specialized groups of methods, for example, all derivative-free methods or all gradient-based methods not storing a Hessian approximation. This would permit easier comparison of candidates that are best-practice within one group. The author welcomes discussion and collaboration to that end.

Benchmarking problems should, of course, reflect the needs of users. For problem classes where R has support for solving NLPE problems, methods are often compared using published benchmarking results for certain well-known problem sets. This allows for disciplined comparisons, but traditional test problems may be uncharacteristic of actual usage patterns. For example, the problems in the packages **NISTnls** (Bates 2012) and **NISTopt** (Nash 2012) which are ports of the problems described in National Institute of Standards and Technology (1999), though interesting, are extreme cases in my opinion. It would be helpful to be able to sample the experiences of real users. This has been proposed informally on a wiki run by the author, using tools with names like `gatherreport()` and `sendreport()` that would interact with instrumented versions of optimization codes. While this requires the cooperation of users, it may be workable with appropriate safeguards that anonymize the users and problems, while still providing some characterization of the problem size in terms of data, number of parameters, and possibly the types of computations involved (i.e., functions called).

With whatever data of this genre that can be collected, R itself can be used to build ongoing profiles of usage and performance. Even an imperfect picture would be closer to the actual experiences of R users for the NLPE class of problems than many of the traditional test functions. Moreover, the availability of such data could advance development if it indicates

- that methods for a particular problem are either satisfactory or not; or
- that efficient tools for a problem class are not presently represented in the R packages.

Besides a large number of methods, the user must sometimes cope with a plethora of options and controls for each of them. A considerable part of the effort in advancing **optimx** has been to try to simplify everyday usage while maintaining the flexibility to use options when they are required. This work is far from finished.

A related, and equally difficult, implementation issue concerns the provision of diagnostic information to verify that a solution has been found. Since many problems have multiple solutions to the optimization sub-problem, an answer that simply seems wrong may still be a valid output from the optimizer. `optimx()` will attempt to compute the Kuhn-Karush-Tucker (KKT) tests of gradient and Hessian (Karush 1939; Kuhn and Tucker 1951). A concern is that computing the KKT tests can cost multiples of the cost of performing the optimization, so `optimx()` does not carry out the KKT tests when there are large numbers of parameters, and this criterion is adjusted downwards for methods requiring matrix storage. An experimental version of **optimx** also tried an axial search post-termination, which gives measures of symmetry and curvature or restarts the optimizer if a better point is found. This approach (Nash and Walker-Smith 1987) is useful as a way to avoid premature termination, but is costly in terms of function evaluations. This is a common dilemma for the developer.

9. A way forward

The ideas above, I believe, argue strongly that `optim()` should be deprecated, using a warning message and/or a user-reversible replacement mechanism as suggested by the **optreplace** experiment. However, I agree with one of the reviewers that *"I'd rather see `optim()` fall into disuse because some package provides a clearly better choice for most users, rather than see changes made to it that mess up legacy code."* My recommendations for moving forward with NLPE tools in R are:

- that `optimx()`, or preferably an evolved replacement, be adopted by users for NLPE tasks and be added to the recommended packages;
- that work to simplify the task of obtaining guidance on NLPE methods be encouraged, such as developing customized views of package lists for particular classes of problem; and
- that interested R users work to provide wrappers and interfaces to simplify the usage of NLPE tools. In particular, stochastic optimization methods – in the family of `optim:SANN`, **DEoptim** (Mullen, Ardia, Gil, Windover, and Cline 2011) and others – and constraint-based optimization methods from the mathematical programming area deserve attention.

10. Conclusion

The main concerns of this paper are that R, as a collective system, should provide tools to solve problems of the NLPE class that:

- offer default methods that are as good as possible;
- allow users to learn about and differentiate packages and methods easily;

- provide a structure allowing for the evolution of methods as the general consensus in the R community evolves as to what is best practice.

Choices need to be presented in clearer ways to both novice and experienced users. Default tools need to be good, or at least defensible, and it should be easier to replace them, for example in a fairly straightforward wrapper infrastructure. Users happy with the defaults need make no extra effort, but those needing either legacy or special methods should have them available, and with a consistent interface and the availability of tests on inputs and outputs. For those needing still more flexibility, there is still the option of a separate package.

Some illustrations of what might be possible have been presented. The experimental package **optreplace** shows how the functions of `optim()` could be replaced in an R session. **optimx** provides a template for a common front-end to many optimizers, and is evolving as a tool that provides guidance on their usage.

Putting best-practice methods into R is more a social and political than technical issue. If appropriate support is provided, R community members can provide input for the selection of suitable choices of the methods that are the defaults for various categories of calculations. Ultimately, making available the best possible tools to users is basic to the continued use and development of R.

Acknowledgments

Exchanges with or software help from John Fox, Ben Bolker, Ravi Varadhan, Gabor Grothendieck and others have been extremely helpful in developing the ideas in this paper. The reviewers were thorough and their comments useful in clarifying and sharpening the material. Discussions with a number of R users in the preparation of Nash (2014a) led to improvements in this article.

References

Bates D (2012). ***NISTnls**: Nonlinear Least Squares Examples from NIST*. R package version 0.9-13, URL http://CRAN.R-project.org/package=NISTnls.

Bolker B, R Core Team (2014). ***bbmle**: Tools for General Maximum Likelihood Estimation*. R package version 1.0.17, URL http://CRAN.R-project.org/package=bbmle.

Boyd S, Vandenberghe L (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA. ISBN 0521833787.

Dennis JE, Schnabel RB (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs.

Fox PA (1997). "The **PORT** Mathematical Subroutine Library, Version 3." URL http://www.bell-labs.com/project/PORT/.

Fox PA, Hall AD, Schryer NL (1978). "The **PORT** Mathematical Subroutine Library." *ACM Transactions on Mathematical Software*, **4**(2), 104–126.

Graves S, Dorai-Raj S, François R (2013). ***sos**: Search Contributed R Packages, Sort by Package*. R package version 1.3-8, URL http://CRAN.R-project.org/package=sos.

Huet S, *et al.* (1996). *Statistical Tools for Nonlinear Regression: A Practical Guide with S-PLUS Examples*. Springer-Verlag, Berlin.

Joe H, Nash JC (2003). "Numerical Optimization and Surface Estimation with Imprecise Function Evaluations." *Statistics and Computing*, **13**(3), 277–286.

Karush W (1939). *Minima of Functions of Several Variables with Inequalities as Side Constraints*. Master's thesis, Department of Mathematics, University of Chicago, Chicago. M.Sc. Dissertation.

Kuhn HW, Tucker AW (1951). "Nonlinear Programming." In J Neyman (ed.), *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492. Berkeley.

Lawson CL, Hanson RJ, Kincaid DR, Krogh FT (1979). "Basic Linear Algebra Subprograms for Fortran Usage." *ACM Transactions on Mathematical Software*, **5**(3), 308–323.

Moré JJ, Wild SM (2009). "Benchmarking Derivative-Free Optimization Algorithms." *SIAM Journal on Optimization*, **20**(1), 172–191.

Mullen K, Ardia D, Gil D, Windover D, Cline J (2011). "**DEoptim**: An R Package for Global Optimization by Differential Evolution." *Journal of Statistical Software*, **40**(6), 1–26. URL http://www.jstatsoft.org/v40/i06/.

Nash JC (1979). *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Adam Hilger, Bristol. Second Edition, 1990, Bristol: Institute of Physics Publications.

Nash JC (2012). ***NISTopt**: Nonlinear Least Squares Examples from NIST in Form of Functions for* `optim()` *and* `optimx()`. R package version 2012-3.12/r572, URL http://R-Forge.R-project.org/projects/optimizer/.

Nash JC (2013a). ***optreplace**: Trial Package to Replace* `optim()` *Function with Better Codes*. R package version 2013.7-23/r750, URL http://R-Forge.R-project.org/projects/optimizer/.

Nash JC (2013b). ***Rcgmin**: Conjugate Gradient Minimization of Nonlinear Functions with Box Constraints*. R package version 2013-02.20, URL http://CRAN.R-project.org/package=Rcgmin.

Nash JC (2013c). ***Rvmmin**: Variable Metric Nonlinear Function Minimization with Bounds Constraints*. R package version 2013-11.11, URL http://CRAN.R-project.org/package=Rvmmin.

Nash JC (2014a). *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons, Chichester.

Nash JC (2014b). ***nlmrt**: Functions for Nonlinear Least Squares Solutions*. R package version 2014.5.4, URL http://CRAN.R-project.org/package=nlmrt.

Nash JC, Varadhan R (2011). "Unifying Optimization Algorithms to Aid Software System Users: **optimx** for R." *Journal of Statistical Software*, **43**(9), 1–14. URL http://www.jstatsoft.org/v43/i09/.

Nash JC, Walker-Smith M (1987). *Nonlinear Parameter Estimation: An Integrated System in BASIC*. Marcel Dekker, New York. See http://www.nashinfo.com/nlpe.htm for a downloadable version including some extras.

National Institute of Standards and Technology (1999). "Statistical Reference Datasets – Nonlinear Regression." Information Technology Laboratory, URL http://www.itl.nist.gov/div898/strd/nls/nls_main.shtml.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Theussl S (2014). "CRAN Task View: Optimization and Mathematical Programming." Version 2014-08-08, URL http://CRAN.R-project.org/view=Optimization.

Varadhan R, Borchers HW (2011). ***dfoptim**: Derivative-Free Optimization*. R package version 2011.8-1, URL http://CRAN.R-project.org/package=dfoptim.

Wikipedia (2014). "Limited-Memory BFGS — Wikipedia, The Free Encyclopedia." URL http://en.wikipedia.org/wiki/Limited-memory_BFGS, accessed 2014-09-04.

Ypma J (2014). ***nloptr**: R Interface to **NLopt***. R package version 1.04, URL http://CRAN.R-project.org/package=nloptr.

Zhu C, Byrd RH, Lu P, Nocedal J (1997). "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization." *ACM Transactions on Mathematical Software*, **23**(4), 550–560.

Affiliation:

John C. Nash
Telfer School of Management
University of Ottawa
55 Laurier Avenue E
Ottawa, ON K1N 6N5, Canada
E-mail: nashjc@uottawa.ca

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 60, Issue 2 *Submitted:* 2012-12-26
September 2014 *Accepted:* 2014-08-05