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 Nash (1979), the author realized that this variety could be an instructive subject for software designers.

Underlying algorithms

Gauss Newton

Hartley

Marquardt (Levenberg)

Spiral (Jones??)

Sources of implementation variety

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 Nash (1979) and 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

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.

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.