Journal of Statistical Software

August 2011, Volume 43, Issue 9. *http://www.jstatsoft.org/*

Unifying Optimization Algorithms to Aid Software System Users: optimx for R

John C. Nash
University of Ottawa

Ravi Varadhan
Johns Hopkins University

Abstract

R users can often solve optimization tasks easily using the tools in the `optim` function in the **stats** package provided by default on R installations. However, there are many other optimization and nonlinear modelling tools in R or in easily installed add-on packages. These present users with a bewildering array of choices. **optimx** is a wrapper to consolidate many of these choices for the optimization of functions that are mostly smooth with parameters at most bounds-constrained. We attempt to provide some diagnostic information about the function, its scaling and parameter bounds, and the solution characteristics. **optimx** runs a battery of methods on a given problem, thus facilitating comparative studies of optimization algorithms for the problem at hand. **optimx** can also be a useful pedagogical tool for demonstrating the strengths and pitfalls of different classes of optimization approaches including Newton, gradient, and derivative-free methods.

Keywords: minimization, maximization,wrapper, R, scaling, Newton, gradient.

1. Background

Most modern software systems for scientific computation are now large, complex aggregations of tools and interface components. The system of interest to us here, R (R Development Core Team 2011), is both a statistical package and computing language with extensive computational and graphical capabilities. It is not alone. There are other statistical tools (SAS, Genstat, Stata), as well as mathematical packages such as Mathematica, Maple, MATLAB, and Euler, and this is but a small sample of the available options.

Users of such complex systems often have a choice between several different tools for the solution of a given problem. Novice users may have difficulty appreciating which is appropriate to their situation. There may also be different interface or syntax requirements to use each tool. Optimization tools, in particular, frequently present users with difficulties. Users who

are experienced with the software system may still be unfamiliar with the vocabulary and context of optimization. It is clear to us from postings on the R-help mailing list (https://stat.ethz.ch/mailman/listinfo/r-help) that many users of `optim` and related tools are poorly informed about optimization, even if they have strong statistical knowledge. For example, few statistical workers are experienced in modifying the expression of mathematical functions to render them numerically stable (re-parameterization). Most users, ourselves included, find it tedious or difficult to provide gradient information. There may be multiple optima, plateaux or nearly flat regions on the functional surface, so that optimization tools can return answers that are not acceptable to the user as solutions to the problem at hand, even if they satisfactorily approximate the given optimization sub-task. McCullough and Renfro (2000) and (McCullough 2004, Chapter 8) point out that there are many reasons that a solver, unknowingly misused by a researcher, can return an incorrect answer. Sometimes, these inaccuracies can have serious implications as exemplified by Dominici (2002).

Three of the methods in `optim` (`"Nelder-Mead"`, `"CG"`, `"BFGS"`) were chosen and adapted by one of us (Nash 1979) for use on computers with at most 8K bytes (that is K, not M) of memory for both program and problem data. The Pascal codes in the second edition of Nash (1979) in 1990 were converted to C using **p2c** (Gillespie 1993) and then interfaced to R by Brian Ripley. At the time this was done, BASIC codes were available that have extended features such as parameter bounds (box constraints, Nash and Walker-Smith 1987). Some of these ideas have recently been made available in R packages **Rcgmin** (Nash 2011a) and **Rvmmin** (Nash 2011b).

The 1970s vintage `optim` methods are still usable and useful. However, statistical problems and optimization techniques have evolved. There are also useful optimization packages for R exist outside of `optim`. Besides updating of some current R implementations of `optim` and other optimization tools, tools to help users to better set up their problems and to understand the output from optimization tasks are also needed.

Ultimately, we would like a package option `"GUIDED"` to assist the user in developing his or her problem and in calling the appropriate function(s). In this, we take inspiration from the decision tree for optimization software of (Mittelman 2008). However, our present work has more limited aims.

We have aggregated a useful set of the existing tools for optimizing mostly smooth functions of unconstrained or bounds-constrained parameters under a common calling syntax that is close to that of `optim`. Within the **optimx** wrapper we have included checks on the parameter bounds, if supplied; a computation of some scaling diagnostics; and some solution analysis, primarily checks on the Karush-Kuhn-Tucker conditions (see Chapter 3 of Gill *et al.* 1981). The process of developing **optimx** has revealed the majority of the issues we are likely to encounter in building a comprehensive interface to R optimization facilities in cooperation and collaboration with others.

Thus, this article describes and justifies the main features of **optimx** (Nash and Varadhan 2011), shows how this tool might be used, and presents some of the software engineering issues encountered in building and maintaining an optimization infrastructure like **optimx**.

2. Function minimization approaches

In order that a single function call can be organized to address a variety of methods, it is helpful to have a categorization of optimization methods, which this section will supply.

We will focus on the minimization of general nonlinear functions of n parameters. These parameters may have bounds, often called box constraints. Thus, we wish to solve the problem

$$x^* = \text{argmin}(f(x)) \quad \text{subject to} \quad L \leq x \leq U \tag{1}$$

where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \mapsto \mathbb{R}$. Ideally, we would like to assume $f()$ is smooth, that is, the function and all its derivatives are continuous. Many statistical problems do not fit this assumption, but are sufficiently well-behaved that the function to be minimized is continuous, with first and second derivatives defined and continuous in all but a restricted subset of arguments. Problems with non-smooth objectives are the subject of ongoing work outside the scope of this article.

To maximize a function, we will minimize the negative of that function. The special case that $f()$ is a sum of squared components (nonlinear least squares) has a large place in statistical computing and also has special tools in R for its solution, in particular the function `nls` based on Bates and Watts (1988). Such problems, for which general optimization tools are usually not as reliable or efficient, are nonetheless common as tests, including many of the test problems we use.

Methods for function minimization are most easily categorized by their use of derivative information. Derivative-free methods require only the function to be evaluated. Gradient methods use function values and gradients. Newton-like methods employ the second derivatives of the function (Hessian matrix) and the gradient as well as the function value. Different conditions allow different methods to be best suited to a particular problem, which may require specialist knowledge. This has led to a lively debate on the level of knowledge users need to safely use optimization tools intelligently.

It is also common to use numerical approximation of derivatives so that gradient methods can be used when only function values are provided, but a first-order finite-difference approximation of a gradient requires p function evaluations per iteration, and this can be time-consuming. Our experience and that of others is that analytic derivatives usually, but not always, give more accurate solutions much more efficiently.

3. Tools in R for function minimization

Let us consider some of principal tools in R for the problem defined in Equation (1). The base package in R has the `optim` function which includes five tools for function minimization: `"Nelder-Mead"`, `"BFGS"`, `"CG"`, `"SANN"` and `"L-BFGS-B"`. The default is the `"Nelder-Mead"` method from the second edition of Nash (1979), which is derivative-free. `"CG"` is a gradient method, while `"BFGS"` and `"L-BFGS-B"` are quasi-Newton methods. `"BFGS"` is the Nash (1979) version of the Fletcher (1970) variable metric method, while `"L-BFGS-B"` is a limited memory quasi-Newton method of Byrd *et al.* (1995). The similarity in the names is unfortunate in hiding rather large differences in underlying algorithms. For example, `"BFGS"` builds an explicit matrix approximation to the inverse Hessian and uses a simple step-length backtrack line search while `"L-BFGS-B"` uses a compact representation for the inverse Hessian and a true line search as well as dealing explicitly with bounds constraints. `"CG"` offers three strategies for the conjugate gradient method from Nash (1979).

The **base** package also has a function `nlm` (Schnabel *et al.* 1985) which is a Newton-like method, and `nlminb` from the Port library (Fox 1997) – unconstrained and box-constrained

optimization using PORT routines written by David Gay at Bell Labs (imported to R by Douglas Bates) – which also uses gradients.

Outside the **base** package, the **BB** package (Varadhan and Gilbert 2009) includes the `spg` function that uses projected gradients (Birgin *et al.* 2000). The most recent version of the package can readily handle linear equality and inequality constraints, where the projection is obtained by solving a quadratic programming sub-problem within each iteration. `spg` can also handle nonlinear constraints provided the user can develop the correct projection for them, but this is no trivial task except under special cases.

ucminf (Nielsen and Mortensen 2011) is another implementation of the (unconstrained) variable metric approach (inverse Hessian BFGS update and line search) that is used by the `optim` method BFGS, but its line search and strategic settings may give better performance. A related effort is **Rvmmin**, an all-R version of the general algorithm with bounds constraints as well as temporarily fixed parameters (masks).

More information than the above has been captured in the Comprehensive R Archive Network (CRAN) task view "Optimization and Mathematical Programming" (Theussl 2011).

Recent innovations in optimization include a modification of the CG code to use a suggestion by Dai and Yuan (2001). There is further work in this vein by Hager and Zhang (2006b) and Hager and Zhang (2006a) which remains to be investigated for its potential for use in R. Nash has coded the Dai/Yuan approach purely in R and added box constraints, yet the totality is no more complex than the existing CG implementation in `optim`. This is available as the **Rcgmin** package.

Another development has been the interfacing of several quadratic approximation approaches of Powell (2002, 2006, 2008) by Katherine Mullen (aided by John Nash, Douglas Bates and Ravi Varadhan) in the form of the **minqa** package (Bates *et al.* 2011). This includes the three derivative-free methods `uobyqa`, `newuoa`, and `bobyqa`. `uobyqa` is Powell's original Fortran code. `uobyqa` appears to be superseded by `newuoa` of which `bobyqa` is the box-constrained version. Ongoing work may allow us to confirm `bobyqa` can optimize unconstrained problems as efficiently as `newuoa`, and whether both dominate `uobyqa`, thereby simplifying the package structure and documentation. Powell believes that users need to set control parameters for his routines, while the general spirit of R functions is that default settings should at least provide safe operation. This is an important issue that requires testing, analysis and compromise to resolve. Indeed the Fortran version of `bobyqa` described in (Powell 2008) was not made available until January 2009 pending selection of some options.

There are other packages concerning optimization among the approximately 3000 available in the CRAN repositories of which the master site is at `http://CRAN.R-project.org/`. Several packages for optimization specialized to problems in genomics are in the **Bioconductor** collection (Gentleman *et al.* 2004). Many developmental packages, including some variants of **optimx** and some of the tools it uses can be found on R-Forge, `http://R-Forge.R-project.org/`. These large collections of packages underline two general and continuing issues for all large scientific computing systems like R, that is, the selection of appropriate packages for given problems and the aggregation of packages to obtain reliable and usable wrapper-tools that are more accessible to users. **optimx** is an attempt to address these issues. Currently, in **optimx** we provide a wrapper function that results in calls to the algorithms listed in Table 1.

Algorithm	Package	Method type	Box constraints
`Nelder-Mead`	**stats**	Derivative-free	No
`BFGS`	**stats**	Quasi-Newton	No
`L-BFGS-B`	**stats**	Quasi-Newton	Yes
`CG`	**stats**	Gradient	No
`nlminb`	**stats**	Newton	Yes
`ucminf`	**ucminf**	Quasi-Newton	No
`nlm`	**stats**	Newton	No
`uobyqa`	**minqa**	Derivative-free	No
`newuoa`	**minqa**	Derivative-free	No
`bobyqa`	**minqa**	Derivative-free	Yes
`Rcgmin`	**Rcgmin**	Gradient	Yes
`Rvmmin`	**Rvmmin**	Quasi-Newton	Yes
`spg`	**BB**	Gradient	Yes

Table 1: Optimization algorithms included in **optimx**.

4. A motivating example

To illustrate why **optimx** is a useful addition to CRAN, we consider the modeling of the mineralization of terbuthylazine (Johannesen and Aamand 2003). This maximum likelihood estimation problem was introduced to us by Anders Nielsen as part of work with the National Center for Ecological Analysis and Synthesis (NCEAS) Non-Linear Modeling Working Group. In abbreviated form, the problem involves the application of "free" terbuthylazine (a pesticide), after which there are kinetic processes by which this is converted to and from a bound state, with a concurrent one-way conversion to a mineralized form. The data consists of the amount of the mineralized form at 26 time points, with initial amounts of free, bound and mineralized form $F_0 = 100$, $B_0 = 0$ and $M_0 = 0$. The model has three equations,

$$\begin{aligned} dB_t/dt &= -k_1 B_t + k_2 F_t \\ dF_t/dt &= k_1 B_t - (k_2 + k_3) F_t \\ dM_t/dt &= k_3 F_t \end{aligned}$$

However, the conservation of material means $M_t = 100 - B_t - F_t$. These equations can be put into a matrix form by writing

$$X_t = \begin{pmatrix} B_t \\ F_t \end{pmatrix} \quad \text{using} \quad X_0 = \begin{pmatrix} 0 \\ 100 \end{pmatrix}$$

with the equations then consolidated as $dX_t/dt = AX_t$ or their solution via the matrix exponential as $X_t = \exp(At)X_0$ where

$$A = \begin{pmatrix} -k_1 & k_2 \\ k_1 & -(k_2 + k_3) \end{pmatrix}.$$

Let Y_t be the observed value of X_t, subject to random error, such that $Y_t = X_t + \varepsilon_t$. We will assume that the errors ε_t, for different t are independent of each other and follow a Gaussian distribution: $\varepsilon_t \sim N(0, \sigma^2)$.

A negative log-likelihood can be computed as function `nlogL` as in the code below. **optimx** can be applied to minimize `nlogL` with respect to the logarithms of the three kinetic parameters (k_1, k_2, and k_3) and σ.

```
R> library("Matrix")
R> library("optimx")
R> dat <- read.table("min.dat", skip = 3, header = FALSE)
R> nlogL <- function(theta) {
+    k <- exp(theta[1:3])
+    sigma <- exp(theta[4])
+    A <- rbind(c(-k[1], k[2]), c(k[1], -(k[2] + k[3])))
+    x0 <- c(0,100)
+    sol <- function(t) 100 - sum(expm(A * t) %*% x0)
+    pred <- sapply(dat[, 1], sol)
+    -sum(dnorm(dat[,2], mean = pred, sd = sigma, log = TRUE))
+  }
```

Now carry out the optimization and display the result.

```
R> fit <- optimx(c(-2, -2, -2, -2), nlogL, hessian = TRUE,
+    control = list(all.methods = TRUE, trace = 2))
R> fit
```

```
                                            par   fvalues      method  fns
11       8.585494, -11.052841, 4.658852, 4.477432  153.3056      bobyqa  137
9     -19.173708, 17.416518, -44.176230, 2.533604  102.7661      Rcgmin  144
1     -27.551413, 26.888311, -64.750701, 2.533706   102.766        BFGS   73
5         -24.57290, 23.52085, -57.43591, 2.53360   102.766         nlm   NA
10    -27.551412, 26.888313, -64.750702, 2.533601   102.766      Rvmmin   77
2     -14.051912, 11.625828, -31.597800, 2.533601   102.766          CG  560
3  -7.2529683, -1.4200733, -3.3243726, -0.3518798  19.26905 Nelder-Mead  223
8      -7.249683, -1.582542, -3.476132, -1.382814 0.9392184      ucminf   40
7      -7.249615, -1.582468, -3.476068, -1.382855 0.9392142         spg  217
4      -7.249608, -1.582466, -3.476067, -1.382813 0.9392142    L-BFGS-B   90
12     -7.249611, -1.582464, -3.476065, -1.382814 0.9392142      newuoa 3017
6      -7.249611, -1.582464, -3.476065, -1.382815 0.9392142      nlminb   32
   grs itns conv  KKT1  KKT2  xtimes
11  NA NULL    0  TRUE FALSE  26.206
9   69 NULL    0  TRUE FALSE  92.218
1   22 NULL    0  TRUE FALSE  46.927
5   NA   26    0  TRUE FALSE  33.906
10  18 NULL    0  TRUE FALSE  31.218
2  101 NULL    1  TRUE FALSE 256.048
3   NA NULL    0 FALSE FALSE  42.199
8   40 NULL    0 FALSE  TRUE  42.663
7   NA  178    0 FALSE  TRUE 176.043
4   90 NULL    0 FALSE  TRUE 153.142
12  NA NULL    0  TRUE  TRUE 569.952
6  120   27    0  TRUE  TRUE  29.021
```

Of the methods in the function `optim`, all but method SANN are subsumed in **optimx**, but among them only L-BFGS-B has succeeded. Moreover, we copied Anders Nielsen's formulation of this problem that works with the logarithm of the parameters, guaranteeing that they are positive and in some cases improving the scaling of the problem. When we used a formulation in the direct scale of the parameters, with an approximately equivalent starting point, again we found none of the `optim` methods gave a satisfactory minimum approximation, but some of the additional methods in **optimx** did reasonably well, though the optimality tests (referred to as KKT1 and KKT2 in the output and discussed later) were reported as not satisfied.

We believe that it is important to be able to provide at least approximate solutions when problems are presented in rather poor formulations, even as we wish to encourage and provide guidance to better approaches. This has motivated the scaling and Karush-Kuhn-Tucker (KKT) optimality tests in **optimx**.

5. From `optim` to optimx

We want **optimx** to offer a pathway to update the optimization methods in legacy codes where `optim` is called by some R tools, for example, in time series and neural networks problems. Thus **optimx** has a calling sequence intentionally similar to that of `optim`, namely,

```
optimx(par, fn, gr = NULL, hess = NULL, lower = -Inf, upper = Inf,
  method = c("Nelder-Mead", "BFGS"), itnmax = NULL, hessian = NULL,
  control = list(), ...)
```

The argument names here are almost identical to those for `optim`, though not necessarily in the same order. `par` provides the vector of initial parameters. `fn`, `gr`, and `hess` are the user-supplied functions to compute the objective function and optionally its gradient and Hessian at a set of parameters. `hess` is not an argument to `optim`, but some methods callable by **optimx** may be able to use the Hessian. `lower` and `upper` are the parameter bounds. `method` is now a vector of methods that will be applied to a given objective function; in `optim` we are restricted to a single choice. The function argument `itnmax` is a vector of iteration limits for the chosen methods and corresponds to the scalar `maxit` element of the control list in `optim`. For consistency with `optim`, the argument `hessian` is permitted as a logical variable indicating whether the computed Hessian at the final parameter estimates should be returned. The default is `hessian = FALSE`, but the element `kkt` of the `control` list, which determines if the KKT optimality conditions will be tested, defaults to `TRUE` for any problem that does not have a large number of parameters, and the Hessian will usually be computed.

The dot-dot-dot argument `...` is used as in many other R packages to provide named data to the objective, gradient and hessian functions. It could also be used to input controls to particular optimizers that are extra to the ones provided by **optimx**, for example, `parscale` values that are used only by some of the methods.

The `control` list serves the same general function as that in `optim`, but there are several new options, some of which we describe below.

Some guidance and ease-of-use issues that we have considered and partially addressed in **optimx** are:

- Indicators that the solution is optimal.
- Scaling tests.
- Permitting maximization of functions by simply setting the control argument `maximize` to `TRUE`.
- Checking box constraints for validity.

For the first of these, we have included code to evaluate the KKT conditions (see Chapter 3 of Gill *et al.* 1981). The first-order KKT test checks whether the gradient at the final parameter estimate is small, and the second-order KKT test checks whether the Hessian at the final parameter estimate is positive definite, except when some of the parameters are on the boundary in which case the Hessian is allowed to be positive semi-definite. KKT tests are not evaluated in high-dimensional problems since the computation of gradients and Hessians can be prohibitively expensive. Note that for efficiency, we compute the Hessian as the Jacobian of the gradient when a gradient function is available.

Bad scaling, in our experience, is a common cause of unsatisfactory results (second only to miscoded gradients for local optimization). `optim` allows for both parameter scaling and function scaling, but these facilities are not, in our view, easy to use, and they complicate the optimizer code. Such scaling facilities are also not a part of many other R optimization packages, making it awkward to provide scaling generally within **optimx** in a consistent manner. However, we include a test on the scaling of parameters and (where supplied) bounds. This test issues a warning when we believe there are serious differences in scale in different dimensions.

To maximize a function, we need to minimize the negative of the objective function. That is, within our code we must redefine the user objective and gradient function to the negative of those supplied. In `optim`, we can use the function scaling to do this by setting `control$fnscale = -1`. In `BB::spg`, there is an explicit control argument `maximize` which can be set to `TRUE` for maximization. We have used this construct in **optimx** because we believe it is simpler for users than `fnscale`. Indeed, **optimx** will stop if users attempt to use both `maximize` and `fnscale`.

Upper and lower bounds on parameters, that is, box constraints, are very helpful in making users think about the scale and nature of the parameters being optimized. Indeed, our experience is that asking the user for bounds encourages the user to propose much better scalings and starting values. We caution that overly stringent bounds can interfere with efficient optimization by some methods. Bounds are permitted in **optimx** via `L-BFGS-B`, `bobyqa` from **minqa**, **Rcgmin**, **Rvmmin**, `nlminb`, and `spg` from **BB**.

There are several issues we considered tackling but, as yet, have only addressed in experimental codes. These are automatic choice of starting values, uncertainty measures on solution parameters, masks (temporarily fixed parameters), and the handling of inadmissible arguments to function or gradient sub-programs.

Users typically do not provide either scale or bounds, but nonlinear optimization codes rarely allow them to escape providing starting parameters. Where particular problems are common, we could follow the approach of the `selfStart` capability for `nls`. Unfortunately, the variety of problems is daunting, and it is likely that progress to automatic selection of starting values will be made where optimization problems arise within other packages, such as those for maximum likelihood or subject-specific problems.

Most statisticians want measures of uncertainty ("standard errors") for the parameters determined from the optimization, i.e., the covariance matrix of parameters at the convergence. This is an important concept, but depends on the origin and structure of the function being minimized. Therefore, we provide for return of the (approximate) Hessian matrix at the suggested optimum by setting `hessian = TRUE`, which measures the curvature of the objective function at the final estimate. Measures of uncertainty can be easily obtained from the Hessian matrix, except when one or more of the box constraints are active at the solution.

In Nash and Walker-Smith (1987) we borrowed an idea of Duggleby (1984) and also allowed masks, that is, parameters which are frozen at a fixed value for a given "run" of a program, but which can be allowed later to participate in the optimization. Masks are part of **Rcgmin** and **Rvmmin**. An interface to masks is planned for **optimx** as soon as some clear practical examples are available, and the essential structure for masks is already present in **optimx**.

Also from Nash and Walker-Smith (1987), we have considered ways to manage inadmissible arguments to the objective function. This is often a cause of failure when, for example, the square root of a negative number or a zero-divide is requested. In such cases, the objective function code should return a value that can be interpreted as "objective is non-computable". Some minimization tools can cope with such returned data. For example, if there is a line search, then we can reduce the step size and try again. However, this implies we need to modify every minimizer. Some experimentation is already in process with an extension of **Rvmmin** on R-forge that is set up to check for and handle "non-computable" returns. However, we consider that we need to work with other developers to gradually integrate such capabilities generally.

6. Capabilities and controls

optimx is a "wrapper" unifying many existing optimization tools under one calling sequence. We want to use this as a prototype upon which to develop and incorporate better user interface capabilities.

In unifying the calls to the various minimizers in R, we were forced to address several particular matters. First, the `control` list varies considerably from package to package, and there is both overlap and inconsistency. For example, output detail is controlled by the integer `trace` in `optim`, the integer `iprint` in **minqa**::`bobyqa`, the logical `trace` in `BB::spg`. We chose in **optimx** to use the integer `trace` to control the volume of output generated.

Different functions have different calling syntax (number, meaning and order of arguments). Cleaning this up is mostly a matter of detail work to recast the invocation. For example `nlm` required us to adapt our wrapper to provide an objective function that can have gradient and/or Hessian attributes specified.

As mentioned earlier, we added code to assess the KKT conditions for a satisfactory local minimum. The control `kkt` defaults to `TRUE` to run KKT tests of results, since we believe they provide important information to users. However, to avoid very long run times, we set `kkt` `FALSE` if the problem has > 500 parameters when an analytic gradient function `gr` is supplied, or > 50 parameters otherwise.

The user may also supply two tolerances `kkttol` and `kkt2tol` that are used to decide if a gradient is near zero, if Hessian eigenvalues are negative, and if the ratio of the extreme Hessian eigenvalues implies that some are essentially zero. The choice of defaults for these

parameters is challenging because of the range of problems and scalings that may be provided by users. At the time of writing, `kkttol` defaults to 0.001 and `kkt2tol` defaults to $1e-6$. Users are advised to consult the source code if they need to delve deeper into the KKT conditions.

The **numDeriv** package (Gilbert 2011) makes it fairly easy to compute numerical approximations to gradients, Jacobians and Hessians. If the user supplies gradient or Hessian functions, we automatically check them before starting the optimization by using **numDeriv** tools and stop if there is a serious discrepancy. Note that some optimization tools, e.g., `Rvmmin`, allow a choice of **numDeriv** or a simple forward difference approach to approximate derivatives, but **optimx** simply passes along a null gradient function so that each method uses its own numerical gradient approximation.

As mentioned above, we include a scale check on parameters and bounds. Currently, we warn when the log of the range of a parameter exceeds 3, i.e., we want parameters to be between 1 and 20 in size approximately. Such tools are imperfect; we welcome discussion.

A logical control `follow.on` is set `TRUE` (default is `FALSE`) in the case that multiple methods are specified will allow the final parameters of one method to be used as the starting set for the next. For example, we might use Nelder-Mead as a first try with fairly arbitrary starting values, then refine the solution with a gradient method such as L-BFGS-B. One of the examples given in the `?optimx` documentation uses a generalized Rosenbrock test function via the invocation

```
R> ans9 <- optimx(startx, fn = genrose.f, gr = genrose.g, hess = genrose.h,
+    method = c("Nelder-Mead", "ucminf"), itnmax = c(200, 75),
+    control = list(follow.on = TRUE, save.failures = TRUE,trace = TRUE),
+    gs = 10)
```

where the number of cycles of each of the Nelder-Mead and **ucminf** methods is controlled by the vector `itnmax`.

The logical control `save.failures` when `TRUE` (default) keeps "answers" from runs where the method does not return with the value of `conv` equal to 0. This is useful to see what a method has done when no apparent solution is obtained.

The logical control `all.methods` is set `TRUE` (default is `FALSE`) if we want to use all methods that are applicable to the problem presented, that is, to the function, gradient, hessian and box constraints. Within the code, it is simple to restrict the list of methods used, as we may wish to avoid unnecessary computations. For example, at the time of writing, `uobyqa` is omitted. We acknowledge that such trimming of the list of methods is contrary to the spirit of `all.methods`, but fear a different name for this control would be cumbersome.

We find that `all.methods` is a very useful tool for comparing the performance of different methods. A package developer who needs to solve an unconstrained or box-constrained optimization problem can use this feature to compare the performance of various optimizers on his class of problems, and choose the most appropriate one for his/her package. We have already used this option in the terbuthylazine example.

The control `sort.result` is set `TRUE` by default to sort results in decreasing order of the final function value so that the bottom of the output table lists the "best" results when multiple methods are run.

A logical control `starttests` defaults to `TRUE` to run tests of the function and parameters for feasibility relative to bounds, to check gradient code against numerical approximations, and to check parameter and bounds scaling before we run optimization methods.

By default, we emit warnings where appropriate, but may suppress them by setting the logical control `dowarn = FALSE`.

On return, **optimx** provides a data frame where each row corresponds to one method. Each row gives similar content to the list returned by `optim` for a single method, with the addition, in particular, of the method name and the execution time. We have, however, chosen to use the name `fvalues` for the returned best function values, rather than the name `value`. There are also logical elements KKT1 and KKT2 reporting the results of (approximate) tests of the Kuhn-Karush-Tucker conditions for gradient and hessian respectively. As with `optim`, termination or "convergence" codes vary with the method. We believe it would be helpful to standardize such codes, and welcome collaboration toward this goal.

We have also chosen to return other information, such as the Hessian approximation at the solution, in the form of attributes. These have a sufficiently complicated structure to warrant writing the utility function `get.result`, which facilitates the retrieval of useful information from the returned data structure. Examples included in **optimx** show the usage. Users need to be aware that **optimx** can return a large volume of data, especially when `all.methods = TRUE`, and similarly generate a large amount of output when `trace` is set greater than 1.

Users intending to embed **optimx** in application packages or scripts should generally avoid calling multiple optimization methods with **optimx**. The facility for calling multiple methods is intended to streamline the *choice* of a method in such tasks, not to run a battery of optimizations on each problem. Note that the "best" solution is, however, available in the final row of the returned data frame. This will be the only row if a single method is called,

7. Conclusion and outlook

Ultimately, we believe the greatest benefit that we can aim to provide is advice, backed by empirical evidence as well as mathematical and computational results, on suitable methods for particular optimization problems that users encounter. Along with many similar software systems, R has far too many tools that cannot easily be differentiated by users. The issue is not that any of these tools are "bad", but that there are so many the user gets confused and may mis-apply them. Even those of us familiar with optimization may have difficulty in discerning subtle yet important differences between methods and how they are invoked. **optimx** is a first step to simplifying at least the mechanical details. Moreover, it can assist the task of building a base of standardized testing to give well-supported advice that is focussed on optimization tools in R.

Such a base of experience is not yet in place. In the course of the present work, we sometimes found that our "traditional wisdom" concerning performance and reliability of methods was at odds with the results of experimentation using **optimx**. The summary table produced by **optimx** for comparing different methods has been very useful in this regard. We intend an ongoing effort to provide easy-to-use software that we hope will allow for collaborative evaluation and analysis of optimization methods and problems, as well as tools for regular application to statistical problems.

optimx is a useful wrapper for unifying the optimization of essentially smooth, box-constrained

problems. Development versions appear first on R-Forge (http://R-Forge.R-project.org/), then move to CRAN (http://CRAN.R-project.org/package=optimx). We are continuing our development of **optimx** and related tools to include more capabilities and to enlarge the understanding of them.

8. Acknowledgments

We appreciate exchanges of ideas with Karl Ove Hufthammer, James Ramsay, Jason Nielsen, Spencer Graves, Paul Gilbert and Bruce McCullough that have helped us to shape the ideas in this paper. We are particularly indebted to Katherine Mullen and Douglas Bates for interfacing Powell's Fortran codes for quadratic approximation to R as well as to Mike Powell himself for generously sharing his programs. Anders Nielsen provided the terbuthylazine problem in the context of the NCEAS Non-Linear Modeling Working Group. The second author (R.V.) was supported by the funding from NIH grant DA023879-01.

References

Bates D, Mullen KM, Nash JC, Varadhan R (2011). ***minqa**: Derivative-Free Optimization Algorithms by Quadratic Approximation*. R package version 1.1.15, URL http://CRAN.R-project.org/package=minqa.

Bates DM, Watts DG (1988). *Nonlinear Regression Analysis and Its Applications*. John Wiley & Sons.

Birgin EG, Martínez JM, Raydan M (2000). "Nonmonotone Spectral Projected Gradient Methods on Convex Sets." *SIAM Journal on Optimization*, **10**(4), 1196–1211.

Byrd RH, Lu P, Nocedal J, Zhu CY (1995). "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing*, **16**(5), 1190–1208.

Dai YH, Yuan Y (2001). "An Efficient Hybrid Conjugate Gradient Method for Unconstrained Optimization." *Annals of Operations Research*, **103**(1-4), 33–47.

Dominici F (2002). "On the Use of Generalized Additive Models in Time-Series Studies of Air Pollution and Health." *American Journal of Epidemiology*, **156**(3), 193–202.

Duggleby RG (1984). "Regression Analysis of Nonlinear Arrhenius Plots: An Empirical Model and a Computer Program." *Computers in Biology and Medicine*, **14**(4), 447–55.

Fletcher R (1970). "A New Approach to Variable Metric Algorithms." *Computer Journal*, **13**(3), 317–322.

Fox P (1997). *The Port Mathematical Subroutine Library, Version 3*. AT&T Bell Laboratories, Murray Hill, NJ. URL http://www.bell-labs.com/project/PORT/.

Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JYH, Zhang

J (2004). "**Bioconductor**: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology*, **5**, R80. URL http://genomebiology.com/2004/5/10/R80.

Gilbert P (2011). ***numDeriv***: *Accurate Numerical Derivatives*. R package version 2010.11-1, URL http://CRAN.R-project.org/package=numDeriv.

Gill PE, Murray W, Wright MH (1981). *Practical Optimization*. Academic Press.

Gillespie D (1993). ***p2c***. Free Software Foundation, Boston, MA. URL http://directory.fsf.org/project/p2c/.

Hager WW, Zhang H (2006a). "Algorithm 851: CG DESCENT, a Conjugate Gradient Method with Guaranteed Descent." *ACM Transactions on Mathematical Software*, **32**(1), 113–137.

Hager WW, Zhang H (2006b). "A Survey of Nonlinear Conjugate Gradient Methods." *Pacific Journal of Optimization*, **2**, 35–58.

Johannesen H, Aamand J (2003). "Mineralization of Aged Atrazine, Terbuthylazine, 2,4-D, and Mecoprop in Soil and Aquifer Sediment." *Environmental Toxicology and Chemistry*, **22**, 722–729.

McCullough BD (2004). "Some Details of Nonlinear Estimation." In M Altman, J Gill, MP McDonald (eds.), *Numerical Issues in Statistical Computing for the Social Scientist*, chapter 8, pp. 199–218. John Wiley & Sons.

McCullough BD, Renfro CG (2000). "Some Numerical Aspects of Nonlinear Estimation." *Journal of Economic and Social Measurement*, **26**(1), 63–77.

Mittelman H (2008). *Decision Tree for Optimization Software*. URL http://plato.asu.edu/guide.html.

Nash JC (1979). *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Adam Hilger, Bristol. 2nd Edition, 1990, Institute of Physics Publications, Bristol.

Nash JC (2011a). ***Rcgmin***: *Conjugate Gradient Minimization of Nonlinear Functions with Box Constraints*. R package version 2011-2.10, URL http://CRAN.R-project.org/package=Rcgmin.

Nash JC (2011b). ***Rvmmin***: *Variable Metric Nonlinear Function Minimization with Bounds Constraints*. R package version 2011-5.09, URL http://CRAN.R-project.org/package=Rvmmin.

Nash JC, Varadhan R (2011). ***optimx***: *A Replacement and Extension of the* `optim()` *Function*. R package version 2011-8.1, URL http://CRAN.R-project.org/package=optimx.

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 of this plus some extras.

Nielsen HB, Mortensen SB (2011). ***ucminf***: *General-Purpose Unconstrained Non-Linear Optimization*. R package version 1.1-2, URL http://CRAN.R-project.org/package=ucminf.

Powell MJD (2002). "**UOBYQA**: Unconstrained Optimization by Quadratic Approximation." *Mathematical Programming*, **92**(3), 555–582.

Powell MJD (2006). "The **NEWUOA** Software for Unconstrained Optimization without Derivatives." *Large Scale Nonlinear Optimization*, pp. 255–297.

Powell MJD (2008). "Developments of **NEWUOA** for Minimization without Derivatives." *IMA Journal of Numerical Analysis*, **28**(4), 649–664.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Schnabel RB, Koontz JE, Weiss BE (1985). "A Modular System of Algorithms for Unconstrained Minimization." *ACM Transactions on Mathematical Software*, **11**(4), 419–440.

Theussl S (2011). "CRAN Task View: Optimization and Mathematical Programming." Version 2011-04-12, URL http://CRAN.R-project.org/view=Optimization.

Varadhan R, Gilbert P (2009). "**BB**: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function." *Journal of Statistical Software*, **32**(4), 1–26. URL http://www.jstatsoft.org/v32/i04/.

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

Ravi Varadhan
Center on Aging and Health
Division of Geriatric Medicine & Gerontology
Johns Hopkins University
United States of America
E-mail: rvaradhan@jhmi.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 43, Issue 9 *Submitted:* 2010-08-12
August 2011 *Accepted:* 2011-08-05