Variety in the Implementation of Nonlinear Least Squares Program Codes

John C Nash, retired professor, University of Ottawa

16/02/2021

Contents

Abstract

There are many ways to structure a Gauss-Newton style nonlinear least squares program code. In organizing and documenting the nearly half-century of programs in the Nashlib collection associated with John C. Nash (1979), the author realized that this variety could be an instructive subject for software designers.

Underlying algorithms

Hartley Hartley (1961)

Marquardt

(**marq63?**) is perhaps one of the most important developments in nonlinear least squares apart from the Gauss-Newton method. There are several ways to view the method.

First, considering that the $J'J$ may be effectively singular in the computational environment at hand, the Gauss-Newton method may be unable to compute a search step that reduces the sum of squared residuals. The right hand side of the normal equations, that is, $g = -J'r$ is the gradient for the sum of squares. Thus a solution of

$$1_n\delta = g$$

is clearly the gradient. And if we solve

$$\lambda 1_n\delta = g$$

various values of λ will produce steps along the **steepest descents** direction. Solutions of the Levenberg-Marquardt equations

$$(J'J + \lambda 1_n)\delta = -J'r$$

can be thought of as yielding a step δ that merges the Gauss-Newtn and steepest-descents direction. This approach was actually first suggested by Levenberg (1944), but it is my opinion that while the idea is sound, Levenberg very likely never tested them in practice. Before computers were commonly available, this was not unusual, and many papers were written with computational ideas that were not tested or were given only cursory trial.

Others

There have been many proposed approaches to nonlinear least squares. Here are one (or two or ...)

Spiral

Jones (1970) is a method ...

Sources of implementation variety

The sources of variety in implementation include:

- programming language
- possibly operating environment features
- solver for the least squares or linear equations sub-problems
- stucture of storage for the solver, that is, compact or full
- sequential or full creation of the Jacobian and residual, since it may be done in parts
- how the Jacobian is computed or approximated
- higher level presentation of the problem to the computer, as in R's `nls` or packages `minpack.lm` and `nlsr`.

Programming language

We have a versions of the Nashlib Algorithm 23 in BASIC, Fortran, Pascal, R, ...

There may be dialects of these programming languages also.

Operating environment

?? issues of how data is provided

Solver for the least squares or linear equations sub-problems

Solution of the linear normal equations

- Gauss elimination with partial pivoting
- Gauss elimination with complete pivoting
- Variants of Gauss elimination that build matrix decompositions
- Gauss-Jordan inversion
- Choleski Decomposition and back substitution
- Eigendecompositions of the SSCP

Solution of the least squares sub-problem by matrix decomposition

- Householder
- Givens
- pivoting options
- Marquardt and Marquardt Nash options
- SVD approaches

Avoiding duplication of effort when increasing the $\lambda parameter$

Storage stucture

J, J'J, If J'J, then vector form of lower triangle. ??

If the choice of approach to Gauss-Newton or Marquardt is to build the normal equations and hence the sum of squares and cross products (SSCP) matrix, we know by construnction that this is a symmetric matrix and also positive definite. In this case, we can use algorithms that specifically take advantage of both these properties, namely Algorithms 7, 8 and 9 of Nashlib. Algorithms 7 and 8 are the Cholesky decomposition and back-solution using a vector of length `n*(n+1)/2` to store just the lower triangle of the SSCP matrix. Algorithm 9 inverts this matrix *in situ*.

The original John C. Nash (1979) Algorithm 23 (Marquardt nonlinear least squares solution) computes the SSCP matrix $J'J$ and solves the Marquardt-Nash augmented normal equations with the Cholesky approach. This was continued in the Second Edition John C. Nash (1990) and in John C. Nash and Walker-Smith (1987). However, in the now defunct John C. Nash (2012) and successor John C. Nash and Murdoch (2019), the choice has been to use a QR decomposition as described below in ???. The particular QR calculations are in these packages internal to R-base, complicating comparisons of storage, complexity and performance.

Other storage approaches.

Sequential or full Jacobian computation

We could compute a row of the Jacobian plus the corresponding residual element and process this before computing the next row etc.

Analytic or approximate Jacobian

Use of finite difference approximations??

Problem interfacing

R allows the nonlinear least squares problem to be presented via a formula for the model.

Saving storage

Measuring performance

Test problems

Implementation comparisons

Linear least squares and storage considerations

Without going into too many details, we will present the linear least squares problem as

$$Ax \tilde{=} b$$

In this case A is an m by n matrix with $m >= n$ and b a vector of lenght m. We write **residuals** as

$$r = Ax - b$$

or as

$$r_1 = b - Ax$$

Then we wish to minimize the sum of squares $r'r$. This problem does not necessarily have a unique solution, but the **minimal length least squares solution** which is the x that has the smallest $x'x$ that also minimizes $r'r$ is unique.

The historically traditional method for solving the linear least squares problem was to form the **normal equations**

$$A'Ax = A'b$$

This was attractive to early computational workers, since while A is m by n, $A'A$ is only n by n. Unfortunately, this **sum of squares and cross-products** (SSCP) matrix can make the solution less reliable, and this is discussed with examples in John C. Nash (1979) and John C. Nash (1990).

Another approach is to form a QR decomposition of A, for example with Givens rotations.

$$A = QR$$

where Q is orthogonal (by construction for plane rotations) and R is upper triangular. We can rewrite our original form of the least squares problem as

$$Q'A = Q'QR = R \tilde{=} Q'b$$

R is an upper triangular matrix R_n stacked on an $m - n$ by n matrix of zeros. But $z = Q'b$ can be thought of as n-vector z_1 stacked on $(m - n)$-vector z_2. It can easily be shown (we won't do so here) that a least squares solution is the rather easily found (by back-substitution) solution of

$$R_n x = z_1$$

and the minimal sum of squares turns out to be the cross-product $z_2' z_2$. Sometimes the elements of z_2 are called **uncorrelated residuals**. The solution for x can actually be formed in the space used to store z_1 as a further storage saving, since back-substitution forms the elements of x in reverse order.

All this is very nice, but how can we use the ideas to both avoid forming the SSCP matrix and keep our storage requirements low?

Let us think of the row-wise application of the Givens transformations, and use a working array that is $n + 1$ by $n + 1$. (We can actually add more columns if we have more than one b vector.)

Suppose we put the first $n + 1$ rows of a merged $A|b$ working matrix into this storage and apply the row-wise Givens transformations until we have an n by n upper triangular matrix in the first n rows and columns of our working array. We further want row $n + 1$ to have n zeros (which is possible by simple transformations) and a single number in the $n + 1$, $n + 1$ position. This is the first element of z_2. We can write it out to external storage if was want to have it available, or else we can begin to accumulate the sum of squares.

We then put row $n + 2$ of $[A|b]$ into the bottom row of our working storage and eliminate the first n columns of this row with Givens transformations. This gives us another element of z_2. Repeat until all the data has been processed.

We can at this point solve for x. Algorithm 4, however, applies the one-sided Jacobi method to get a singular value decomposition of A allowing of a minimal length least squares solution as well as some useful diagnostic information about the condition of our problem. This was also published as Lefkovitch and Nash (1976).

References

Hartley, H. O. 1961. "The Modified Gauss-Newton Method for Fitting of Nonlinear Regression Functions by Least Squares." *Technometrics* 3: 269–80.

Jones, A. 1970. "Spiral—A new algorithm for non-linear parameter estimation using least squares." *The Computer Journal* 13 (3): 301–8.

Lefkovitch, L. P., and John C. Nash. 1976. "Principal Components and Regression by Singular Value Decomposition on a Small Computer." *Applied Statistics* 25 (3): 210–16.

Levenberg, Kenneth. 1944. "A Method for the Solution of Certain Non-Linear Problems in Least Squares." *Quarterly of Applied Mathematics* 2: 164--168.

Nash, John C. 1979. *Compact Numerical Methods for Computers : Linear Algebra and Function Minimisation*. Book. Hilger: Bristol.

———. 1990. *Compact Numerical Methods for Computers : Linear Algebra and Function Minimisation, Second Edition*. Book. Institute of Physics : Bristol.

———. 2012. *Nlmrt: Functions for Nonlinear Least Squares Solutions*.

Nash, John C., and Mary Walker-Smith. 1987. *Nonlinear Parameter Estimation: An Integrated System in BASIC*. New York: Marcel Dekker.

Nash, John C, and Duncan Murdoch. 2019. *Nlsr: Functions for Nonlinear Least Squares Solutions*.