GN--a Simple and Effective Nonlinear Least-Squares Algorithm for the Open Source Literature.

Kenneth Klare (kklare@gmail.com) and Guthrie Miller (guthriemiller@gmail.com)

February 21, 2013

Abstract

Although the GN (Gauss-Newton) algorithm was written in the 1980's, we have recently simplified the algorithm and updated the Fortran, and we would like to make it available to others through the open-source literature. The algorithm follows the general guidance given in the 1983 book by Dennis and Schnabel, using an augmented Gauss-Newton, Levenberg–Marquardt approach. We have tested the algorithm against 36 difficult nonlinear minimization problems based on the 1981 article by Moré, Garbow, and Hillstrom. For these test problems the GN algorithm compares favourably to other minimization routines in terms of 1) effectiveness (ability to find the solution), 2) efficiency (number of function evaluations required for convergence), and 3) complexity (number of executable lines of code). The most similar competing open-source algorithm is the LMDIF algorithm from MINPACK by Garbow, Hillstrom, and Moré, against which the more detailed comparisons are made.

Running title: GN--a Simple and Effective Nonlinear Least-Squares Algorithm

Keywords: Nonlinear Least squares, nonlinear minimization, robust, optimization, finite-difference derivatives, augmented Gauss-Newton, Levenberg–Marquard.

1. Introduction

The problem of designing a reliable and efficient minimizer has been around for a long time and while the favorite version has changed through time, there are some basics. Perhaps the most basic, assuming differentiability, is to take steps *s* along the downhill gradient of the sum of squares *f* and can be called the Gauss Method, where

$$s \propto -\nabla f \quad .$$

If the step size is small enough, one can be assured that the step will decrease *f*.

In contrast the Newton Method uses the condition that the gradient is zero at the minimum

$$\nabla f(x+s) = 0 \quad ,$$

where *x* is the current point in parameter space. By first-order Taylor expansion around the current point, this equation becomes a linear equation for the step.

In real problems, although Newton's method can be very fast near the solution—one step in the linear case—it may overstep and cause evaluation errors on more difficult nonlinear problems. The Gauss method usually under-steps and requires too many evaluations.

The algorithm GN (Gauss-Newton) presented here uses the Levenberg–Marquardt[1,2] trust-region compromise and many ideas from the excellent book by Dennis and Schnabel[3]. The algorithm is straightforward, and the Fortran is relatively small sized (323 executable lines).

Matrix methods follow Davidon's approach [4] to updating the Hessian matrix and use a search or a reevaluation dog leg to improve the function and matrix. These methods became the Davidon-Fletcher-Powell (DFP) method and that was superseded by updating the inverse in the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [5-8] method. These methods use a line search along the gradient direction but the trust region method is likely more evaluation efficient and numerically safer.

We assume that evaluations are expensive. The evaluations may be complex or involve processing much data. The goal is to get to the best values reliably with the least number of evaluations. Furthermore, the method is designed to be robust and able to handle most numerical problems.

The functions to be optimized are often not easy to analytically differentiate and are prone to errors in the math, so we always use finite differences to form the derivatives. To reduce the number of evaluations we update the second-derivative matrix until it is not giving results close to expectations or we have reached what believe to be a solution.

When needed, adding a large number to a residual (penalty) may be used to constrain the solution.

The very useful set of test problems from the article by Moré, Garbow, and Hillstrom[9] have been coded in Fortran and are used to validate and tune the algorithm. The GN algorithm was written in the mid-1980s and has been used with good success by the few individuals who were aware of it. The intent in publishing this work at this stage is to make the algorithm and the test problems available to a wider audience through open-access publication. The algorithm is

described in the next section, and the Fortran for the algorithm and the test problems are also available as supplementary material accompanying this article.

2. The GN Algorithm

The algorithm minimizes *f* given by

$$f \equiv \sum_{i=1}^{m} r_i^2$$

in terms of the scaled residuals r_i , from "the data" and fitting functions, for $i = 1...m$.

Each residual is some function of the parameters x_p for $p = 1...n$. In general, the residuals *r* can be represented as a Taylor expansion around some starting point r_0 in parameter space,

$$r_j \cong r_{0j} + \sum_{p=1}^{n} \frac{\partial r(x_0)}{\partial x_p}(x_p - x_{0p}) + \frac{1}{2}\sum_{p,q=1}^{n} \frac{\partial^2 r(x_0)}{\partial x_p \partial x_q}(x_p - x_{0p})(x_q - x_{0q}) + ...$$

In a sufficiently small neighborhood of x_0, *r* can be well approximated as linear, depending linearly on the parameters with coefficients given by the Jacobian matrix

$$J_{jp} = \frac{\partial r_j}{\partial x_p}(x_0) \quad .$$

The fundamental quantity *f* is then given by

$$f = \sum_{j=1}^{m}\left(r_{0j} + \sum_{p=1}^{n} J_{jp}(x_p - x_{p0}) \right)^2 \quad ,$$

neglecting the second derivatives of the residuals, which is justified in a sufficiently small neighborhood of x_0. To simplify the notation,

$$x_p - x_{p0} \rightarrow s_p \quad ,$$

in terms of an iterative "step" *s*.

Also, to simplify the algebraic manipulations, matrix notation will be used, where a capital letter denotes a matrix, for example

$$A \leftrightarrow \begin{bmatrix} A_{11}, A_{12}, ... \\ A_{21}, A_{22}, ... \\ ... \end{bmatrix}$$

In matrix notation

$$A^T{}_{i,j} \equiv A_{j,i}$$
$$(AB)^T = B^T A^T$$

and for a square matrix,

$$(A^{-1})^T = (A^T)^{-1}$$

where A^T denotes the transpose of matrix A, *i.e.*, rows and columns exchanged. This makes the sum of squares f,

$$f = (r + Js)^T (r + Js) = r^T r + s^T J^T r + r^T Js + s^T J^T Js \quad , \tag{1}$$

where s and r are column vectors containing the elements s_p and r_i. In Eq. (1) note that the matrix J is m (the number of data) rows high by n (the number of parameters) columns wide, s is a single column with n rows, and Js and r are single columns with m rows. See Appendix B for an example. In matrix multiplication, for example AB, the number of columns of A must be the same as the number of rows of B. The product has the number of rows of A and the number of columns of B.

$$(AB)_{i,j} \equiv \sum_k A_{i,k} B_{k,j} \quad .$$

Our approach to Eq. (1) is to complete the square—a simple but powerful algebraic technique. For example the quadratic form $ax^2 + bx + c$ can be rewritten as $a(x + b/(2a))^2 + c - b^2/(4a)$, which shows immediately the minimum obtained for $x = -b/(2a)$. Similarly, one can verify that Eq. (1) can be rewritten as

$$f = \left(s + (J^T J)^{-1} J^T r\right)^T J^T J \left(s + (J^T J)^{-1} J^T r\right) + r^T \left(I - J(J^T J)^{-1} J^T\right) r \quad , \tag{2}$$

where I is the $m \times m$ identity matrix with all 1's on the diagonal. For the moment, we simply assume that the $n \times n$ "Hessian" matrix $H = J^T J$ is nonsingular and can be inverted. From Eq. (2), one sees immediately the step required to minimize f is given by

$$s = -H^{-1} J^T r \quad .$$

For a linear problem, the minimum is immediately achieved with this step.

For a nonlinear problem, this step (Newton step) may be too ambitious. You may not decrease f when you begin climbing the opposite side. Notice from Eq. (1) that the derivative of f near the origin is given by

$$\frac{\partial f}{\partial x_p} = 2\left(J^T r\right)_p \quad .$$

Therefore a step of the form

$$s = -\frac{1}{\mu} J^T r \equiv -\frac{g}{\mu}$$

with μ sufficiently large will be a step down the gradient (steepest descent) and will always decrease *f*, where we have introduced the notation $g \equiv J^T r$. The disadvantage of this conservative (Gaussian) step is that it may take a very large number of iterations to reach the minimum, while the Newton step goes immediately to the minimum for a linear problem and for a well-behaved nonlinear problem gives quadratic convergence near the minimum (the error eventually decreasing like the square of some factor at each step.) The Levenberg-Marquardt compromise solution[1,2] is a step of the form

$$s = -(H + \mu D^2)^{-1} g \quad , \tag{3}$$

which becomes the Newton step for small μ and is a Gaussian step for large μ, where D^2 is a positive diagonal scaling matrix. This technique also takes care of possible singularity of *H*, because the matrix $H + \mu D^2$ will always be invertible when the diagonal is augmented by some positive μ. The example of Appendix B may be helpful.

To understand the Levenberg-Marquardt method, note that Eq. (3), for some $\mu > 0$ is the solution of the constrained minimization of the quadratic form given by Eq. (2) subject to the constraint on step size

$$\sqrt{(Ds)^T Ds} \le \delta \quad ,$$

which can be demonstrated by introducing μ as a Lagrange multiplier in the constrained minimization of Eq. (2).

Therefore, the solution of Eq. (3) represents the minimum of Eq. (2) within the "trust radius" δ.

The notation connected with scaling is simplified by the transformation of variables

$$\begin{aligned} &\frac{H_{j,j'}}{D_j D_{j'}} \to H_{j,j'} \\ &D_j s_j \to s_j \quad , \\ &\frac{g_j}{D_j} \to g_j \end{aligned}$$

under which Eq. (3) becomes $s = -(H + \mu I)^{-1} g$, where *I* is the identity matrix. Transformed variables will be used subsequently.

Scaling is important when there is a vast difference in the natural scale of different parameters. Many times this is not the case, for example, when the parameters represent the logs of physical parameters. The log transformation is a good way to normalize the scale and to impose positivity on the physical parameter. The calculations given in this paper do not use scaling (`iscale` = 0), even though some of the problems have large differences in final parameter values. For these problems we find that scaling is necessary only when using single precision.

The GN algorithm proceeds as follows.

In one mode (`NewtStepFirst = .true.`), first try a full Newton step, with μ just the minimum required to prevent singularity (minimum augmentation). If the problem is linear, the minimum will have been obtained in this one step.

If the function increases, go back and start over using a smaller step (the "Cauchy step"), but still in the Newton direction. The Cauchy step is the step in the direction g that minimizes the quadratic given by Eq. (2). One can show that the length of this step is given by

$$s_C = \frac{\left(g^T g\right)^{3/2}}{g^T H g} \quad .$$

It provides a natural step length, without requiring matrix inversion, as fallback when the Newton step does not decrease f.

The more conservative mode (`NewtStepFirst = .false.`), which is used in the calculations given in this paper, does not first try a Newton step but just begins with the Cauchy step. The large Newton step can sometimes cause numerical problems where the function cannot be evaluated.

After the first step, one has available an initial f_0, a final f after taking the step, and the derivatives at the initial point. The trust radius is then revised using a quadratic fit based on these values as described in Appendix C.

For example, consider the Rosenblock test problem (problem 1), which has $m = n = 2$. The residuals are defined by

$$r = \begin{pmatrix} 10\left(x_2 - x_1^2\right) \\ 1 - x_1 \end{pmatrix} \quad . \tag{4}$$

Figure 1 shows the steps to the minimum for two different starting points.

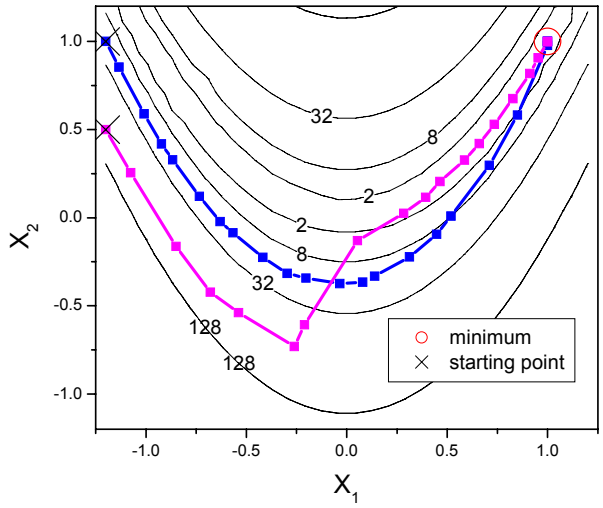

Figure 1—steps taken to reach the minimum superimposed on a contour plot of the function f for test problem 1. The number of function evaluations required was 39 or 36 depending on the starting points.

To provide contrast, Fig. 2 shows the steps to the minimum if there is only one residual defined as the square root of the sum of squares given by Eq. (4).

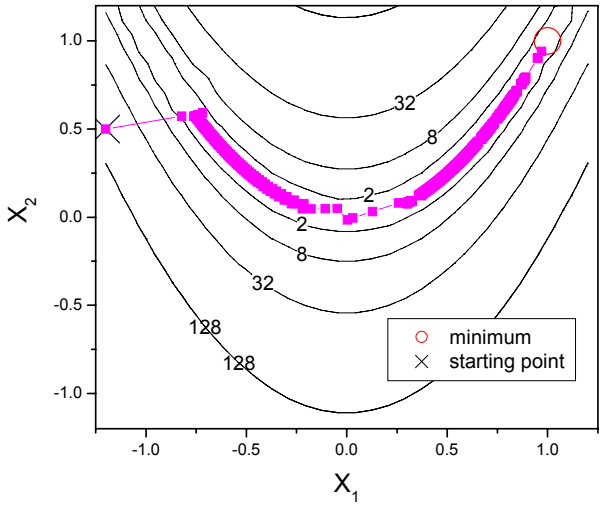

Figure 2—steps taken to get close to the minimum superimposed on a contour plot of the function f for test problem 1. In this case there is only one piece of data (m = 1) equal to the square root of the sums of squares for problem 1. The number of function evaluations required was much larger than in Fig. 1.

As shown in Fig. 2, if only the function *f* is available, the algorithm steps down the gradient of *f* going quickly down the steep sides to the bottom of the valley, which then is very gently sloping, and then follows the "stream bed" towards the minimum. In contrast, the Newton step

is mostly at right angles to the steepest descent direction, and stays high until reaching the region of quadratic convergence at the end.

3. The Test Problems

The 36 test problems used come from a suite published in 1981 [9]. This collection has systems of nonlinear equations, nonlinear least squares, and unconstrained minimization. The GN program can treat all of these, overdetermined (m>n) and underdetermined (m<n), in the same manner as nonlinear equations (m=n). Very large m or n can lead to ill-conditioned problems. Testing serves several purposes: it is a general check on the coding, it checks the efficiency of the algorithm, and it is a source of confidence that the algorithm will give a correct answer on new problems. Even a minor change in the coding needs to be tested.

In some cases, the Hessian without augmentation of the diagonal is singular. One cannot expect to correctly solve the Hessian matrix's inherent inversion when the ratio of eigenvalues before augmentation exceeds the machine precision or the accuracy of the data and equations. The inverse Hessian is the variance-covariance matrix in a linear uncertainty analysis. However, except for linear problems, we find this linear analysis to not be a reliable indication of parameter uncertainties, even if the all second-order derivative terms (found numerically using GN itself) are included in the Hessian. We find it necessary to do a Monte Carlo analysis where some large number (say 100) of alternate realizations of the data are generated and the nonlinear fits repeated (the residual is imagined to represent fit-data divided by the standard deviation of the data.)

The GN method uses a fair amount of storage for large problems and may have numeric problems when summing large arrays. Normally double precision is sufficient to handle the latter problem. Quad precision has been tested but produces almost identical results for these problems.

One can use either parameter change (`stptol`) or function decrease (`abstol` and `reltol`) as a stopping criterion. Use `abstol` and `reltol` to obtain a specific accuracy in the function values or `stptol` to get the positions or other parameters to the accuracy needed. Use `stptol` = 1e-4 for about 3-digit accuracy. Use of `stptol` is appropriate for say placing objects, defining field strengths, etc. Use `abstol` and `reltol` to set what the sum-of-squares of residuals accuracy is required. Each is limited to the attainable computational accuracy. GN will stop on the first criterion met. To stop on `abstol` the function must be less than `abstol`. To stop on `reltol` the change of the function must be less than the maximum of `reltol` times the function and `abstol`.

Table I is a sample run using a Windows XP system, Intel Q9650 processor, and MinGW G95 Fortran (www.mingw.org) compiler with absolute function accuracy `abstol`= 1×10^{-11}, relative function accuracy `reltol`= 1×10^{-7}, position accuracy `xacc` = 1×10^{-4}, and derivative step size = 1×10^{-4}. Listed are the problem number, number of residuals (m), number of parameters (n), iteration count, number of function evaluations, the final value obtained for the sum of squares, and the function's name from Moré et al. An iteration involves calculating the function and possibly all derivatives.

The final parameter values were checked against the correct values. Because of the possibility of multiple solutions, the starting points were changed from the Moré values in several cases in order to avoid instability toward secondary solutions. Increasing `abstol` and `reltol` results

in fewer evaluations. It was found that `abstol` and `reltol` could not be further increased over the values stated in the previous paragraph and still preserve an error tolerance of 0.005 for all parameters and all test problems. Decreasing these parameters, in particular `reltol`, increased the number of evaluations, in some cases markedly–not for these test problems but for their modified versions used for the Monte Carlo uncertainty calculations. This dramatic increase in the number of evaluations for small `reltol` was not eliminated by going to quad precision, so it is not just a matter of numerical noise, which could be prevented by imposing a lower limit on `reltol` related to the machine precision. The conclusion from all this is that one must choose the tolerances, in particular `reltol`, with some care.

Table I—Performance of the GN algorithm for the Moré test problems. The total number of function evaluations is 2116 for the 36 problems.

prob	m	n	iter	evals	value	Moré	function name
1	2	2	22	36	0	0	Rosenbrock
2	2	2	39	61	48.98425	48.9842	Freudenstein & Roth
3	2	2	25	41	8.54E-17	0	Powell badly scaled
4	3	2	27	45	1.41E-22	0	Brown badly scaled
5	3	2	14	24	3.33E-16	0	Beale
6	10	2	24	38	124.3622	124.362	Jennrich & Sampson
7	3	3	16	31	7.48E-20	0	Helical valley
8	15	3	9	18	8.21E-03	8.21E-03	Bard
9	15	3	12	24	1.13E-08	1.13E-08	Gaussian
10	16	3	28	52	87.94586	87.9458	Meyer
11	99	3	34	58	2.20E-16	0	Gulf R & D
12	9	3	8	14	1.47E-12	0	Box 3D
13	4	4	18	34	4.57E-12	0	Powell singular
14	6	4	82	144	1.81E-16	0	Wood
15	11	4	20	44	3.08E-04	3.08E-04	Kowalik & Osborne
16	20	4	60	116	85822.24	85822.2	Brown & Dennis
17	33	5	19	39	5.46E-05	5.46E-05	Osborne 1
18	13	6	43	85	1.17E-15	5.66E-30	Biggs EXP6
19	65	11	25	69	4.01E-02	4.01E-02	Osborne 2
20	31	9	22	49	1.40E-06	1.40E-06	Watson
21	12	12	31	67	3.71E-28	0	Extended Rosenbrock
22	12	12	17	41	4.82E-12	0	Extended Powell
23	5	4	30	54	2.25E-05	2.25E-05	Penalty I
24	8	4	32	60	9.38E-06	9.38E-06	Penalty II
25	11	9	24	51	5.62E-15	0	Variably dimensioned
26	9	9	23	50	4.41E-13	0	Trigonometric
27	9	9	3	12	1.23E-27	0	Brown almost linear
28	9	9	4	13	3.39E-15	0	Discrete boundary value
29	9	9	4	13	1.15E-13	0	Discrete integral equ.
30	9	9	9	27	1.30E-18	0	Broyden tridiagonal
31	9	9	13	31	2.41E-12	0	Broyden banded
32	12	9	5	23	3	m-n = 3	Linear full rank
33	12	9	5	23	2.64	m(m-1)/ (4m+2) = 2.64	Linear rank 1
34	12	9	5	23	4.142857	$(m^2+3m-6)/ (4m-6) = 4.14286$	Linear rank 1 with zeros
35	9	12	6	30	2.10E-16	NA	Chebyquad
36	3	2	378	574	1.232	NA	Modified Beale (prob 5)

Problem 35 is Moré's problem changed into an under-constrained example with more parameters (12) than data (9). Problem 36 is altered version of the Beale problem (problem 5) that required many consecutive steps, each causing a very small decrease in the sum of squares.

The GN algorithm has an addition based on this phenomenon, where the trust radius is increased by a factor based on the number of consecutive steps with function decreases.

Table II—Performance of the GN algorithm for the Moré test problems in terms of total number of function evaluations on two platforms: Windows XP with Intel Q 9650 processor using MinGW G95 Fortran (www.mingw.org) and MacBook Intel 64-bit OS 10.6.8 using GNU Fortran 4.6.3 (gcc.gnu.org). Three different parameter starting values are used: near(0.001) the answer, the Moré values, and the Moré values slightly perturbed (0.04). All final parameter values were within a tolerance of 0.005 from the correct values.

Compiler	start	#evals
G95	near answer	746
G95	Moré	2114
G95	perturbed Moré	2579
GNU	near answer	746
GNU	Moré	2118
GNU	perturbed Moré	2596

4. Comparison of GN to other minimization algorithms

There are several types of minimization algorithms. Ranked in terms of generality these are:

1. Those requiring only a function to be minimized that may not be differentiable.
2. Those requiring a differentiable function to be minimized.
3. Those minimizing the sum of squares of differentiable functions.

When the problem permits, we expect that type-3 algorithms (like GN) would work best in terms of effectiveness (finding the solution) and efficiency (using a small number of function evaluations). For an open-source algorithm that is meant to be used, understood, and possibly modified by others, small program size is also important. Comparisons with GN are summarized in Table III.

Table III—Comparisons of numbers of function evaluations to find the minimum and executable lines of code for three algorithms of different types versus GN.

type	algorithm	#lines	#probs	#evals	#evalsGN	ratio
1	SUBPLEX	739	28	4480845	4637	966.3
2	MINUIT	8000	28	54890	4571	12.0
3	LMDIF	555	30	13896	4485	3.1
3	GN	323	37	5439	5439	1

These comparisons used the G95 compiler on a Windows XP system.

The type-1 algorithm chosen was the SUBPLEX algorithm, by Tom Rowan, with 739 executable lines of Fortran, which uses the Simplex method (DMOZ website, see references). SUBFLEX was easy to use and required only the tolerance for ending, which we took to be our `reltol` $= 3 \times 10^{-7}$, and the default choice (`scale(1)=-1`) for parameter scaling. When minimizing the sum of squares from our 36 test problems for the 3 starting points, SUBPLEX was out of tolerance (f tolerance = 5×10^{-5}, x tolerance = 0.01) from the expected answer for problems (starting points: 1 = near answer, 2 = Moré, 3 = perturbed Moré): 3 (2), 17(1,3), 20(2,3), 24(2), 26(2,3), 31(2,3), and 35(3). The SUBPLEX solutions, when used as starting points for GN, were within tolerance of the GN final solution for 5/11 of these cases and were improved by GN for 7/11 cases at the expense of 365 further function evaluations. For the 28 test problems that returned the expected solutions within tolerance for all 3 starting points, the number of function evaluations required was 4,480,845 compared to 4637 using GN.

The type-2 algorithm used in the comparison is the steepest descent algorithm MIGRAD from the MINUIT package developed at CERN by Fred James (MINUIT website, see references). MINUIT is a large complex program with almost 8000 executable lines of code. Within MINUIT we chose the steepest descent algorithm MIGRAD ("This is the best minimizer for nearly all functions" according to the MINUIT documentation) where the function to be minimized was the sum of squares for each of our 36 test problems. Setting the tolerance again to be our `reltol` $= 3 \times 10^{-7}$, we found that MINUIT was out of tolerance from the expected answer for problems (starting points): 3 (1,2,3), 5(3), 17(1), 18(2,3), 24(2,3), 26(2,3), 31(2,3), and 35(2). The MINUIT solutions, when used as starting points for GN, were within tolerance of the GN final solutions for 6/14 of these cases and were improved by GN for 8/14 cases at the expense of 754 further function evaluations. For the 28 test problems that returned the expected solutions within tolerance for all 3 starting points, the number of function evaluations required was 54,890 compared to 4571 using GN.

The most similar algorithm to GN that we could find in the open-source literature was LMDIF, which is part of MINPACK (www.netlib.org) written by Garbow, Hillstrom, and Moré. Using LMDIF, which has 555 executable lines of Fortran, on the same 36 test problems was very easy and only one parameter needed to be specified, the tolerance for ending, which we took to be our `reltol` $= 3 \times 10^{-7}$. We found LMDIF could not be used on problem 35, where the number of parameters exceeded the number of data, as this produced an input error. LMDIF was out of tolerance from the expected answer for problems (starting points): 5(3), 14(2,3), 16(2,3), 18(2,3), and 26(2,3). The LMDIF solutions, when used as starting points for GN, were within tolerance of the GN final solutions for 6/9 of these cases and were improved by GN for 3/9 cases at the expense of 55 further function evaluations. For the 30 test problems that returned the expected solutions within tolerance for all 3 starting points, the number of function evaluations required was 13,896 compared to 4,485 using GN. The problem-by-problem comparison for just the normal starting point is shown in Table IV. Note that LMDIF has fewer evaluations than GN only on problem 2 and that GN requires less than 70 evaluations for all problems except the last.

Table IV—Number of function evaluations required for LMDIF versus GN, with the normal starting point for the subset of 30 problems where LMDIF gave the expected result.

Problem	LMDIF	GN
1	54	36
2	29	61
3	50	41

4	46	45
6	42	38
7	35	31
8	21	18
9	34	24
10	474	52
11	77	58
12	25	14
13	1001	34
15	67	44
17	93	39
19	148	69
20	51	49
21	214	67
22	2601	41
23	119	54
24	591	60
25	101	51
27	21	12
28	41	13
29	41	13
30	51	27
31	83	31
32	21	23
33	22	23
34	21	23
36	600	574
totals	6774	1665

5. Discussion and Conclusion

The GN algorithm is less general than other minimization algorithms in that it minimizes the sum of squares of differentiable functions rather than a general function. However for problems of this type, and in particular for our 36 test problems, in comparison to other open source algorithms we find that it is superior in terms of 1) effectiveness (ability to find the solution), 2) efficiency (number of function evaluations required for convergence), and 3) complexity (number of executable lines of code).

This method was developed as part of the experimental magnetic-fusion-energy program at Los Alamos and has been in use for 28 years with good success. We hope it will now be of use to a wider audience. Note that it can be efficiently used for linear equations; it takes only one additional function evaluation to confirm the solution.

Some “tricks” and insights discovered include:

1. Computing the machine numerical precision by compiler function.
2. By augmenting the diagonal of the Hessian with a small number related to machine precision we are assured a solution for the Newton step, even when the Hessian itself is singular.

3. Rank-1 updates of the Jacobian, without resorting to a full Jacobian recalculation.
4. Limiting the range of trust radius change.
5. Imposing a minimum on the Cauchy step--important when starting near the solution.
6. Increasing the trust radius based on having a long run of function decreases.
7. Automatic updating of the scaling vector by averaging with the current parameter absolute value.
8. The importance of tuning the GN algorithm parameters based on the performance for test problems.

Acknowledgement

This work was performed with partial support from funding from the National Institute of Allergy and Infectious Diseases under contract No. HHSN272201000046C.

References

1. Levenberg, K. (1944), "A Method for the Solution of Certain Problems in Least Squares", *Quart. Appl. Math.* **2**, 164-168 (1944).
2. Marquardt, D. (1963), "An algorithm for Least-Squares Estimation of Nonlinear Parameters", *SIAM J. Appl. Math.* **11**, 431-441 (1963).
3. Dennis, J.E. and Schnabel, R.B. (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, (1983) SIAM, 378p. in 1996 ed.
4. Davidon, W. C (1959), "Variable metric method for minimization", SIAM Journal on Optimization 1: 1–17, doi:10.1137/0801001, a reprint of Davidon, W.C. (1959), "Variable metric method for minimization", Argonne National Laboratory, Report ANL-5990 (Rev.), Argonne, Illinois.
5. Broyden, C. G. (1970), "The convergence of a class of double-rank minimization algorithms", *Journal of the Institute of Mathematics and Its Applications* **6**: 76–90, doi:10.1093/imamat/6.1.76
6. Fletcher, Roger (1987), *Practical methods of optimization* (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8
7. Goldfarb, D. (1970), "A Family of Variable Metric Updates Derived by Variational Means", *Mathematics of Computation* **24** (109): 23–26, doi:10.1090/S0025-5718-1970-0258249-6
8. Shanno, David F.; Kettler, Paul C. (July 1970), "Optimal conditioning of quasi-Newton methods", *Math. Comput.* **24** (111): 657–664, doi:10.1090/S0025-5718-1970-0274030-6, MR42:8906
9. Moré, J.J., Garbow, B.S., and Hillstrom, K.E., "Testing Unconstrained Optimization Software", *ACM transactions on Mathematical Software*, **7,** 1 (1981).
10. DMOZ website, www.dmoz.org/Computers/Programming/Languages/Fortran/Source_Code/Optimization
11. MINUIT obtained from https://github.com/ramos/minuit/downloads).

Appendix A—Solution of $\Phi(\mu) = 0$

Let $\Phi(\mu)$ be the magnitude of the "hook" step $x(\mu) = -(H + \mu I)^{-1} g$ minus some specified length δ. In other words, $\Phi(\mu) = 0$ expresses the condition that the magnitude of the step is the trust radius δ. In order to solve for the step x with a given radius δ, we need to first solve the equation

$$\Phi(\mu) = 0 \quad ,$$

for μ, and then obtain the step from $x(\mu) = -(H + \mu I)^{-1} g$.

How do we solve $\Phi(\mu) = 0$? Motivated by the one-dimensional case and following Dennis and Schnabel, a local model is used for $\Phi(\mu)$,

$$\Phi(\mu) \cong \frac{\alpha}{\beta + \mu} - \delta \quad , \tag{A.1}$$

for some scalar parameters α and β. Assume a current value μ_c of μ. The current values of α_c and β_c are obtained from the two equations

$$\begin{aligned} \Phi(\mu_c) &= \frac{\alpha_c}{\beta_c + \mu_c} - \delta_c \\ \Phi'(\mu_c) &= -\frac{\alpha_c}{(\beta_c + \mu_c)^2} \end{aligned} \quad ,$$

which when solved give

$$\begin{aligned} \alpha_c &= -\frac{(\Phi(\mu_c) + \delta_c)^2}{\Phi'(\mu_c)} \\ \beta_c &= -\mu_c - \frac{(\Phi(\mu_c) + \delta_c)}{\Phi'(\mu_c)} \end{aligned} \quad . \tag{A.2}$$

The iteration of μ ("hook search") based on the local model is

$$\mu_+ = \frac{\alpha_c}{\delta_c} - \beta_c \quad ,$$

obtained by solving Eq. (A.1) for μ. By substituting from Eq. (A.2), this gives

$$\mu_+ = \mu_c - \frac{\Phi(\mu_c) + \delta_c}{\delta_c} \frac{\Phi(\mu_c)}{\Phi'(\mu_c)} \quad . \tag{A.3}$$

The derivative with respect to μ of the column matrix $x(\mu)$ defined by $x(\mu) = (H + \mu I)^{-1} g$ can be obtained by taking the derivative of the equation $(H + \mu I)x(\mu) = g$ with respect to μ and then solving for $x(\mu)$. One obtains

$$\frac{dx}{d\mu} \equiv x'(\mu) = -\left(H + \mu I\right)^{-1} x(\mu) \quad .$$

Therefore, from

$$\Phi(\mu) \equiv \sqrt{x(\mu)^T x(\mu)} - \delta \,, \tag{A.4}$$

one finds that

$$\Phi'(\mu) = \frac{x(\mu)^T x'(\mu)}{\sqrt{x(\mu)^T x(\mu)}} = -\frac{x(\mu)^T (H + \mu I)^{-1} x(\mu)}{\sqrt{x(\mu)^T x(\mu)}} \quad . \tag{A.5}$$

Equations (A.3), (A.4), and (A.5) allow one to iteratively solve for μ, given δ.

Appendix B—Example for the case m=1, n=2

In this case there are two parameters and one residual *r*. The basic equations are:

$$f = r^2$$

$$J = \left(\frac{\partial r}{\partial x} \quad \frac{\partial r}{\partial y} \right)$$

$$H = J^T J = \begin{pmatrix} \left(\frac{\partial r}{\partial x}\right)^2 & \frac{\partial r}{\partial x}\frac{\partial r}{\partial y} \\ \frac{\partial r}{\partial x}\frac{\partial r}{\partial y} & \left(\frac{\partial r}{\partial y}\right)^2 \end{pmatrix} \quad .$$

$$g = J^T r = \begin{pmatrix} r\frac{\partial r}{\partial x} \\ r\frac{\partial r}{\partial y} \end{pmatrix}$$

The equation for step *s* is

$$\left(H + \mu I\right)s = -g \quad ,$$

or

$$\begin{pmatrix} \left(\frac{\partial r}{\partial x}\right)^2 + \mu & \frac{\partial r}{\partial x}\frac{\partial r}{\partial y} \\ \frac{\partial r}{\partial x}\frac{\partial r}{\partial y} & \left(\frac{\partial r}{\partial y}\right)^2 + \mu \end{pmatrix} s = -g \quad .$$

The solution is

$$s = -\frac{\begin{pmatrix} \left(\frac{\partial r}{\partial y}\right)^2 + \mu & -\frac{\partial r}{\partial x}\frac{\partial r}{\partial y} \\ -\frac{\partial r}{\partial x}\frac{\partial r}{\partial y} & \left(\frac{\partial r}{\partial x}\right)^2 + \mu \end{pmatrix}}{\mu\left(\left(\frac{\partial r}{\partial x}\right)^2 + \left(\frac{\partial r}{\partial y}\right)^2\right) + \mu^2} \mathrm{g} \quad .$$

Substituting for g, we get

$$s = -\frac{\begin{pmatrix} \frac{\partial r}{\partial x} \\ \frac{\partial r}{\partial y} \end{pmatrix}}{\left(\left(\frac{\partial r}{\partial x}\right)^2 + \left(\frac{\partial r}{\partial y}\right)^2\right) + \mu} r \quad .$$

Thus there is a well-defined step directed down the gradient in the limit $\mu \to 0$, even though the matrix *H* is singular.

One can verify that the same step is obtained if *m* is increased to any number by adding more residuals having *r* = 0.

Appendix C—Updating the trust radius

Consider movement in the direction *s*, $x = \lambda s / \| s \|$ and the value of $f(\lambda)$ as function of the λ. From the definition of *f*, for *x* small

$$f'(\lambda) = \frac{df}{d\lambda} = 2g^T \frac{dx}{d\lambda}$$
$$f'(0) = 2\frac{g^T s}{\| s \|} \quad .$$

The distance λ starts out at 0, where *f* takes the value f_0, and ends with the value $\lambda = \| s \| = \sqrt{s^T s}$ at the final point of the step, where $f(\lambda)$ takes value *f*. By fitting a quadratic $f(\lambda) \cong f_0 + a\lambda + b\lambda^2$, the minimum occurs at

$$\lambda_* = -\frac{a}{2b} \quad ,$$

where

$$a = f'(0)$$
$$b = \frac{f - f_0 - a \| s \|}{\| s \|^2} \quad ,$$

so that

$$\lambda_* = -\frac{a}{2b} = \frac{1}{2}\frac{\| s \|}{1 - r} \tag{C.1}$$

with

$$r = \frac{f_0 - f}{-f'(0) \| s \|} = \frac{f_0 - f}{-2 g^T s} \quad .$$

The quantity r is bounded above by 1 (when the curvature is negligible) and unbounded below (when $f > f_0$). With the revised trust radius given by Eq. (C.1), which can be greater than the last step if $r > 1/2$, a new value of μ and a new step are calculated as described in Appendix A.