NBER WORKING PAPER SERIES

AN ADAPTIVE NONLINEAR LEAST-SQUARES ALGORITHM

John E. Dennis, Jr.*, David M. Gay**,
and Roy E. Welsch†

Working Paper No. 196

COMPUTER RESEARCH CENTER FOR ECONOMICS AND MANAGEMENT SCIENCE
National Bureau of Economic Research, Inc.
575 Technology Square
Cambridge, Massachusetts 02139

August 1977

Preliminary

*Cornell University. Research supported in part by NSF Grant MCS76-00324.

**NBER Computer Research Center. Research supported in part by NSF Grant MCS76-00324.

†Massachusetts Institute of Technology. Research supported in part by NSF Grants SOC76-14311 and MCS76-00324.

ABSTRACT

NL2SOL is a modular program for solving the nonlinear least-squares problem that incorporates a number of novel features. It maintains a secant approximation S to the second-order part of the least-squares Hessian and adaptively decides when to use this approximation. We have found it very helpful to "size" S before updating it, something which looks much akin to Oren-Luenberger scaling. Rather than resorting to line searches or Levenberg-Marquardt modifications, we use the double-dogleg scheme of Dennis and Mei together with a special module for assessing the quality of the step thus computed. We discuss these and other ideas behind NL2SOL and briefly describe its evolution and current implementation.

CONTENTS

AN ADAPTIVE NONLINEAR LEAST-SQUARES ALGORITHM

by

J.E. Dennis, Jr., David M. Gay, Roy E. Welsch

1. Introduction

This project began in order to meet a need at the Computer Research Center of the National Bureau of Economic Research for a nonlinear least-squares algorithm which, in the large residual case, would be more reliable than the Gauss-Newton or Levenberg-Marquardt method [Dennis, 1977] and more efficient than the secant or variable metric algorithms [Dennis & Moré, 1977] such as the Davidon-Fletcher-Powell, which are intended for general function minimization.

We have developed a satisfactory computer program called NL2SOL based on ideas in [Dennis & Welsch, 1976] and our primary purpose here is to report the details and to give some test results. On the other hand, we learned so much during the development which seems likely to be applicable in the development of other algorithms that we have chosen to expand our exposition to include this experience.

In section 2 we will set out the problem and the notation we intend to use. Section 3 will deal with our way of supplementing the classical Gauss-Newton approximation to the least-squares Hessian by various analogs of the Davidon-Fletcher-Powell method. Section 4 will briefly describe our interpretation of the Oren-Luenburger [Oren, 1973] sizing strategy for this augmentation. In section 5 we describe our adaptive quadratic modeling of the objective function. Section 6 contains a discussion of the linear algebra involved in extracting information from the quadratic models and section 7 contains test results. The NL2SOL Usage Summary is included as an appendix.

2. The Nonlinear Least-Squares Problem

There are good reasons for numerical analysts to study this problem. In the first place, it is a computation of primary importance in statistical data analysis and hence in the social sciences, as well as in the more traditional areas within the physical sciences. Thus a computer algorithm able to deal efficiently with both sorts of data is widely applicable.

Although applicability should always constitute sufficient justification to tackle a problem, in this case there is also an opportunity for more far reaching progress in numerical optimization. In order to be more specific, it will be useful to have a formal statement of the nonlinear least-squares problem.

We adopt notation consistent with fitting a model to n pieces of data using p parameters: Given $R : \mathbb{R}^p \to \mathbb{R}^n$, we wish to solve the unconstrained minimization problem

$$\min f(x) = \tfrac{1}{2} R(x)^T R(x) = \tfrac{1}{2} \sum_{i=1}^{n} r_i(x)^2 \quad . \tag{2.1}$$

Notice that for $J(x) = R'(x) = (\partial_j r_i(x))$, the gradient of f is:

$$\nabla f(x) = J(x)^T R(x) \tag{2.2}$$

and the Hessian of f is:

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{i=1}^{n} r_i(x) \nabla^2 r_i(x) \quad . \tag{2.3}$$

Since we are seeking a minimum of f, we would wish to have

$f(x^*) = 0$, an obviously global minimum; in the more realistic case when f is not anywhere near zero, we will be forced to terminate on small parameter changes or to use some version of a gradient test. These stopping conditions are controlled by FCONCR, GCONCR, GRDMIN, and XCONCR (see the NL2SOL Usage Summary). For nonzero f, the test involving GCONCR, which is explained in more detail in [Dennis, 1977], is a scale-free test on the maximum of the absolute cosines of the angles between $R(x_k)$ and the p columns of $J(x_k)$. This is useful because it is clear from (2.2) that $\nabla f(x^*) = 0$ and $R(x^*) \neq 0$ corresponds to $R(x^*) \perp C(J(x^*))$, the column space of $J(x^*)$. Thus it is essential as the iteration proceeds that $J(x_k)$ be approximated very well in the usual case when $p < n$ and $R(x^*) \neq 0$.

In addition to making a precise convergence test possible, having an accurate Jacobian matrix means that a good approximation to a portion of the Hessian is available as a byproduct of the gradient computation. In fact, it is often possible to ignore the second order term $\sum r_i(x)\nabla^2 r_i(x)$ of the Hessian altogether on the grounds that if the nonzero residuals are not of a sort that reinforce their nonlinearity, $J(x)^T J(x)$ is a sufficiently good Hessian approximation [Wedin, 1972, 1974a-c], [Dennis, 1977]. In the resulting Gauss-Newton method, the "Newton step" for x_k is defined by the linear system of equations

$$J(x_k)^T J(x_k) s_k = -J(x_k)^T R(x_k). \tag{2.4}$$

Since (2.4) is the system of normal equations for the linear least-squares problem

$$(2.5) \qquad \min \tfrac{1}{2}(J(x_k)s + R(x_k))^T(J(x_k)s + R(x_k)) \ ,$$

it is better to obtain s_k from a QR or singular value decomposition (SVD) of $J(x_k)$ (see [Golub, 1969]).

We can view (2.5) as defining a quadratic model in $x = x_k + s$ of the least-squares loss function (2.1):

$$(2.6) \qquad q_k^G(x) = \tfrac{1}{2}R(x_k)^TR(x_k) + (x-x_k)^TJ(x_k)^TR(x_k) + \\ + \tfrac{1}{2}(x-x_k)^TJ(x_k)^TJ(x_k)(x-x_k) \ .$$

From (2.1), (2.2), (2.3) we see that the difference between this Gauss-Newton model and the usual Newton model obtained from a quadratic Taylor expansion around x_k is just the term $\tfrac{1}{2}(x-x_k)^T[\sum_i r_i(x_k)\nabla^2 r_i(x_k)](x-x_k)$.

3. An Augmentation of the Gauss-Newton Hessian

Our purpose in this section is to suggest a way to augment the Gauss-Newton model (2.6) by adding an approximation to the difference between it and the quadratic Taylor expansion to obtain

$$(3.1) \qquad q_k^S(x) = \tfrac{1}{2}R(x_k)^TR(x_k) + (x-x_k)^TJ(x_k)^TR(x_k) + \\ + \tfrac{1}{2}(x-x_k)^T[J(x_k)^TJ(x_k) + S_k](x-x_k) \ .$$

We will suggest an approximation rule for S_k which is simple, general and geometric. The approach is to decide on a

set of desirable characteristics for the approximant and then to select S_{k+1} to be the nearest such feasible point to S_k. The rationale is that every point in the feasible set incorporates equally well the new information gained at x_{k+1} and that taking the nearest point (in a sense to be explained later) corresponds to destroying as little of the information stored in S_k as possible.

Currently we begin with $S_0 = 0$, since this is both cheap and reasonable in the sense that $q_0^S = q_0^G$.

First let us decide on the properties S_{k+1} should have. Remember that it is to approximate $\sum r_i(x_{k+1})\nabla^2 r_i(x_{k+1})$ and so it should obviously be symmetric. It is easy to find examples where the term to be approximated is indefinite, so we reject any restriction on the eigenvalues of S_{k+1}, except that it is reasonable to make $S_{k+1} + J_{k+1}^T J_{k+1}$ positive definite if this is consistent with other properties. Finally, we want to incorporate the new information about the problem, J_{k+1} and R_{k+1}, into S_{k+1}. The standard way to do this is to ask the second order approximant to transform the current x-change into the observed first order change, i.e.,

$$
\begin{aligned}
S_{k+1}\Delta x_k &\doteq \sum r_i(x_{k+1})\nabla^2 r_i(x_{k+1})\Delta x_k \\
&\doteq \sum r_i(x_{k+1})(\nabla r_i(x_{k+1}) - \nabla r_i(x_k)) \qquad (3.2) \\
&= J_{k+1}^T R_{k+1} - J_k^T R_{k+1} \equiv y_k \ .
\end{aligned}
$$

It is perhaps worth noting in passing that we tested several

choices for y_k including the Broyden-Dennis [Dennis, 1973] choice $J_{k+1}^T R_{k+1} - J_k^T R_k - J_{k+1}^T J_{k+1} \Delta x_k$ and the Betts [1976] choice $J_{k+1}^T R_{k+1} - J_k^T R_k - J_k^T J_k \Delta x_k$. Happily, (3.2), which makes more use of the structure of the problem, was the slight but clear winner. In summary then, we choose $S_0 = 0$, $S_{k+1} \in \mathcal{S} = \{S : S = S^T \text{ and } S\Delta x_k = y_k\}$.

Our choice of S_{k+1} from $\mathcal{S}$ is made in analogy with the DFP method for unconstrained minimization [Dennis & Moré, 1977]. Before giving the formula and its properties, we review some useful notation.

If A is any real matrix, then the Frobenius norm of A is $\|A\|_F \equiv (\sum A_{ij}^2)^{\frac{1}{2}}$. If B is any symmetric positive definite matrix, then B has a symmetric, positive definite square root, $B^{\frac{1}{2}}$. Define $\|A\|_{F,B} = \|B^{-\frac{1}{2}} A B^{-\frac{1}{2}}\|_F$. This weighted Frobenius norm is a natural analog of the Frobenius norm for a matrix when the standard inner product norm on the domain is replaced by $\|x\|_B = (x^T B x)^{\frac{1}{2}}$. The following theorem gives the update formulas as well as their defining properties. The proof is a straightforward modification of Theorem 7.3 of [Dennis and Moré, 1977].

THEOREM 3.1: Let $v^T \Delta x_k > 0$. Then for any positive definite symmetric matrix H for which $H\Delta x_k = v$,

$$\min \|S - S_k\|_{F,H} \qquad \text{for } S \in \mathcal{S}$$

is solved by

$$S_{k+1} = S_k + \frac{(y_k - S_k\Delta x_k)v^T + v(y_k - S_k\Delta x_k)^T}{\Delta x_k^T v} -$$

$$- \frac{\Delta x_k^T(y_k - S_k\Delta x_k)vv^T}{(\Delta x_k^T v)^2} \quad .$$

In NL2SOL we compute S_{k+1} corresponding to $v = \Delta g_k = J_{k+1}^T R_{k+1} - J_k^T R_k$. This corresponds to weighting the change by any positive definite symmetric matrix that sends Δx_k to Δg_k. Thus we hope the metric being used is not too different from that induced by the natural scaling of the problem.

4. Sizing the Hessian Augmentation

It is well known by now that the update methods do not generate approximations that become arbitrarily accurate as the iteration proceeds. On the other hand, we know that for zero residual problems, S_k should ideally converge to zero and that if it does not at least become small in those cases, then the augmented model (3.1) cannot hope to compete with (2.6), the Gauss-Newton model.

The crux of the problem can be seen by observing that even if R_{k+1} happened to be zero and even if y_k defined by (3.2) were used to make the update to S_k, then $S_{k+1}\Delta x_k = y_k = 0$, but S_{k+1} would be the same as S_k on the orthogonal complement of $\{\Delta x_k, v\}$.

We use a straightforward modification of the Oren-Luenburger self scaling technique [Oren, 1973]. The idea is to update $\tau_k S_k$, rather than S_k, to get S_{k+1}. The scalar τ_k is chosen to try

to shift the spectrum of S_k in hopes that the spectrum of $\tau_k S_k$ will overlap that of the second order term we are approximating. We could take the scalar to be

$$\frac{\Delta x_k^T y_k}{\Delta x_k^T S_k \Delta x_k} \doteq \left[\frac{\Delta x_k^T [\sum r_i(x_{k+1}) \nabla^2 r_i(x_{k+1})] \Delta x_k}{\Delta x_k^T \Delta x_k}\right] \left[\frac{\Delta x_k^T S_k \Delta x_k}{\Delta x_k^T \Delta x_k}\right]^{-1} .$$

We prefer to call this sizing, and since we are primarily concerned with S_k being too large, we actually take

$$\tau_k = \min \left\{ \left| \frac{\Delta x_k^T y_k}{\Delta x_k^T S_k \Delta x_k} \right|, 1 \right\} . \tag{4.1}$$

Whatever this strategy is called, notice that when $R_{k+1} = 0$, our $y_k = 0$, and so $\tau_k = 0$ and $S_{k+1} = 0$. The use of sizing factor (4.1) made a great difference in the performance of the algorithm.

5. Adaptive Quadratic Modeling

In section 3 we noted that $S_0 = 0$, which means that the augmented model (3.1) is initially equal to the Gauss-Newton model (2.6). Tests have shown that often $q_k^G(x_{k+1})$ predicts $f(x_{k+1})$ better than $q_k^S(x_{k+1})$ for small k, so it seems useful to have some way to decide which model to use to determine the step.

Betts [1976] also starts with $S_0 = 0$, and he takes Gauss-Newton steps for at least p iterations and until Δx_k is small enough to make it likely that x_{k+1} is near x^*. It seems therefore as though his aim is to make a last few refining iterations

based on the augmented Hessian. The heuristic we use in NL2SOL usually uses the augmented Hessian much sooner.

NL2SOL uses the double dogleg [Dennis & Mei, 1975] strategy to pick Δx_k. This modification of the Powell dogleg algorithm [Powell, 1970] is based on the model, trust-region concept which also underlies the Levenberg-Marquardt and Goldfeld-Quandt-Trotter [1966] algorithms, particularly as viewed by Hebden [1973]. In fact, the dogleg step is properly thought of as a cheap approximation to the step that the relevant one of the latter two algorithms would generate.

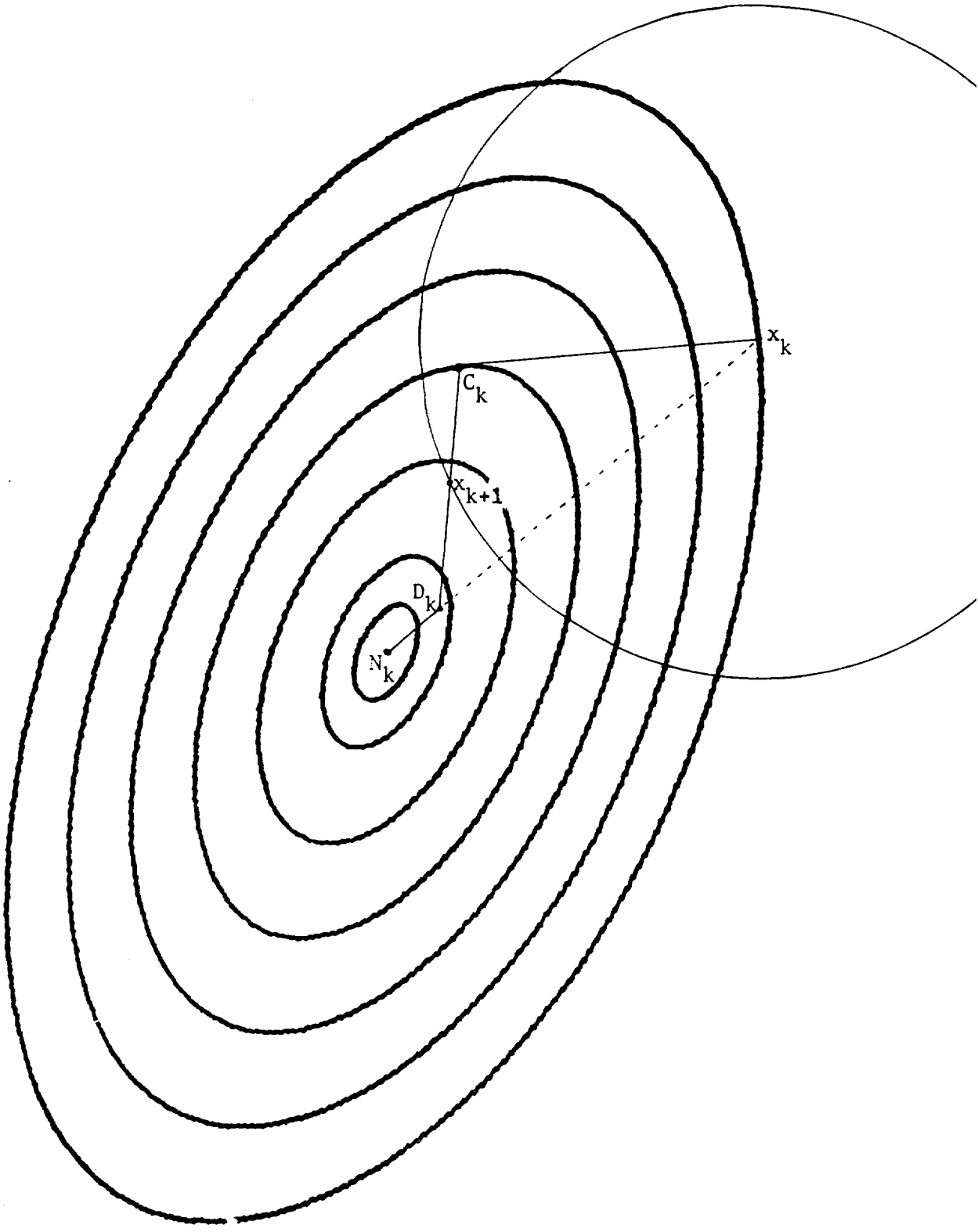
The important thing is the idea of having at x_k a local quadratic model q_k of f and an estimate of a region in which q_k is trusted to represent f. The next point x_{k+1} is chosen to approximately minimize q_k in this region or to minimize q_k in an approximation to this region. In either case, the information gained about f at x_{k+1} is then used to update the model and also to update the size or shape of the trust region.

We begin with the assumption that q_0^G holds globally. Since the trust region revision is always based on the length of the step just taken, this causes the radius to be set automatically by the initial Gauss-Newton step. Some advantages of this scheme will be mentioned in the next section, but it does have some problems. If the Gauss-Newton step is too long, the trust region may have to be shrunk repeatedly with attendant evaluations of the residual function R to obtain an acceptable x_1. Much more serious is the possibility of overflow. The initial assumption of global linearity can be overruled by assigning a small value to LMAX0, the

maximum length allowed for the very first step attempted.

The picture given here will perhaps be helpful. The ellipses represent the contours of q_k and the circle is the trust region. The point N_k is the "Newton step" or global minimum of the convex quadratic model q_k, and the point C_k is the "Cauchy point" or minimum of q_k along its negative gradient direction. The "double dogleg" point D_k is chosen using the parameter BIAS to be along the Newton direction far enough so that $q_k(C_k) \geq q_k(D_k) \geq q_k(N_k)$. In fact, we choose D_k so that q_k is monotone nonincreasing between C_k and D_k.

Since we were using this adaptive approach, it is not surprising that we also thought of using the new information at x_{k+1} to select between q^S_{k+1} and q^G_{k+1} for use in determining x_{k+2}. Our decision rule is rather straightforward. Since $S_0 = 0$, we clearly begin using the Gauss-Newton model. After making a prospective step based on the currently preferred model to obtain, say, x^1_{k+1}, we compute R^1_{k+1} and f^1_{k+1}. If $f^1_{k+1} > f_k$ then x^1_{k+1} is discarded, but first the other model is evaluated at x^1_{k+1} to see how well it agrees with f^1_{k+1}. If there is not sufficient agreement between f^1_{k+1} and the other model, then we keep the original model preference, shrink the trust region, and try again. If the agreement is sufficiently good then we change our model preference and, with the same trust region, compute x^2_{k+1}. If x^2_{k+1} is unacceptable then the trust region is shrunk and we repeat the above process on the smaller trust region with whichever model gave the least function value, but now we no longer consider changing models while continuing to seek an acceptable x_{k+1}.


x_k
C_k
x_{k+1}
D_k
N_k

When an acceptable x_{k+1} is found, $q_k^S(x_{k+1})$ and $q_k^G(x_{k+1})$ are compared to f_{k+1}. We have found that it is best to retain the currently preferred model unless the other model does a significantly better job of predicting the new function value.

6. Linear Algebra

Nonlinear problems are almost always solved approximately by solving a related sequence of linear problems. The solution of these linear approximations has been found to be a crucial part of many implementations, both in its effect on performance and, more often, in extending the applicability of the program.

The purpose of this section is to outline the linear algebra our current program uses. We do not feel that we are doing anything overtly stupid but we believe this part of the algorithm is the most susceptible to improvement.

The update to S_{k-1} is carried out as in [Gay, 1976]. Remember that S_k is a $p \times p$ symmetric indefinite matrix. Since $p < 20$ is the rule, we have not tried to conserve storage by exploiting symmetry.

The important linear algebra considerations are in the computation of the "Newton step". There are two different cases depending on the current model preference. Since $S_0 = 0$ is the usual case, the first step is a Gauss-Newton step. We actually compute the Gauss-Newton step in different ways, using the SVD or QR decomposition, depending on the situation. This step, defined in section 2, comes from solving the linear least squares problem

(6.1) $$J_k s \doteq -R_k \ .$$

In order to solve (6.1) in the least squares sense, we first apply Householder transformations to effect a QR decomposition of J_k. We then apply the techniques of Cline, Moler, Stewart and Wilkinson [1977] to estimate the condition number of R and hence of J_k. If the conditioning is acceptable, then the step is computed in the obvious way. If J_k and R are judged to be ill-conditioned, then we carry out an SVD on R.

Recall that when $k = 0$ we are assuming that the problem is linear (unless the user specifies otherwise through LMAX0), and so we want to use a currently recommended procedure to solve the linear least-squares problem (6.2.0), where

(6.2.k) $$Rs \doteq -Q^T R_k .$$

In the case $k = 0$, our condition test is controlled by RKTOL (see Usage Summary), the default for which amounts to

(6.3) $$\kappa_2(R) \leq (\text{Macheps})^{-\frac{1}{2}}$$

where $\kappa_2(R)$ denotes the Euclidean condition number (i.e., ratio of largest to smallest nonzero singular value) of R and Macheps is the unit rounding error. At present we make a call to MINSOL [Coleman et al., 1977] with the default condition that the rank of R is reduced by setting its smallest singular values to zero until the condition number is bounded as above.

In the previous section we mentioned disadvantages that arise from this assumption of linearity, but we feel there is an important advantage to the user. In our experience, users with a mixture of linear and nonlinear problems will frequently treat all the problems as nonlinear rather than learn to interface with two programs. In fact, this is not unreasonable from the occasional user's point of view.

Now suppose $k > 0$ and R and J_k are ill-conditioned. As mentioned above, we obtain an SVD of R (by calling the EISPACK routine MINFIT [Garbow et al., 1977]). In this case, we improve the condition by adding to the small singular values of R until $\kappa_2(R)^2 \doteq 1/\text{EPSLON}$, where EPSLON is an eigenvalue modification tolerance (see Usage Summary). The reason the rank is not reduced is to avoid doing the minimization prematurely in a parameter subspace.

For many problems that are at least moderately difficult, Gauss-Newton steps are only computed for the first few iterations. The augmented model quickly becomes the preferred one. Thus we have to solve the symmetric $p \times p$ linear system

$$(S_k + J_k^T J_k)s = -J_k^T R_k \quad . \tag{6.4}$$

We do form the augmented Hessian explicitly since it seems that the addition of S_k submerges any ill effects of forming $J_k^T J_k$.

We currently deal with negative curvature when computing the "Newton" step by effectively modifying the augmented Hessian, if necessary, to be positive definite. Since p is small, we do not presently use methods based on the Cholesky and symmetric

indefinite factorizations. Instead we do a symmetric eigendecomposition and obtain modified eigenvalues $\hat{\sigma}_1, \cdots, \hat{\sigma}_p$ from the original eigenvalues $\sigma_1, \cdots, \sigma_p$ according to the rule

$$\hat{\sigma} = \begin{cases} \sigma & \text{if } \sigma \geq \varepsilon \\ \varepsilon/[1 + \ln(1 + \varepsilon - \sigma)] & \text{if } \sigma < \varepsilon , \end{cases}$$

where $\varepsilon = \max\{\varepsilon_1, \varepsilon_2\}$ with $\varepsilon_1 = \text{EPSLON} \cdot \max\{\sigma_1, \cdots, \sigma_p\}$, $\varepsilon_2 = 100 \cdot \text{Macheps} \cdot \delta$, and δ is the maximum diagonal element of $J_k^T J_k$. We include ε_2 to guard against the unlikely case that considerable cancellation occurs in computing the sum $J_k^T J_k + S_k$, and we arrange the calculations for the Gauss-Newton model so that we compute the same "Newton" step from it as from the augmented model with $S_k = 0$.

7. Test Results

We have run NL2SOL on a number of the test problems reported in the literature. In particular, we have run it on the test problems listed in [Gill & Murray, 1976], which include all those considered in [Betts, 1976]. Table I below gives the results we obtained on the Gill & Murray test set with default stopping tolerances for the machine used, the IBM 370/168 at Cornell University. The columns labelled n and p specify the dimensions of the problems, while those labelled N_f and N_g tell how many function (i.e., residual vector) and gradient (i.e., Jacobian matrix) evaluations respectively were required to achieve convergence. The type of convergence attained is listed in the column

labelled CT: F means function convergence: $f(x) \le \text{Macheps}^{3/2} \doteq 3.3 \times 10^{-24}$; G means gradient convergence: $\|J^T R(x)\|_2 \le 10^{-10}$; C means cosine convergence: the maximum absolute cosine between $R(x)$ and the columns of $J(x)$ is at most $5 \times 10^7 \times \text{Macheps} \doteq 1.11 \times 10^{-8}$; and X means X-convergence: a poor step of length at most $(1 + \|x\|_2) \times \text{Macheps} \times 10^3 \doteq (1 + \|x\|_2) \times 2.22 \times 10^{-13}$ was attempted. Table II below gives the original reference for each problem in Table I.

We have encountered a couple of test problems on which NL2SOL performs poorly, including one due to Meyer [1970] and the one obtained (at the suggestion of Jorge Moré [private communication]) from the Kowalik & Osborne problem by adding 10 to each component of the residual vector. Many eigenvalue modifications occur on these problems, so we suspect that a better way of dealing with negative (or not sufficiently positive) curvature in the Hessian approximation may be needed. We intend to experiment with some promising alternatives and report about them later if they succeed.

Acknowledgment

We are grateful to Virginia Klema for helpful discussions, to Scott Smolka for some programming help, and to Mark Gelfand for his programming of DMINIMIZ, a nonlinear robust regression program with which we tested various ideas incorporated into NL2SOL. We give special thanks to Stephen Peters for his outstanding help in preparing the NL2SOL package. In addition, we are happy to thank the various people who have given us useful comments after running NL2SOL on their computers.

TABLE I

Performance of NL2SOL on Gill & Murray Test Set

Problem Name	n	p	N_f	N_g	CT	Note
Rosenbrock*	2	2	9	7	F,G	
Helix*	3	3	12	11	F,G	
Singular*	4	4	15	15	G	
Woods*	7	4	63	47	F,G	
Zangwill*	3	3	3	3	F,G	1
Engvall*	5	3	18	15	F,G	
Branin*	2	2	2	2	F,G	
Beale*	3	2	9	7	F,G	
Miele*	5	4	16	15	G	2
Box 3*	10	3	7	7	F,G	
Davidon 1*	15	15	2	1	X	3
Freudenstein & Roth	2	2	8	8	C	4
Watson 6	31	6	8	8	C	
Watson 9	31	9	9	9	C,G	
Watson 12	31	12	11	10	C,G	5
Watson 20	31	20	6	6	G	
Chebyquad 8	8	8	30	18	C	
Chebyquad 9*	9	9	11	7	F,G	5
Chebyquad 10	10	10	51	28	C	5
Brown & Dennis	20	4	21	18	C	6
Bard	15	3	7	7	C,G	
Jennrich & Sampson	10	2	16	14	C	
Kowalik & Osborne	11	4	13	12	C	
Osborne 1	33	5	9	9	C	
Osborne 2	65	11	17	14	C,G	
Madsen	3	2	12	12	C	

*denotes zero residual problems: $f(x^*) = 0$.

Note 1: If LMAX0, the limit on the length of the very first step tried, were increased slightly from its default value of 100, say to 101, then NL2SOL would solve this problem with $N_f = N_g = 2$.

Note 2: This problem differs from the original Miele problem described in [Cragg & Levy, 1969] in that it has an additional residual component $r_5(x) = x_4 - 1$, which has the effect of forcing x_4 to move more rapidly towards its optimal value. NL2SOL gave similar performance on the original Miele problem: G convergence with $N_f = N_g = 16$.

Note 3: This is a linear least-squares problem so ill-conditioned that it cannot be accurately solved from the approximate singular value decomposition computed in double precision on an IBM 370. The first step that NL2SOL attempts is thus both very poor and very small, so NL2SOL recognizes X-convergence at the starting point. Ironically, if NL2SOL had simply used the QR decomposition which it first computed, rather than going on to compute a singular value decomposition of the R just obtained, then the problem would have been solved in one step.

Note 4: NL2SOL found a local solution of this problem; the residual vector vanishes at the global solution.

Note 5: Gill & Murray [1976] do not list results for Watson 12, Chebyquad 9, or Chebyquad 10. Jorge Moré [private communication] suggested that these might be interesting test problems.

Note 6: Gill & Murray [1976] call this problem "Davidon 2".

Table II

Original Sources of Test Problems

Problem Name	Source
Rosenbrock	[Rosenbrock, 1960]
Helix	[Fletcher & Powell, 1963]
Singular	[Powell, 1962]
Woods	[Colville, 1968]
Zangwill	[Zangwill, 1967]
Engvall	[Engvall, 1966]
Branin	[Branin, 1971]
Beale	[Beale, 1958]
Miele	[Gill & Murray, 1976]
Box 3	[Box, 1966]
Davidon 1	[Davidon, 1976]
Freudenstein & Roth	[Freudenstein & Roth, 1963]
Watson 6, 9, 12, 20	[Kowalik & Osborne, 1968]
Chebyquad 8, 9, 10	[Fletcher, 1965]
Brown & Dennis	[Brown & Dennis, 1971]
Bard	[Bard, 1970]
Jennrich & Sampson	[Jennrich & Sampson, 1968]
Kowalik & Osborne	[Kowalik & Osborne, 1968]
Osborne 1, 2	[Osborne, 1972]

REFERENCES

BARD, Y. (1970), Comparison of gradient methods for the solution of nonlinear parameter estimation problems. SIAM J. Numer. Anal. 7, pp. 157-186.

BEALE, E.M.L. (1958), On an iterative method for finding a local minimum of a function of more than one variable. Tech. Rept. No. 25, Statistical Techniques Research Group, Princeton University, Princeton, New Jersey.

BETTS, J.T. (1976), Solving the nonlinear least square problem: Application of a general method. J. Optimization Theory Appl. 18, pp. 469-484.

BOX, M.J. (1966), A comparison of several current optimization methods and the use of transformations in constrained problems. Comput. J. 9, pp. 67-77.

BRANIN, F.H. (1971), Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. IBM J. Res. Develop. 16, pp. 504-522.

BROWN, K.M. and DENNIS, J.E. (1971), A new algorithm for nonlinear least-squares curve fitting. in Mathematical Software edited by John R. Rice, Academic Press, New York, pp. 391-396.

CLINE, A.; MOLER, C.; STEWART, G.W.; and WILKINSON, J.H. (1977), On an estimate for the condition number of a matrix. informal manuscript.

COLEMAN, D.; HOLLAND, P.; KADEN, N.; and KLEMA, V. (1977), A system of subroutines for iteratively reweighted least squares computations. NBER Working Paper No. 189.

COLVILLE, A.R. (1968), A comparative study of nonlinear programming codes. IBM New York Scientific Center Tech. Rept. No. 320-2949.

CRAGG, E.E. and LEVY, A.V. (1969), Study on a supermemory gradient method for the minimization of functions, J. Optimization Theory Appl. 4, pp. 191-205.

DAVIDON, W.C. (1976), New least-square algorithms, J. Optimization Theory Appl. 18, pp. 187-197.

DENNIS, J.E. (1973), Some computational techniques for the nonlinear least squares problem. in Numerical Solution of Systems of Nonlinear Equations edited by G.D. Byrne and C.A. Hall, Academic Press, New York.

DENNIS, J.E. (1977), Nonlinear least squares and equations. in The State of the Art of Numerical Analysis edited by D. Jacobs, Academic Press, London.

DENNIS, J.E. and MEI, H.H-W. (1975), An unconstrained optimization algorithm which uses function and gradient values. Cornell Computer Science Tech. Rept. TR 75-246, (to appear in J. Optimization Theory Appl.).

DENNIS, J.E. and MORÉ, J.J. (1977), Quasi-Newton methods, motivation and theory. SIAM Rev. 19, pp. 46-89.

DENNIS, J.E. and WELSCH, R.E. (1976), Techniques for nonlinear least squares and robust regression. 1976 Proceedings of the ASA Statistical Computing Section, American Statistical Association, Washington, D.C., pp. 83-87.

ENGVALL, J.L. (1966), Numerical algorithm for solving over-determined systems of nonlinear equations. NASA document N70-35600.

FLETCHER, R. (1965), Function minimization without evaluating derivatives--a review. Comput. J. 8, pp. 33-41.

FLETCHER, R. and POWELL, M.J.D. (1963), A rapidly convergent descent method for minimization. Comput. J. 6, pp. 163-168.

FREUDENSTEIN, F. and ROTH, B. (1963), Numerical solutions of nonlinear equations. J. Assoc. Comput. Mach. 10, pp. 550-556.

GARBOW, B.S.; BOYLE, J.M.; DONGARRA, J.J.; and MOLER, C. (1977), Matrix Eigensystem Routines--EISPACK Guide Extension, Lecture Notes in Computer Science 51, Springer-Verlag, Berlin, Heidelberg, and New York.

GAY, D.M. (1976), Representing symmetric rank 2 updates. NBER Working Paper No. 124.

GILL, P.E. and MURRAY, W. (1976), Nonlinear least squares and nonlinearly constrained optimization. in Lecture Notes in Mathematics No. 506 Numerical Analysis, Springer-Verlag, Berlin, Heidelberg, and New York.

GILL, P.E. and MURRAY, W. (1976), Algorithm for the solution of the non-linear least-squares problem. NPL Report NAC 71.

GOLDFELD, S.M.; QUANDT, R.E.; and TROTTER, H.F. (1966), Maximization by quadratic hill-climbing. Econometrica 34, pp. 541-551.

GOLUB, G.H. (1969), Matrix decompositions and statistical calculations. in Statistical Computation edited by R.C. Milton and J.A. Nelder, Academic Press, New York, pp. 365-397.

GOLUB, G.H. and REINSCH, C. (1971), Singular value decomposition and least squares solutions. in Handbook for Automatic Computation, Vol. II: Linear Algebra edited by J.H. Wilkinson and C. Reinsch, Springer-Verlag, Berlin, Heidelberg, New York.

HEBDEN, M.D. (1973), An algorithm for minimization using exact second derivatives. A.E.R.E. Harwell Report TP515.

JENNRICH, R.I. and SAMPSON, P.F. (1968), Application of stepwise regression to nonlinear estimation. Technometrics 10, pp. 63-72.

KOWALIK, J.S. and OSBORNE, M.R. (1968), Methods for Unconstrained Optimization Problems, American Elsevier, New York.

LAWSON, C.L. and HANSON, R.J. (1974), Solving Least Squares Problems, Prentice Hall, Englewood Cliffs, New Jersey.

OREN, S.S. (1973), Self-scaling variable metric algorithms without line search for unconstrained minimization. Math. Comput. 27, pp. 873-885.

OSBORNE, M.R. (1972), Some aspects of nonlinear least squares calculations. in Numerical Methods for Nonlinear Optimization edited by F.A. Lootsma, Academic Press, New York and London.

POWELL, M.J.D. (1962), An iterative method for finding stationary values of a function of several variables. Comput. J. 5, pp. 147-151.

POWELL, M.J.D. (1970), A new algorithm for unconstrained optimization. in Nonlinear Programming edited by J.B. Rosen, O.L. Mangasarian, and K. Ritter, Academic Press, New York.

ROSENBROCK, H.H. (1960), An automatic method for finding the greatest or least value of a function. Comput. J. 3, pp. 175-184.

WEDIN, P-A. (1972), (1974a), The non-linear least squares problem from a numerical point of view, I and II. Lund Univ. Computer Sci. Tech. Repts.

WEDIN, P-A. (1974b), On surface dependent properties of methods for seperable non-linear least squares problems. Inst. för tellämpad matematik, Box 5073 Stockholm 5, ITM Arbetsrapport nr. 23.

WEDIN, P-A. (1974c), On the Gauss-Newton method for the non-linear least squares problem. Inst. för tellämpad matematik, Box 5073 Stockholm 5, ITM Arbetsrapport nr. 24.

ZANGWILL, W.J. (1967), Nonlinear programming via penalty functions. Management Sci. 13, pp. 344-358.

A P P E N D I X

NL2SOL Usage Summary

1. Purpose

Given a continuously differentiable function (residual vector) $R(x) = (R_1(x), R_2(x), \ldots, R_n(x))^T$ of p parameters $x = (x_1, x_2, \ldots, x_p)^T$, NL2SOL attempts to find a parameter vector x^* which minimizes the sum-of-squares function $F(x) = \frac{1}{2}\Sigma_{i=1}^{n} R_i(x)^2$.

2. Method

Reference 1 explains the algorithm realized by NL2SOL in detail. The algorithm amounts in part to a variation on Newton's method in which part of the Hessian matrix is computed exactly and part is approximated by a quasi-Newton technique; once the iterates come sufficiently close to a (local) solution, they usually converge quite rapidly. To promote convergence when the iterates lie far from a solution, NL2SOL may choose to step in an adaptively chosen combination of the "Newton" and steepest-descent directions. At times the algorithm reduces to the Gauss-Newton method, while at other times it may bear some similarity to the Levenberg-Marquardt method. On large-residual problems (in which $F(x^*)$ is large), however, NL2SOL often works much better than these methods.

3. Calling Sequence

```
CALL NL2SOL(N, P, X, MAXITR, CALCR, CALCJ, IPARM, RPARM, IWORK, WORK,
            UIPARM, URPARM, UFPARM)
```

Note: In DOUBLE PRECISION versions of NL2SOL, all variables and arrays termed REAL below are actually DOUBLE PRECISION.

N (input INTEGER) is the number of components in the residual vector R.

P (input INTEGER) is the number of parameters on which R depends.

X (I/O, REAL array of length P) on input is an initial guess at the desired solution x^*; on output, X contains the best parameter estimate that NL2SOL has found so far (i.e. the one giving the least value of F).

MAXITR (input INTEGER), if positive, is the maximum number of iterations that NL2SOL should perform: this value is stored in IPARM(MXITER) = IPARM(3). If MAXITR is not positive, then IPARM(MXITER) is left unchanged.

CALCR (input subroutine) computes the residual vector R = R(X) when called by:

```
CALL CALCR(N, P, X, NFCALL, R, UIPARM, URPARM, UFPARM)
```

CALCR should not change its first four arguments. NFCALL is the invocation count of CALCR and may be stored to indicate which X belongs to any intermediate results that may be helpful to CALCJ; such results may be stored in UIPARM and URPARM (or in COMMON).

CALCJ (input subroutine) computes the Jacobian matrix J = J(X) of first partials

$J_{ij} = \frac{\partial R_i}{\partial x_j}(X)$ when called by:

CALL CALCJ(NN, N, P, X, NFCALL, R, J, UIPARM, URPARM, UFPARM, CALCR)

CALCJ should not change its first 6 arguments. NN is the declared row dimension of J (i.e. J is declared to be REAL J(NN,P)), which in fact always equals N in the present version of NL2SOL. NFCALL is the invocation count of CALCR at the time R was computed at the X passed; thus NFCALL may be used to decide whether -- or which copy of -- any intermediate results stored by CALCR are valid for this X. (At most two copies of intermediate results would have to be stored, since CALCJ is always called at the X passed on the latest or next-to-latest call to CALCR.)

R = R(X) and CALCR are passed to CALCJ to facilitate computing J by finite differences. The subroutine FDJ, supplied with the NL2SOL package, may be passed as CALCJ to compute an approximate J by forward differences.

Note: the subroutines passed for CALCR and CALCJ must be declared EXTERNAL in the calling program.

IPARM (I/O INTEGER array of length 19) on input contains certain quantities (such as limits on the number of iterations and calls on CALCR allowed) that control the behavior of NL2SOL and on output contains various counts and other quantities of interest: see the detailed description given below. If IPARM(1) = 0 on input, then default values are supplied for the input quantities of both IPARM and RPARM. The caller may supply nondefault values for selected quantities in IPARM and RPARM by first executing a CALL DFAULT(IPARM, RPARM) and then assigning the appropriate nondefault values before calling NL2SOL.

RPARM (I/O REAL array of length 34) on input contains certain quantities (such as convergence tolerances) that control the behavior of NL2SOL and on output contains various quantities of interest (such as the current value of F(X) and the norm of the most recent gradient computed): see the detailed description given below.

IWORK (INTEGER scratch array of length 2P).

WORK (REAL scratch array of length 2P(N + 2P + 7) + 3N) contains certain quantities that may be of interest on output: after any return with IPARM(1) < 10,

R(X), the residual vector giving the least value of F so far encountered, starts at WORK(IPARM(17));

G, the gradient of F at X, starts at WORK(IPARM(16));

J, the Jacobian matrix of R at X, starts at WORK(IPARM(18)) and is dimensioned J(N,P);

GG, the current approximation to the Hessian of F, starts at WORK(IPARM(19));

S, the current quasi-Newton approximation to the second-order part of the Hessian of F, starts at WORK(IPARM(15)) = WORK(1).

UIPARM (INTEGER array of length determined by the caller) is passed without change to CALCR and CALCJ and may be used in any way that the caller may find convenient.

URPARM (REAL array of length determined by the caller) is passed without change to CALCR and CALCJ.

UFPARM (subroutine) is passed without change to CALCR and CALCJ. If there is no need for such a subroutine, then the caller may pass an arbitrary variable or constant. But if an actual subroutine is passed for UFPARM, then it should be declared EXTERNAL in the calling program.

4. Return Codes

When NL2SOL returns, IPARM(1) contains one of the following return codes:

3 - X-convergence (see RPARM(XCONCR) below);
4 - gradient convergence (see RPARM(FCONCR), RPARM(GCONCR), and RPARM(GRDMIN) below);
5 - function evaluation limit reached (see IPARM(MXFCAL) below);
6 - iteration limit reached (see IPARM(MXITER) below);
7 - STOPX returned .TRUE. after ASSESS (this return is only possible when a system-dependent STOPX function is used in place of the one included in the NL2SOL package -- see the discussion of STOPX below);
8 - STOPX returned .TRUE. after UPDATE;
9 - error in an EISPACK routine (IMTQL2 or MINFIT) -- let us know if this occurs;
10 - P is less than 1 or N is less than P;
11 through 22 - IPARM(IPARM(1)-10) is out of range;
23 through 50 - RPARM(IPARM(1)-22) is out of range.

5. COMMON

NL2SOL makes use of two COMMON blocks:

```
COMMON /NL2ICM/ ISTART, MXFCAL, MXITER, NITER,  NFCALL, NGCALL,
                IASSES, KASSES, SUSED,  INCRAD, NFGCAL, USES,
                OUTLEV, FDTYPE

COMMON /NL2RCM/ LMAX0,  EPSLON, FCONCR, GCONCR, GRDMIN, XCONCR,
                BIAS,   TUNER1, TUNER2, TUNER3, TUNER4, INCFCR,
                DECFCR, FUZZ,   COSMIN, RADIUS, DOGLOC, GTHESG,
                MU,     SIZE,   WSCALE, GTSTEP, STNORM, FDIF,
                PHI,    FLSTGD, F,      FNEXT,  GNORM,  MAXCOS,
                RKTOL,  FDSTEP, FERROR
```

These variables are used to index the IPARM and RPARM arrays respectively. They are initialized in a BLOCK DATA program -- and the caller's only concern with these COMMON blocks should be to ensure that the BLOCK DATA subprogram is loaded along with the other modules of the NL2SOL package. The BLOCK DATA subprogram supplied with the NL2SOL package initializes the variables in NL2ICM to 1 through 14 in order of appearance and the variables in NL2RCM to 1 through 18, then 20 through 34 in order of appearance.

6. IPARM Entries

Before describing the IPARM entries in detail, we list their indices alphabetically together with the corresponding subscript:

FDTYPE = 14	NFCALL = 5
IASSES = 7	NFGCAL = 11
INCRAD = 10	NGCALL = 6
ISTART = 1	NITER = 4
KASSES = 8	OUTLEV = 13
MXFCAL = 2	SUSED = 9
MXITER = 3	USES = 12

The INPUT entries include: ISTART, MXFCAL, MXITER, and OUTLEV. Default values for these are listed in square brackets [] at the beginning of the description of each. These values are supplied when NL2SOL is called with IPARM(1) = 0 and may also be obtained via a CALL DFAULT(IPARM, RPARM) statement.

The OUTPUT entries include: ISTART, NITER, NFCALL, NGCALL, IASSES, KASSES, SUSED, INCRAD, NFGCAL, and USES.

ISTART: IPARM(1) [1] should be 0, 1, or 2 in the initial call on NL2SOL; 0 means that default values should be assigned to the IPARM and RPARM arrays; 0 and 1 mean that the $N \times N$ matrix S that occupies the first N^2 locations of WORK and is used to approximate the second-order part of the Hessian of the least-squares objective function should be initialized to 0; 2 means that an initial value for S (as well as for the input entries of IPARM and RPARM) has been supplied.

IPARM(1) contains a return code when NL2SOL returns; the possible return codes are listed above (§4).

MXFCAL: IPARM(2) [200] is the maximum number of function evaluations (calls on CALCR) allowed. NL2SOL returns with IPARM(1) = 5 if this limit is reached without convergence.

MXITER: IPARM(3) [100] is the maximum number of iterations allowed. NL2SOL returns with IPARM(1) = 6 if this limit is reached without convergence. This also amounts to a limit on the number of Jacobian matrix evaluations, since there is one call on CALCJ per iteration.

NITER: IPARM(4) (output) contains the number of iterations performed.

NFCALL: IPARM(5) (output) contains the number of function evaluations (calls on CALCR) performed.

NGCALL: IPARM(6) (output) contains the number of gradient evaluations (calls on CALCJ) performed.

IASSES, KASSES, SUSED, INCRAD, NFGCAL, USES: IPARM(i) for i = 7,8,9,10,11,12 (output) contain information describing the state of the NL2SOL algorithm; this information facilitates restarting (see below) and is described in more detail in the comments at the beginning of the module NL2ITR.

OUTLEV: IPARM(13) [1] tells the module ITSMRY how often to print an iteration summary and how much to include in that summary. If IPARM(OUTLEV) = 0, then no iteration summaries are printed. Otherwise a summary is printed every $|\text{IPARM(OUTLEV)}|$ iterations. If IPARM(OUTLEV) is positive, then

the summary lines have a maximum length of 117 characters (including the carriage control character), and if IPARM(OUTLEV) is negative, then the summary lines are less detailed and have a maximum length of 66 characters. See the discussion of ITSMRY below for a description of the various quantities printed.

FDTYPE: IPARM(14) [1] is currently unused, but may see future use by a more elaborate version of FDJ.

7. RPARM Entries

Before describing the RPARM entries in detail, we list their indices alphabetically together with the corresponding subscript:

BIAS	= 7	GTSTEP	= 23
COSMIN	= 15	INCFCR	= 12
DECFCR	= 13	LMAX0	= 1
DOGLOC	= 17	MAXCOS	= 31
EPSLON	= 2	MU	= 20
F	= 28	PHI	= 26
FCONCR	= 3	RADIUS	= 16
FDIF	= 25	RKTOL	= 32
FDSTEP	= 33	SIZE	= 21
FERROR	= 34	STNORM	= 24
FLSTGD	= 27	TUNER1	= 8
FNEXT	= 29	TUNER2	= 9
FUZZ	= 14	TUNER3	= 10
GCONCR	= 4	TUNER4	= 11
GNORM	= 30	WSCALE	= 22
GRDMIN	= 5	XCONCR	= 6
GTHESG	= 18		

The INPUT entries include: LMAX0, EPSLON, FCONCR, GCONCR, GRDMIN, XCONCR, BIAS, TUNER1, TUNER2, TUNER3, TUNER4, INCFCR, DECFCR, FUZZ, COSMIN, and RKTOL. Default values for these are listed in square brackets [] at the beginning of the description of each. These values are supplied when NL2SOL is called with IPARM(1) = 0 and may also be obtained via a CALL DFAULT(IPARM, RPARM) statement.

The OUTPUT entries include: RADIUS, DOGLOC, GTHESG, MU, SIZE, WSCALE, GTSTEP, STNORM, FDIF, PHI, FLSTGD, F, FNEXT, GNORM, and MAXCOS.

LMAX0: RPARM(1) [100.0] is the maximum length allowed for the very first step that NL2SOL tries. Ordinarily (unless NL2SOL is called with IPARM(1) = 2) this first step is a Gauss-Newton step; however, if its Euclidean length exceeds RPARM(LMAX0), then it is scaled down to make its length exactly RPARM(LMAX0). Such scaling is vital on certain problems where NL2SOL would otherwise attempt a disastrously large first step.

EPSLON: RPARM(2) [$\max\{2.22\times10^{-14}, 100\mu\}$, where μ is the unit roundoff of the computer in question, i.e., the smallest positive number such that $1 + \mu > 1$ and $1 - \mu < 1$] helps determine the threshold ε at which eigenvalue modification should begin. If σ is the maximum eigenvalue of the current Hessian approximation and δ is the maximum diagonal

element of J^TJ, then $\varepsilon = \max\{\text{RPARM(EPSLON)}\cdot\sigma, 100\mu\delta\}$, and any eigenvalues less than ε are shifted upwards for the purpose of computing a "Newton" step.

FCONCR: RPARM(3) [$\mu^{3/2}$, where μ is described above with EPSLON] is a scale-dependent gradient-convergence tolerance intended for use with zero-residual problems: gradient convergence (IPARM(1) = 4) occurs if the function value F(X) (half the sum of squares) is at most RPARM(FCONCR).

GCONCR: RPARM(4) [$\mu\times5\times10^7$, where μ is described above with EPSLON] is a scale-free gradient convergence tolerance: gradient convergence (IPARM(1) = 4) occurs if the maximum absolute cosine between R(X) (the current residual vector) and J(X) (the corresponding Jacobian matrix) is at most RPARM(GCONCR).

GRDMIN: RPARM(5) [10^{-10}] is a scale-dependent gradient-convergence tolerance: gradient convergence (IPARM(1) = 4) occurs if the least-squares gradient $J^TR(X)$ has Euclidean norm at most RPARM(GRDMIN).

XCONCR: RPARM(6) [$\mu\times10^3$, where μ is described above with EPSLON] is a tolerance for X-convergence: X-convergence (IPARM(1) = 3) occurs if the current step ΔX fails to yield much improvement and $||\Delta X||_2 \leq \text{RPARM(XCONCR)}\cdot(||X||_2 + 1)$.

BIAS, TUNER1, TUNER2, TUNER3, TUNER4, INCFCR, DECFCR, FUZZ, and COSMIN: RPARM(i) for $7 \leq i \leq 15$ are values which control the behavior of the modules DBLDOG, ASSESS, CNGMOD, and UPDATE and which should normally require no tinkering. For more details, see these modules or the comments at the start of the module NL2ITR.

RKTOL: RPARM(32) [-1.0] helps decide what singular values should be discarded in computing the very first Gauss-Newton step (assuming NL2SOL is called with IPARM(1) = 0 or 1). This first step Δx_o is special in that no eigenvalue modifications are considered. It is obtained by solving the linear least-squares problem $J(x_o)\Delta x_o \doteq -R(x_o)$. The subroutine MINSOL that computes Δx_o regards as zero any computed singular values of $J(x_o)$ which are less than RPARM(RKTOL) times the largest, except that negative values of RPARM(RKTOL) are treated as though they were $\mu^{1/2}$, where μ is described above with EPSLON.

FNEXT: RPARM(29) (output) contains F(X), i.e., half the sum of squares at the best parameter estimate X yet found.

GNORM: RPARM(30) (output) contains the Euclidean norm of the gradient $J^TR(X)$ at the best parameter estimate X yet found.

MAXCOS: RPARM(31) (output) contains the maximum cosine between the residual vector R(X) and the corresponding Jacobian matrix J(X) at the best parameter estimate X yet found.

RADIUS, DOGLOC, GTHESG, MU, SIZE, WSCALE, GTSTEP, STNORM, FDIF, PHI, FLSTGD, and F: RPARM(i) for $16 \leq i \leq 28$, describe various aspects of the current iteration and facilitate restarting. See the comments at the start of the NL2ITR module for more details.

FDSTEP, FERROR: RPARM(i) for i = 33,34 [0, 0] are currently unused, but may see future use by a more elaborate version of FDJ.

8. Example

Let n = 3, p = 2, and $R(x) = \begin{bmatrix} x_1^2 + x_2^2 + x_1 x_2 \\ \sin x_1 \\ \cos x_2 \end{bmatrix}$. (This problem is due to Madsen, Reference 2.) The following FORTRAN code minimizes $F(x) = \frac{1}{2} R(x)^T R(x)$, starting from the initial guess $(3, 1)^T$, using a single-precision NL2SOL.

```
      INTEGER IPARM(19),IWORK(4)
      REAL RPARM(34), WORK(65), X(2)
      EXTERNAL MADR, MADJ
      X(1) = 3.0
      X(2) = 1.0
      IPARM(1) = 0
      CALL NL2SOL(3, 2, X, 30, MADR, MADJ, IPARM, RPARM, IWORK, WORK, 0, 0., 0)
      WRITE(6, 9001) IPARM(1), X
 9001 FORMAT(//25HNL2SOL RETURNS IPARM(1) =,I3,8H AND X =,2E15.4)
      STOP
      END
      SUBROUTINE MADR(N, P, X, NFCALL, R, UIPARM, URPARM, UFPARM)
      INTEGER N, P, NFCALL, UIPARM(1)
      REAL X(P), R(N), URPARM(1)
      EXTERNAL UFPARM
      R(1) = X(1)**2 + X(2)**2 + X(1)*X(2)
      R(2) = SIN(X(1))
      R(3) = COS(X(2))
      RETURN
      END
      SUBROUTINE MADJ(NN, N, P, X, NFCALL, R, J, UIPARM, URPARM, UFPARM, CALCR)
      INTEGER NN, N, P, NFCALL, UIPARM(1)
      REAL X(P), J(NN,P), URPARM(1)
      EXTERNAL UFPARM, CALCR
      J(1,1) = 2.0*X(1) + X(2)
      J(1,2) = 2.0*X(2) + X(1)
      J(2,1) = COS(X(1))
      J(2,2) = 0.0
      J(3,1) = 0.0
      J(3,2) = -SIN(X(2))
      RETURN
      END
```

The main program above passes MADR as CALCR and MADJ as CALCJ. Since no use is made of UIPARM, URPARM, or UFPARM, zeroes are passed for these parameters.

When the above is executed, NL2SOL will produce output summarizing the 11 iterations required to solve the problem, after which the WRITE statement in the main program will produce the line

```
NL2SOL RETURNS IPARM(1) =  4 AND X =     -0.1554E+00      0.6946E+00
```

If, say, no printed output had been desired from NL2SOL, then the statement IPARM(1) = 0 of the main program, which causes default IPARM and RPARM values to be supplied, would have been replaced by

```
      CALL DFAULT(IPARM, RPARM)
      IPARM(13) = 0
```

(See the description of IPARM(OUTLEV) in §6 above.) Other nondefault IPARM and RPARM values could be supplied similarly.

9. Local Nature of Solutions

It can easily happen that NL2SOL only finds a local minimizer of the sum-of-squares function F(x) and that a different starting guess would cause a point to be found at which F has a still smaller value. Except for cases where special conditions (such as convexity of the objective function) prevail, this shortcoming is shared by all minimization algorithms.

10. LMAX0

It should be stressed that on some problems it is necessary to give RPARM(LMAX0) = RPARM(1) a small value to prevent a disasterously large first step, one which might lead to exponent overflow or arguments out of range to intrinsic functions. Even if no disaster occurs, if NL2SOL takes many function evaluations on the first step, then performance might be improved by a much smaller (or in some cases larger) value of RPARM(LMAX0).

11. Finite Difference Jacobians

Rather than analytically differentiating R(x) and coding a corresponding subroutine for CALCJ, the caller may simply pass FDJ, which uses forward differences to compute an approximation to the Jacobian matrix. FDJ is included as part of the NL2SOL package, so it is merely necessary to include an EXTERNAL FDJ statement in the calling program. FDJ should work well on many problems, though, of course, an analytic CALCJ will work much better on certain problems.

12. STOPX

When NL2SOL is used in an interactive environment (such as CMS), it is possible to arrange for NL2SOL to be interrupted. To do this, it is necessary to replace the logical function STOPX supplied with the NL2SOL package (which always returns .FALSE.) by a system-dependent STOPX that returns .TRUE. if the "break" (i.e., "interrupt") key has been pressed since the last call on STOPX and returns .FALSE. otherwise. Then, depending on where in its main loop NL2SOL is interrupted, NL2SOL will return with IPARM(1) = 7 or 8 when the "break" key is pressed before some other return has occurred.

13. Restarting

After a return with IPARM(1) less than 9, it is possible to resume running the algorithm at the point where it was interrupted. Simply call NL2SOL again, passing the same parameters as before, with nothing changed except possibly MAXITR, IPARM(MXFCAL) = IPARM(2), or the tolerances in RPARM.

14. ITSMRY, the Iteration Summary

Included in the NL2SOL package is the subroutine ITSMRY, which is called once at the end of each iteration to print a summary of that iteration. (It is also called once before the first iteration and once before any return with IPARM(1) less than 10.) In some cases it may be desirable to replace the supplied ITSMRY with another one; this is why the calling sequence

```
CALL ITSMRY(IPARM, RPARM, P, X, G)
```

includes P, X (the best parameter estimate yet found), and G (the corresponding gradient), even though these parameters are not used by the supplied ITSMRY.

IPARM(OUTLEV) = IPARM(13) instructs the supplied ITSMRY how often to print an iteration summary and how much of a summary to print. If this value is zero, then ITSMRY returns without doing any printing. Otherwise an iteration summary is printed every |IPARM(OUTLEV)| iterations. If IPARM(OUTLEV) is positive, then the summary contains the following columns:

IT, the iteration number.
NF, the number of function evaluations (calls on CALCR) done so far.
STEP, the 2-norm of the current step.
F, the current (best-known) function value, i.e., half the sum of squared residuals.
DF, the decrease in function values achieved in the current iteration.
G, the 2-norm of the current gradient.
COSMAX, the maximum of the absolute values of the cosines between the current residual vector and the columns of the corresponding Jacobian matrix.
MODEL, a code, such as G:S or G or S:G:S, which indicates the sequence of models used in the current iteration. G stands for the Gauss-Newton model (which ignores S), while S means the model using S.
MU, the largest eigenvalue modification made while computing the current Newton step.
DOGLOC, a scalar that tells how the current dogleg step was computed: 3 means a full Newton step; between 2 and 3 means a scaled-down Newton step; between 1 and 2 means a combination of the relaxed Newton and Cauchy (i.e., steepest descent) steps; between 0 and 1 means a scaled-down Cauchy step.
RADIUS, the radius of the new trust region (in which we trust our new quadratic approximation to the least-squares objective function).
SIZE, the sizing factor applied during the last update (see Ref. 1).

In this case lines having a maximum width of 117 characters are generated.

If IPARM(OUTLEV) is negative, then the last five columns are omitted and lines having a maximum width of 66 characters are generated.

15. LINALG

Included with the NL2SOL package is a collection of linear algebra routines referred to as LINALG. On small problems execution time may be reduced significantly if some of these FORTRAN routines are replaced by their machine-language equivalents. In particular, on machines where underflows are quietly set to zero, the precautions taken in DOTPRD, MVMUL, and MTVMUL to avoid underflows are unnecessary, and execution time on "cheap" problems may be noticeably reduced just by removing these precautions.

16. Machine-Dependent Constants

Some of the NL2SOL routines have DATA statements that involve machine-dependent constants. Such DATA statements are immediately preceeded by a corresponding C$ DATA statement of the sort recognized by the IMSL FORTRAN Converter (Reference 3). Thus the Converter may be used to produce a version of NL2SOL appropriate for use on another computer (or for a different precision on the present computer). The very first card of subroutine NL2SOL tells for which computer and precision the constants in the present DATA statements are appropriate.

17. Storage Requirements

The NL2SOL package (excluding the Test Program) amounts to about 5800 lines of FORTRAN code. Many of these are comments: there are only about 1900 "internal statemetns", where an IF statement counts as two "internal statements", comments do not count at all, and every other kind of statement counts as one "internal statement". When compiled on an IBM 370 by the H-extended compiler OPT(2), this source code results in around 51200 bytes of object code. The amount of variable storage needed is listed above in §3.

18. References

1. Dennis, J.E.; Gay, D.M.; & Welsch, R.E. (1977), "An Adaptive Nonlinear Least-Squares Algorithm", NBER Working Paper No. 196.

2. Madsen, K. (1973), "An Algorithm for Minimax Solution of Overdetermined Systems of Nonlinear Equations", Report TP 559, A.E.R.E. Harwell, Oxon., England.

3. Aird, T.J.; Battiste, E.L.; & Gregory, W.C. (1977), "Portability of Mathematical Software Coded in Fortran", ACM Trans. Math. Software 3, pp. 113-127.

19. Acknowledgement

Research leading to the NL2SOL package was supported in part by National Science Foundation Grants DCR75-10143, MCS76-00324, and SOC76-14311 to the National Bureau of Economic Research, Inc.