Provenance of R's Gradient Optimizers

by John C Nash

Abstract Gradient optimization methods (function minimizers) are well-represented in both the base and package universe of R (R Core Team, 2019). However, some of the methods and the codes developed from them were published before standards for hardware and software were established, in particular the IEEE arithmetic (IEEE, 1985). There have been cases of unexpected behaviour or outright errors, and these are the focus of the **histoRicalg** project. A summary history of some of the tools in R for gradient optimization methods is presented to give perspective on such methods and the occasions where they could be used effectively.

The task

Ignoring exogenous data (assume that it is supplied when needed), our problem is to find

$$argmin_x f(x)$$

where x is our set of n parameters and $f()$ is a scalar real function of those parameters. The gradient of $f(x)$ is the vector valued function

$$g(x) = \partial f(x)/\partial x.$$

The Hessian of $f(x)$ is the matrix of second partial derivatives

$$H(x) = \partial^2 f(x)/\partial x^2.$$

We will be considering methods for this problem where a gradient can be supplied, though many of the R tools will use a numerical approximation if needed.

General approaches

The so-called **Newton** method tries to find x that satisfies the first order conditions for an extremum, namely,

$$H(x)\delta = -g(x)$$

and then updates x to $(x + \delta)$.
The main objections to this approach are

- the Hessian is generally expensive to compute, though because an accurate Hessian is so useful, I strongly recommend that workers examine whether it can be computed in a reasonable way;
- it is necessary to apply safeguards on the computations and the size of δ to avoid cases where the Hessian is near singular, poorly computed, or the assumptions of the method are violated.

There are four main approaches to simplifying the Newton method:

- The **variable metric** or **quasi-Newton** methods use clever ways to improve an approximate Hessian or Hessian inverse while finding lower points x on the function surface, using only function and gradient values computed in the process.
- Truncated Newton methods use a linear conjugate gradient method to inexactly solve the Newton equations, thereby reducing memory requirements.
- Conjugate gradient methods aim to search "downhill" for better points on the functional surface using only recent gradient information. The traditional first search direction is that of steepest descents, namely, $-g$, the negative gradient. After a line search selects a suitable step size, we have a new point, a new function value that is lower than the initial point, and a new gradient. Using this information and possibly the last search direction, we compute a new search direction that is somehow "conjugate" to the previous one. For a problem with n parameters, of course, there are only n independent search directions, so we need to restart the procedure on or before that step. Indeed there are a number of strategies for deciding when to restart the conjugacy cycle. And, of course, there are a number of choices for updating the search direction and for performing the line search.

- Limited Memory BFGS methods use reduced memory approaches to variable metric methods, but can also be thought of as extending conjugate gradients methods by using several rather than the two most recent gradients.

The histoRicalg project

The R Consortium awarded some modest funding for **histoRicalg**, a project to document and transfer knowledge of some older algorithms used by R and by other computational systems. These older codes are mainly in Fortran, but some are in C, with the original implementations possibly in other programming languages. This effort was prompted by finding some apparent bugs in codes, which could be either from the original programs or else the implementations. Two examples in particular – in `nlm()` and in `optim--L-BFGS-B` – gave motivation for the project. We welcome interest and participation, and have a project repository `https://gitlab.com/nashjc/histoRicalg`, where several aspects of this work are in various stages of completion.

The function `optim()` in base-R has three of the optimizers from the 1990 edition of Compact Numerical Methods (Nash, 1979), abbreviated CNM henceforth. Given the dates of their conception, these are an obvious focus for **histoRicalg**. The direct-search Nelder-Mead method (Nelder and Mead, 1965) is in fact the default solver i.e., it uses `method="Nelder-Mead"` in the `optim()` call and does not require a gradient routine to be supplied. In fact, Nelder-Mead will be used even if a gradient routine is included unless some other method is suggested. The choice if `method="BFGS"` is the variable metric method of Fletcher (1970). The third choice from CNM is `method="CG"` for conjugate gradients, which is a combination of several similar approaches.

In this article on the provenance of the methods, the details will be discussed only in general terms. However, it is critical to get those details right.

Provenance of the R optim–BFGS solver

If the source code for base R is in a directory that is named R-X.Y.Z (in my case 3.5.1 when this vignette was started) then the calling routine for `optim()` is `src/library/stats/R/optim.R`, but this uses the `.External2` method to call `src/library/stats/src/optim.c`, where the function `vmmin` is called. However, the source for `vmmin` is in `src/appl/optim.c`. I will venture that having two files of the same name in different directories is tempting an error.

I will use `optim--BFGS` when using the call `method="BFGS"` in `optim()`, and similar abbreviations for the other methods. To use `optim--BFGS`, the relevant code is a C routine `vmmin`. In `src/appl/optim.c`, above this routine is the comment

```
{,comment}
/*  BFGS variable-metric method, based on Pascal code
in J.C. Nash, `Compact Numerical Methods for Computers', 2nd edition,
converted by p2c then re-crafted by B.D. Ripley */
```

As author of this work, I can say that the code used is Algorithm 21 of (Nash, 1979). In the First Edition, this was a step-and-description code which was replaced by Turbo Pascal in the Second Edition. The methods were worked out mainly on a Data General Nova which had partitions of between 3.5 K and 8 K bytes accessible via a 10 character per second teletype. The floating point available had a 24 bit mantissa, in a single level of precision with (as I recall) no guard digit. Programming was in a fairly early and simple form of BASIC.

This machine was at Agriculture Canada. In the late 1970s, it was replaced with a Data General Eclipse, but largely the facilities were the same, except the floating point went to 6 hexadecimal digits with no guard digit. I also implemented some codes on HP 9830 and Tektronix 4051 "programmable calculators". A Fortran translation was made as NASHLIB, which ran mainly on IBM System 360 class computers, but also was tested at least partially on Univac 1108, ICL 1906A and Xerox-Honeywell CP-6 machines. The codes were distributed as Fortran source code. See https://gams.nist.gov/cgi-bin/serve.cgi/Package/NASHLIB. I have been informed that the servers supporting this link will not be replaced when they fail.

NASHLIB dates from the 1979-1981 period, though the methods were developed in the mid-1970s at Agriculture Canada to support economic modelling. In 1975, I received an invitation (from Brian Ford) to collaborate with the Numerical Algorithms Group in Oxford, and Agriculture Canada generously allowed me two 6-week periods to do so. During the second of these, in the tail end of the hurricane of January 1976, I drove to Dundee to meet Roger Fletcher. He took an interest in the concept of a program that used minimal memory. We found a printout of the Fortran for the method in (Fletcher, 1970), and with a ruler and pencil, Roger and I scratched out the lines of code that were not strictly needed. As I recall, the main deletion was the cubic interpolation line search. The resulting method simply used backtracking with an acceptable point condition.

On my return to Ottawa, I coded the method in BASIC and made two small changes:

- I managed to re-use one vector, thereby reducing the storage requirement to a square matrix and five vectors of length n, the number of parameters.
- The very short floating point precision seemed to give early termination on some problems. Often this was due to failure of the update of the approximate inverse Hessian when an intermediate quantity indicated the approximate Hessian was not positive definite. As a quick and dirty fix, I simply checked if

the current search was a steepest descent. If so, then the method is terminated. If not, the inverse Hessian approximation is reset to the unit matrix and the cycle restarted. This "quick fix" has, of course, become permanent. My experience over the years suggests it is a reasonable compromise, but the possibility of better choices is still open.

Evolution of the code

Considering the simplicity of the method, it is surprising to me how robust and efficient it has shown itself to be over the last four decades.

The code does allow of some variations:

- the organization of some of the calculations may have an influence on outcomes. There are opportunities for accumulation of inner products, and the update of the inverse Hessian approximation involves subtractions, so digit cancellation could be an issue.
- tolerances for termination, for the "acceptable point" (Armijo) condition, and other settings could be changed. However, I have found the original settings seem to work as well or better than other choices I have tried.

In the mid-1980s, the BASIC code was extended to allow for masks (fixed parameters) and bounds (or box) constraints on the parameters (Nash and Walker-Smith, 1987). This adds about 50% more code, as well as vector storage for the constraints and indices to manage them (essentially $3 * n$), but does permit a much wider variety of problems to be solved. Even for unconstrained problems that may have difficult properties, imposing loose bounds can prevent extreme steps in the parameters. To add this capability to R, I put the package `Rvmmin` on CRAN in 2011 (Nash, 2018). This code is, as of 2018, part of a new `optimx` (Nash and Varadhan (2011), Nash (2014a)) package on CRAN. It is entirely written in R.

Extensions and related codes

There are other R packages with related capability. In particular, `ucminf` (Nielsen and Mortensen, 2012) is based on the Fortran code of Nielsen (2000). This appears to use a very similar algorithm to `optim--BFGS`, but employs a more sophisticated line search. This package does not, however, allow for constraints in the form of bounds or masks.

The approximate inverse Hessian could also be saved and used to provide some estimates of parameter dispersion. Clearly, the use of a steepest descents direction for the final line search in `Rvmmin` before termination means that the **penultimate** approximation must be saved. In practice, I have found the approximate inverse Hessian bears little or no resemblance to the actual Hessian. This may be because the construction of the approximation maintains the positive definite condition. It is an open question whether the approximation has any utility.

Package **mize** (Melville, 2017) offers some options to build a gradient minimizer and could possibly allow some algorithmic comparisons to be made, but I have not had the time to delve into this.

Rather more sophisticated codes are part of base R in `nlm()` and `nlminb()`. The former is a polyalgorithm of quasi-Newton type for unconstrained problems (Schnabel et al., 1985). The latter allows bounds constraints and is drawn from the PORT library (Fox, 1997) code by Gay (1990), but there are a number of common ideas in both methods. For users, these two functions generally work well with the default settings of their controls. This is fortunate, as even though I am fairly experienced with such programs, I hesitate to play with the numerous control parameters.

To underline this last sentence, there was a thread on the R-help mailing list in 2010 (see, for example, https://stat.ethz.ch/pipermail/r-help/2010-July/245182.html) noting that the default behaviour of `nlminb()` at that time was to assume a positive objective function such as a sum of squares. Thus the function would terminate if a non-positive evaluation occurred. This has since been corrected, but was present in R for some years, possibly resulting in erroneous results being published.

Provenance of the R optim–CG and related solvers

The original Algorithm 22 of CNM which was converted to C and included in R as the `method="CG"` choice for `optim()` by Brian Ripley is an approach that has never felt quite right to me. And I am the author! It offers three different search direction updates using the `type` element of the `control` list. The default is `type=1` for the Fletcher-Reeves update (Fletcher and Reeves, 1964), with 2 for Polak–Ribiere (Polak and Ribiere, 1969) and 3 for Beale–Sorenson (Sorenson, 1969), (Beale, 1972).

In the mid-1980's I updated the CG code (in BASIC) to handle bounds and masks. Then in 2009 I incorporated the Yuan/Dai (Dai and Yuan, 2001) search direction update that melds the different formulas used in `optim--CG`. This gives a remarkably effective method that retains a surprisingly short code, and entirely in R, with a version that handles bounds and masks. This is package `Rcgmin` (Nash, 2014c).

There has been a flurry of work on CG-related optimizers in the last decade or so, in particular associated with Hager and Zhang. See (Hager and Zhang, 2006a) and (Hager and Zhang, 2006b). I have experimentally wrapped the `CG-Descent` code, but as yet do not feel the program is ready for production release. However, collaboration on this and related codes I have been exploring is welcome.

Provenance of the R optim–L-BFGS-B and related solvers

The base-R code `lbfgsb.c` (at the time of writing in R-3.5.2/src/appl/) is commented:

```
/* l-bfgs-b.f -- translated by f2c (version 19991025).

  From ?optim:
  The code for method ‘"L-BFGS-B"’ is based on Fortran code by Zhu,
  Byrd, Lu-Chen and Nocedal obtained from Netlib (file 'opt/lbfgs_bcm.shar')

  The Fortran files contained no copyright information.

  Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C.  (1995) A limited
  memory algorithm for bound constrained optimization.
  \emph{SIAM J. Scientific Computing}, \bold{16}, 1190--1208.
*/
```

The paper (Byrd et al., 1995) builds on (Lu et al., 1994). There have been a number of other workers who have followed-up on this work, but R code and packages seem to have largely stayed with codes derived from these original papers. Though the date of the paper is 1995, the ideas it embodies were around for a decade and a half at least, in particular in (Nocedal, 1980) and (Liu and Nocedal, 1989). The definitive Fortran code was published as (Zhu et al., 1997). This is available as `toms/778.zip` on http://www.netlib.org.

Besides the ACM TOMS code, there are two related codes from the Northwestern team on NETLIB:

- http://netlib.org/opt/lbfgs_um.shar is for unconstrained minimization, while
- http://netlib.org/opt/lbfgs_bcm.shar handles bounds constrained problems.

To these are attached references (Liu and Nocedal, 1989) and (Byrd et al., 1995) respectively, most likely reflecting the effort required to implement the constraints.

The unconstrained code has been converted to C under the leadership of Naoaki Okazaki (see http://www.chokkan.org/software/liblbfgs/, or the fork at https://github.com/MIRTK/LBFGS). This has been wrapped for R as the `lbfgs` package (Coppola et al., 2014). I have included this as one of the solvers callable from package `optimx`.

A side-by-side comparison of the main subroutines in the two downloads from Netlib (`toms` and `opt`) unfortunately shows a lot of differences. I have not tried to determine if these affect performance or are simply cosmetic.

More seriously perhaps, there were some deficiencies in the code(s), and in 2011 Nocedal's team published a Fortran code with some corrections (Morales and Nocedal, 2011). Since the R code in C predates this by many years, I prepared package `lbfgsb3` to wrap the Fortran code. However, I did not discover any test cases where the `optim--L-BFGS-B` and `lbfgsb3` gave different output different, though I confess that the tests I ran are not exhaustive.

In 2016, I was at a Fields Institute optimization conference in Toronto for the 70th birthday of Andy Conn. By sheer serendipity, Nocedal did not attend the conference, but sat down next to me at the conference dinner. When I asked him about the key changes, he said that the most important one was to fix the computation of the machine precision, which was not always correct in the 1995 code. Since R gets this number as `.Machine$double.eps`, the offending code is irrelevant.

Within (Morales and Nocedal, 2011), there is also reported an improvement in the subspace minimization that is applied in cases of bounds constraints. In the few tests I have applied with bounds constraints, I have yet to see any substantive differences, but welcome communcation should such be found.

Using Rcpp (see Eddelbuettel and François (2011) and the Fortran code in package `lbfgs3`, Matthew Fidler developed package `lbfgsb3c`. As this provides a more standard call and return than `lbfgsb3`, Fidler and I have unified the two packages, but are still checking and cleaning the package at the time of writing.

Provenance of truncated Newton codes for R

There are (at least) two implementations of truncated Newton methods available.

`nloptr--tnewton()` is a wrapper of the NLopt truncated Newton method translated to C and somewhat modified by Steven G. Johnson in the nlopt project (`https://nlopt.readthedocs.io/en/latest/NLopt_Algorithms/`) from Fortran code due to Ladislav Luksan (`http://www.cs.cas.cz/luksan/subroutines.html`). The many layers and translations make it difficult to unravel the particular details in this code, and I do not feel confident to provide a competent overview.

`optimx--tn()` and `optimx--tnbc()` are translations from Matlab source of the Truncated Newton codes of Stephen Nash (Nash, 1983) that I prepared, initially in package `Rtnmin`. The code is entirely in R. In implementing the codes, the principal awkwardness was in making available to different functions a quite extensive set of variables relating to the function and gradient. My choice was to use the `list2env()` function to create an **environment** to hold this data. Note the survey of truncated Newton methods by Stephen in (Nash, 2000).

Discussion

This story of some of the function minimizers available to R users is not finished. There are certain to be related methods in packages or collections I have overlooked. Moreover, I have not had the time or energy to fully explore some of the items I have mentioned. Nevertheless, how codes come about is important to understanding their strengths and weaknesses and in maintaining them to a level that users can reliably use them. Such matters were a large consideration for Nash (2014b), but have been more explicitly addressed by **histoRicalg**.

When evaluating such tools, it is important to consider what is wanted. Personally, I value reliability as more important than speed. By this, I mean

- the program will always proceed towards a minimum
- on termination it will provide useful information on the result obtained, such as whether there are good indications of a satisfactory result, or that we have halted for some reason such as exhausting a computational limit.

"Speed" is, of course, desirable, but how this is measured is debatable. Execution time is notoriously sensitive to hardware and software environments, as well as to the way functions and gradients are coded. Number of function and gradient counts is an alternative, since these computations often are the bottleneck of optimization. However, there are some methods where the optimization calculations are demanding either in cycles or memory.

Readers may note that I have highlighted that some codes are written entirely in R. This allows for customization or in-line use of faster code, and I have exchanged notes with several workers wanting to speed up time-consuming optimizations by such ideas. Profiling and debugging tools are generally quite challenging to use when there are multiple programming languages involved. All-R code may also be much easier to read and maintain if well-coded, especially as the programming language support for "old" languages diminishes.

These considerations are, of course, why we have a number of methods, and why Ravi Varadhan and I prepared the `optimx` package with tools to allow for comparison of several methods. But please, do use the comparison for evaluating and choosing a method, not for production minimization by a "try everything" strategy. As new methods are released, we hope to include them in the set `optimx` can call via a unified interface that is very close to the `optim()` function. We note that a somewhat different approach to unifying methods is available in the ROI package of Hornik et al. (2011).

Bibliography

E. M. L. Beale. A derivation of conjugate gradients. In F. A. Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 39–43. Academic Press, London, 1972. [p4]

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, Sept. 1995. ISSN 1064-8275. doi: 10.1137/0916069. URL http://dx.doi.org/10.1137/0916069. [p4]

A. Coppola, B. Stewart, and N. Okazaki. *lbfgs: Limited-memory BFGS Optimization*, 2014. URL https://CRAN.R-project.org/package=lbfgs. R package version 1.2.1. [p4]

Y. H. Dai and Y. Yuan. An efficient hybrid conjugate gradient method for unconstrained optimization. *Annals of Operations Research*, 103(1-4):33–47, 2001. [p4]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08. URL http://www.jstatsoft.org/v40/i08/. [p5]

R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970. [p2, 3]

R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7 (2):149–154, 1964. [p4]

P. Fox. The Port Mathematical Subroutine Library, version 3, 1997. URL http://www.bell-labs.com/project/PORT/. [p3]

D. M. Gay. Usage summary for selected optimization routines. Computing Science Technical Report 153. Technical report, AT&T Bell Laboratories, Murray Hill, 1990. [p3]

W. W. Hager and H. Zhang. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Transactions on Mathematical Software*, 32(1):113–137, Mar. 2006a. [p4]

W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization*, 2:35–58, 2006b. [p4]

K. Hornik, D. Meyer, and S. Theussl. *ROI: R Optimization Infrastructure*, 2011. URL http://CRAN.R-project.org/package=ROI. R package version 0.0-7. [p6]

IEEE. IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985*, pages 10–11, 1985. doi: 10.1109/IEEESTD.1985.82928. [p1]

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989. doi: 10.1007/BF01589116. URL https://doi.org/10.1007/BF01589116. [p4]

P. Lu, J. Nocedal, C. Zhu, and R. H. Byrd. A limited-memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16:1190–1208, 1994. [p4]

J. Melville. *mize: Unconstrained Numerical Optimization Algorithms*, 2017. URL https://CRAN.R-project.org/package=mize. R package version 0.1.1. [p3]

J. L. Morales and J. Nocedal. Remark on Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Trans. Math. Softw.*, 38(1):7:1–7:4, Dec 2011. URL http://doi.acm.org/10.1145/2049662.2049669. [p5]

J. C. Nash. *Compact numerical methods for computers : linear algebra and function minimisation*. Adam Hilger, Bristol, 1979. Second Edition, 1990, Bristol: Institute of Physics Publications. [p2]

J. C. Nash. On best practice optimization methods in R. *Journal of Statistical Software*, 60(2):1–14, 2014a. URL http://www.jstatsoft.org/v60/i02/. [p3]

J. C. Nash. *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons: Chichester, May 2014b. ISBN 978-1-118-56928-3. URL http://www.wiley.com//legacy/wileychi/nash/. Companion website (see http://www.wiley.com//legacy/wileychi/nash/). JNfile: 14nlpor.pdf. [p5]

J. C. Nash. *Rcgmin: Conjugate Gradient Minimization of Nonlinear Functions*, 2014c. URL https://CRAN.R-project.org/package=Rcgmin. R package version 2013-2.21. [p4]

J. C. Nash. *Rvmmin: Variable Metric Nonlinear Function Minimization*, 2018. URL https://CRAN.R-project.org/package=Rvmmin. R package version 2018-4.17. [p3]

J. C. Nash and R. Varadhan. Unifying optimization algorithms to aid software system users: optimx for R. *Journal of Statistical Software*, 43(9):1–14, 8 2011. ISSN 1548-7660. URL http://www.jstatsoft.org/v43/i09/. [p3]

J. C. Nash and M. Walker-Smith. *Nonlinear Parameter Estimation: An Integrated System in BASIC*. Marcel Dekker, New York, 1987. See http://www.nashinfo.com/nlpe.htm for an expanded downloadable version. [p3]

S. G. Nash. *Truncated-Newton Methods for Large-Scale Minimization*, pages 91–100. Pergamon, 1983. [p5]

S. G. Nash. A survey of truncated-Newton methods. *Journal of Computational and Applied Mathematics*, 124:45–59, 2000. [p5]

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, January 1965. [p2]

H. B. Nielsen. UCMINF - an algorithm for unconstrained, nonlinear optimization. Technical report, Department of Mathematical Modelling, Technical University of Denmark., dec 2000. URL http://www2.imm.dtu.dk/~hbn/publ/TR0019.ps. Report IMM-REP-2000-18. [p3]

H. B. Nielsen and S. B. Mortensen. *ucminf: General-purpose unconstrained non-linear optimization*, 2012. URL http://CRAN.R-project.org/package=ucminf. R package version 1.1-3. [p3]

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35: 773–782, 7 1980. doi: 10.1090/S0025-5718-1980-0572855-7. [p4]

E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, 16:35–43, 1969. [p4]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL https://www.R-project.org/. [p1]

R. B. Schnabel, J. E. Koonatz, and B. E. Weiss. A modular system of algorithms for unconstrained minimization. *ACM Trans. Math. Softw.*, 11(4):419–440, Dec. 1985. ISSN 0098-3500. doi: 10.1145/6187.6192. URL http://doi.acm.org/10.1145/6187.6192. Original report CU-CS-480-82 from 1982, revised. The UNCMIN Manual. [p3]

H. Sorenson. Comparison of some conjugate direction procedures for function minimization. *Journal of The Franklin Institute - Engineering and Applied Mathematics*, 288:421–441, 12 1969. doi: 10.1016/0016-0032(69)90253-1. [p4]

C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, Dec. 1997. ISSN 0098-3500. doi: 10.1145/279232.279236. URL http://doi.acm.org/10.1145/279232.279236. [p4]

John C. Nash
Retired Professor
University of Ottawa, Telfer School of Management
Ottawa, Ontario
Canada
https://orcid.org/0000-0002-2762-8039
nashjc@uottawa.ca