AN ADAPTIVE NONLINEAR LEAST-SQUARES ALGORITHM

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

TR77-321
Revised July 1979

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

**NBER Computer Research Center Research supported in part by NSF Grant MCS76-00324
NSF Grant MCS78-09525
United States Army Contract DAAG29-75-C-0024

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

ABSTRACT

NL2SOL is a modular program for solving nonlinear least-squares problems 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. S is "sized" before updating, something which is similar to Oren-Luenberger scaling. The step choice algorithm is based on minimizing a local quadratic model of the sum of squares function constrained to an elliptical trust region centered at the current approximate minimizer. This is accomplished using ideas discussed by Moré, together with a special module for assessing the quality of the step thus computed. These and other ideas behind NL2SOL are discussed and its evolution and current implementation are also described briefly.

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

near zero, we will be forced to terminate on small parameter changes or to use some other convergence criteria (see Section 6). 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 $C(J(x_k))$ be approximated very well in the usual case where $p < n$ 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 $\Sigma r_i(x)\nabla^2 r_i(x)$ 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" from 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

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

it is better to obtain s_k from a QR decomposition 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 criterion function (2.1):

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

$$+ \tfrac{1}{2}(x-x_k)^T J(x_k)^T J(x_k)(x-x_k).$$

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 $\tfrac{1}{2}(x-x_k)^T[\Sigma r_i(x_k)\nabla^2 r_i(x_k)](x-x_k)$.

The conceptual difference between these two models is interesting in that it exposes some reasons for the deficiencies of the Gauss-Newton algorithm. The Newton model is based on the assumption that f can be adequately modeled by a quadratic, while the Gauss-Newton model (2.6) is shown by (2.5) to result from the stronger assumption that R can be adequately modeled by an affine function.

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

$$(3.1)\qquad q_k^S(x) = \tfrac{1}{2}R(x_k)^T R(x_k) + (x-x_k)^T J(x_k)^T R(x_k)$$

$$+ \tfrac{1}{2}(x-x_k)^T[J(x_k)^T J(x_k) + S_k](x-x_k).$$

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 $q_0^S = q_0^G$. Suppose S_k is available. First let us decide on the properti S_{k+1} should have. Remember that it is to approximate $\Sigma r_i(x_{k+1})\nabla^2 r_i(x_{k+1})$ 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}. 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.,

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

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, 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 $\|A\|_F \equiv (\Sigma A_{ij}^2)^{\frac{1}{2}}$. If B is any symmetric positive definite matrix, then

B has a symmetric, positive definite square root, $B^{\frac{1}{2}}$. Define $||A||_{F,B} = ||B^{-\frac{1}{2}}AB^{-\frac{1}{2}}||_F$. 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 $||x||_B = (x^TBx)^{\frac{1}{2}}$. The following theorem gives the update formulas as well as their defining properties. It is just a restatement 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 $H\Delta x_k = v$,

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

is solved by

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

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

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

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

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 significant difference in the performance of the algorithm. See Table IV.

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 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 a model/trust region strategy to pick Δx_k. The step is of the form

$$\Delta x_k = -(\lambda_k D_k^2 + H_k)^{-1} \nabla f(x_k), \tag{5.1}$$

where H_k is the current Hessian approximation, D_k is a diagonal scaling matrix and $\lambda_k \geq 0$ is chosen by the safeguarded Reinsch iteration as in [Moré, 1978], with the case of near singularity in $\lambda_k D_k^2 + H_k$ handled as in [Gay, 1979]. 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. This scheme often works well, but it can have 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 $V(0)$, the maximum length allowed for the very first step attempted.

Figure 1 will perhaps be helpful at this point. The ellipses represent the contours of q_k and the circle is the trust region -- our picture assumes the diagonal scaling matrix D_k to be the identity. The point N_k is the "Newton step" or global minimum of the convex quadratic model q_k, and the curve $s(r)$ represents the locus of minimizers of $q_k(x_k + s)$ constrained by $||s||_2 \leq r$, $0 < r < \infty$. Complete details, based largely on [Moré, 1978], can be found in [Gay, 1979], but we choose $\Delta x_k = s(r)$ so that $||D_k \Delta x_k||_2$ lies between 0.9 and 1.1 of the current trust radius. (The actual choice of D_k is discussed in Section 7.)

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_{k+1}^S and q_{k+1}^G for use in determining x_{k+2}. Our decision rule is rather

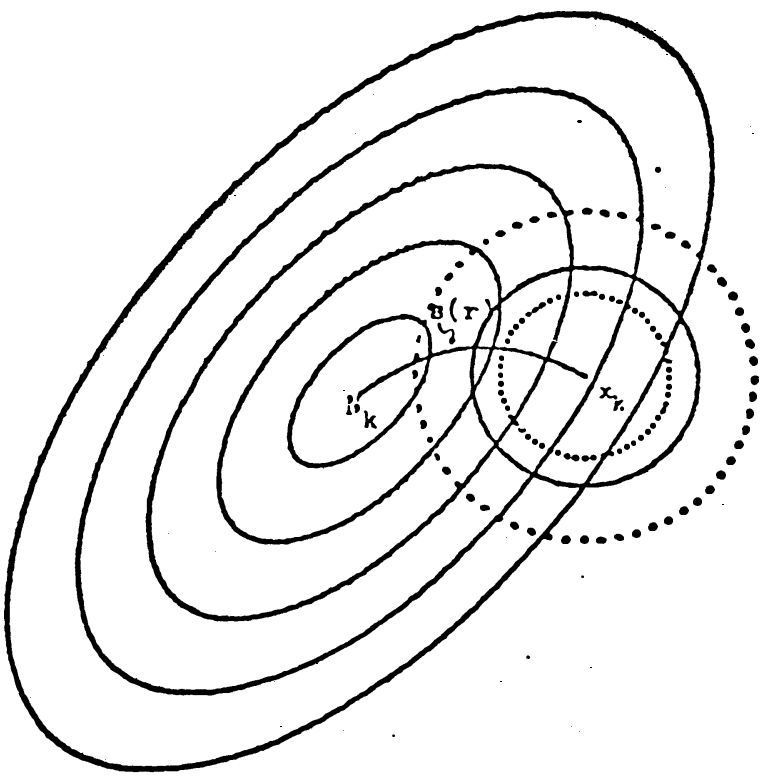

Figure 1

straightforward. Since $S_0 = 0$, we begin using the Gauss-Newton model. After making a prospective step based on the currently preferred model q_k^1 to obtain, say, x_{k+1}^1, we compute R_{k+1}^1 and f_{k+1}^1. If $f_{k+1}^1 > f_k$ then x_{k+1}^1 is discarded, but first the other model q_k^2 is evaluated at x_{k+1}^1 to see how well it agrees with f_{k+1}^1. If there is not sufficient agreement between f_{k+1}^1 and q_k^2 (i.e., if

$$|q_k^1(x_{k+1}^1) - f_{k+1}^1| \leq 2^{-\frac{1}{2}}|q_k^2(x_{k+1}^1) - f_{k+1}^1|), \tag{5.2}$$

then we keep the original model preference, shrink the trust region, and try again. We shrink the trust region radius by the factor suggested by Fletcher [1971] and described by Moré [1978, p. 109]. If the agreement between f_{k+1}^1 and $q_k^2(x_{k+1}^1)$ is sufficiently good, then we change our model preference to q_k^2 and compute x_{k+1}^2 using the same trust region. If x_{k+1}^2 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}.

If, say x_{k+1}^1 yields an acceptable function decrease but

$$f(x_{k+1}^1) - f(x_k) \leq \max\{10\cdot[q_k^1(x_{k+1}^1) - f(x_k)], \; 0.75\cdot\nabla f(x_k)^T(x_{k+1}^1 - x_k)\} \tag{5.3}$$

and $\Delta x_k = x_{k+1}^1 - x_k$ was computed by (5.1) with $\lambda_k > 0$, then we deem it worthwhile to try recomputing x_{k+1}^1 with a larger trust region radius before accepting the step. Hence we double the radius and obtain, say, $x_{k+1}^{1'}$. If $f(x_{k+1}^{1'}) < f(x_{k+1}^1)$, then $x_{k+1}^{1'}$ replaces x_{k+1}^1 and we again check whether to double the radius. Otherwise we discard $x_{k+1}^{1'}$ and accept x_{k+1}^1 as x_{k+1}.

When an acceptable x_{k+1} is found, $q_k^1(x_{k+1})$ and $q_k^2(x_{k+1})$ are compared to f_{k+1}. We have found that it is best to retain the currently preferred model if (5.2) holds with $x_{k+1}^1 = x_{k+1}$, i.e., unless the other model does a significantly better job of predicting the new function value.

Once x_{k+1} has been found, we decide what trust region radius to use first when seeking x_{k+2}. The radius chosen has the form $\mu||D_{k+1}\Delta x_k||_2$, where $\Delta x_k = x_{k+1} - x_k$. If $f(x_{k+1}) - f(x_k) \geq 0.1\cdot[q_k^1(x_{k+1}) - f(x_k)]$, then μ is Fletcher's [1971] decrease factor; if either (5.3) holds with $x_{k+1}^1 = x_{k+1}$, or $||\nabla^2 q_k^1 \Delta x_k - [\nabla f(x_{k+1}) - \nabla f(x_k)]||_2 \leq 0.5||\nabla f(x_{k+1})||$, or $\Delta x_k^T \nabla f(x_{k+1}) < 0.75\cdot\Delta x_k^T \nabla f(x_k)$, then $\mu = 2$; otherwise $\mu = 1$. This rule for updating the radius is a modification of one described by Powell [1970].

6. Convergence Criteria and Covariance.

An important, sometimes difficult issue in practical computing is the matter of deciding when to stop an iterative procedure. We have chosen to include four convergence criteria in NL2SOL: tests for "cosine convergence", "variability convergence", "residual convergence", and "X-convergence".

At any critical point of the sum of squares function (2.1), such as the desired minimizer x^*, the residual vector R is orthogonal to all columns of the Jacobian matrix J. Moreover, the angles between R and the columns of J are independent of the scale of R and columns of J, so it is reasonable to use a test based on these angles [Dennis, 1977]. Hence, we define

$$\text{(6.1)}\quad \text{COSMAX}(x) = \begin{cases} 0 \text{ if } R(x) = 0 \\ \max\{|J_{\cdot,i}(x)^T R(x)|/(||J_{\cdot,i}(x)||_2 ||R(x)||_2) \\ \qquad : ||J_{\cdot,i}(x)|| > \epsilon_{Ji},\ 1 \leq i \leq p\} \\ \text{otherwise,} \end{cases}$$

where $J_{\cdot,i}(x)$ is the $i^{\underline{th}}$ column of $J(x)$, and we detect cosine convergence at any iterate x_k for which $COSMAX(x_k)$ is less than or equal to the cosine convergence tolerance V(CCONCR). By default, $\epsilon_{Ji} = 10^{-9}$, so the COSMAX is scale invariant over a wide range of problems.

For statistical data analysis a different type of convergence criterion is often appropriate. Since there is inherent variability in the data, it is generally not useful to continue iterating when a candidate step $s=(s^1,\ldots,s^p)= \Delta x_k$ is generated for which

$$\max_i \{|s^i|/s.e.(x_k^i)\} \tag{6.2}$$

is sufficiently small. Here $s.e.(x_k^i)$ denotes some estimate of the standard error (square root of the variance) of the $i^{\underline{th}}$ component of the current parameter estimate x_k and so is a function of the statistical variability in the data.

An alternative to (6.2) suggested by Pratt [1977] is to consider general linear combinations $\ell^T s$ of the components of s, i.e.

$$\max\{|\ell^T s|/(\ell^T V_k \ell)^{\frac{1}{2}} : \ell \neq 0\} = (s^T V_k^{-1} s)^{\frac{1}{2}}, \tag{6.3}$$

where V_k is a current estimate of the covariance matrix. For $s.e.(x_k^i) = (e_i^T V_k e_i)^{\frac{1}{2}}$, where e_i is the $i^{\underline{th}}$ standard unit vector, (6.3) clearly dominates (6.2), so we have chosen to include a test based on (6.3).

Our choice for V_k was $\hat{\sigma}_k^2 H_k^{-1}$, where $\hat{\sigma}_k^2$ is the current residual sum of squares divided by $\max\{1,n-p\}$, i.e.

$$\hat{\sigma}_k^2 = 2f(x_k)/\max(1,n-p), \tag{6.4}$$

and H_k is the current Hessian approximation, i.e. $J^TJ(x_k)$ for the Gauss-Newton model and $J^TJ(x_k) + S_k$ for the augmented model. Whenever a candidate

step s is generated for which H_k is positive definite, we thus compute

$$\text{VARIAB}(s) = s^T H_k s / \sigma_k^2, \tag{6.5}$$

and we detect variability convergence if VARIAB(s) does not exceed the variability convergence tolerance V(VCONCR).

For full Newton steps, i.e. $s = s^N = -H_k^{-1} \nabla f(x_k)$, (6.5) gives a quantity closely related to the relative reduction, PRED_k, in $f(x_k)$ that is still possible according to the current model. Specifically, (6.5) and (6.4) imply

$$\begin{aligned} \text{PRED}_k &= [-\nabla f(x_k)^T s^N - \tfrac{1}{2} s^{NT} H_k s^N] / f(x_k) \\ &= \tfrac{1}{2} s^{N^T} H_k s^N / f(x_k) \\ &= \text{VARIAB}(s^N) / \max\{1, n-p\}. \end{aligned}$$

Thus at least for full Newton steps (the steps usually taken near x^*), the variability convergence test checks whether the predicted relative reduction still possible in the residual sum of squares is small.

Zero-residual problems, those for which $R(x^*) = 0$, require special consideration. Indeed, it can be shown that if $J(x^*)$ is nonsingular, then $\liminf_{||x-x^*|| \to 0} \text{COSMAX}(x) \geq 1/[p^{\frac{1}{2}} \text{cond}(J(x^*))] > 0$, where cond(J) is the condition number of J (ratio of largest to smallest singular value). Moreover, for $s = s(x)$ the Newton step from the Gauss-Newton model, i.e. $s(x) = -(J^T J)^{-1} J^T R$, it is easily seen that $\lim_{x \to x^*} \text{VARIAB}(s(x)) = \max\{1, n-p\}$. To handle this case, NL2SOL detects residual convergence if $||R(x_k)|| \leq$ V(RCONCR). We regret that this convergence test must be sensitive to the scaling of R.

It is easy to specify convergence tolerances too strict for the precision of the arithmetic being used. We have therefore included a fourth convergence test, the X-convergence test, which often works when overly tight tolerances have been given for the other tests. This test is satisfied whenever a step s is generated that yields a much smaller function decrease than expected (i.e. $f(x_k+s) \geq f(x_k) + s^T \nabla f(x_k)/10$) and the relative change that s causes in x_k is small, i.e. $RELDX(x_k, x_k+s) \leq V(XCONCR)$, where

$$RELDX(y,z) = \max_i |z_i - y_i| / [|z_i| + |y_i|].$$

Many statistical inference procedures require an estimate of the covariance matrix at the solution x^*. NL2SOL provides three possibilities:

(6.6) $$\hat{\sigma}^2 H^{-1} J^T J H^{-1}$$

(6.7) $$\hat{\sigma}^2 H^{-1}$$

(6.8) $$\hat{\sigma}^2 (J^T J)^{-1}$$

where $\hat{\sigma}^2$ is given by (6.4) with $x_k = x^*$. When (6.6) or (6.7) is specified, a symmetric finite difference Hessian approximation H is obtained at the solution, x^*. If H is positive definite [or J is non-singular at x^* for (6.8)], the specified covariance matrix is computed.

A detailed discussion of all three covariance forms is contained in [Bard, 1974]. The second form (6.7) is based on asymptotic maximum likelihood theory and is perhaps the most common form of estimated covariance matrix. We feel that (6.6), the default, is more useful for smaller sample sizes and in other cases where the conditions necessary [Rao, 1965] for the asymptotic theory may be violated. The third form assumes that the residuals at the solution are small and is therefore often highly suspect.

7. Test Results

We have run NS2SOL 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] and on one described in [Meyer, 1970]. The original sources for these problems, together with the abbreviated problem names used in Tables II-IV and some notes, are given in Table I.

Table I

Original Sources of Test Problems

Problem	Note	Source
ROSNBROK		[Rosenbrock, 1960]
HELIX	1	[Fletcher & Powell, 1963]
SINGULAR		[Powell, 1962]
WOODS		[Colville, 1968]
ZANGWILL	2	[Zangwill, 1967]
ENGVALL		[Engvall, 1966]
BRANIN		[Branin, 1971]
BEALE		[Beale, 1958]
CRAGG	3	[Gill & Murray, 1976]
BOX		[Box, 1966]
DAVIDON1	4	[Davidon, 1976]
FRDSTEIN	5	[Freudenstein & Roth, 1963]
WATSON6,9,12,20		[Kowalik & Osborne, 1968]
CHEBQD8		[Fletcher, 1965]
BROWN	6	[Brown & Dennis, 1971]
BARD		[Bard, 1970]
JENNRICH		[Jennrich & Sampson, 1968]
KOWALIK		[Kowalik & Osborne, 1968]
OSBORNE1,2		[Osborne, 1972]
MEYER		[Meyer, 1970]

Notes on Table I

Note 1: The residual vector $R(x)$ for this problem is a discontinuous function of x. On those runs where NL2SOL halts with X-convergence, the iterates have converged to a point of discontinuity.

Note 2: This is a linear least-squares problem which NL2SOL solves in one step when the limit V(LMAXO) on the length of the first step is increased slightly from its default value.

Note 3: The original Miele problem described in [Cragg & Levy, 1969], which Gill and Murray [1976] cite as the source for this problem, does not have the last residual component $r_5(x) = x_4 - 1$. This new component forces x_4 to move more rapidly towards 1, but otherwise causes no noteworthy change in the performance given by NL2SOL.

Note 4: This is another linear least-squares problem, one that is so ill conditioned that NL2SOL needs two steps to solve it when using double precision on an IBM 370 computer with V(LMAXO) set large. With a double precision of a few bits more accuracy, such as that of the MULTICS (i.e, Honeywell) machine or the Univac 1110, a single step suffices (for large V(LMAXO)).

Note 5: In all our test runs, NL2SOL found a local solution to this problem. The residual vector vanishes at the global solution,

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

The behavior of NL2SOL is determined in part by an integer array IV and a floating-point array V, which contain iteration and function evaluation limits, convergence tolerances, and other switches and constants. In the runs summarized in Tables II-IV, most of the IV and V input components had the default values given them by subroutine DFAULT. Exceptions included the following: variability convergence testing was turned off by setting V(VCONCR) = 0; and on problem MEYER, the iteration and function-evaluation limits were increased.

Table II summarizes the performance of NL2SOL on the test problem set when all IV and V input components have their default values, with the exceptions just noted. Following a suggestion of J.J. Moré [1979], we obtained new starting guesses for many of the test problems by multiplying the standard starting guess by ten and one hundred. The column labelled LS gives the base 10 logarithm of the factor by which the standard starting guess was multiplied. The problem dimensions appear in the columns headed N and P, while the number of function (i.e. R(x)) and gradient (i.e. J(x)) evaluations performed respectively appear under NF and NG. The column labelled F gives the final function value (half the sum of squares of R(x)), while the one labeled COSMAX gives COSMAX(x), computed from (6.1) (with $\epsilon_{Ji} = 10^{-9}$), at the final x. Under C is a code telling why NL2SOL stopped: R means

residual convergence, i.e., $||R(x)||_2 \leq 10^{-9}$; C means cosine convergence, i.e, $COSMAX(x) \leq 10^{-7}$; X means X convergence (see §6) with $V(XCONCR) = 2.22 \times 10^{-13}$; and F means function evaluation limit reached without convergence. The results reported in Tables II-IV were obtained on the IBM 370/168 computer at the Massachusetts Institute of Technology, and the convergence tolerances just mentioned are the defaults for this machine, which has a unit roundoff of $16^{-13} \doteq 2.22 \times 10^{-16}$ in double precision, the precision used.

The choice of scale matrices D_k mentioned in §5 can significantly affect the performance of NL2SOL. By default, $D_k = \mathrm{diag}(d_1^k,\ldots,d_p^k)$ is updated by the rule

$$d_i^k = \max\{[\,||J_{\cdot,i}||_2^2 + \max\{0,S_{ii}\}\,]^{\frac{1}{2}}, 0.6d_i^{k-1}, 10^{-3}\};$$

at the start of each iteration, beginning with $d_i^{-1} = 0$, where $J_{\cdot,i}$ denotes the i<u>th</u> column of the current Jacobian matrix $J(x_k)$. (The factor 0.6 is actually V(DFAC). We experimented with several values of V(DFAC), including zero, 0.5, 0.75, and one, and we felt that 0.6 gave the best overall performance of the values tried.) The advantage of this choice of D_k is that it is largely scale-invariant.

A choice of D_k which is not at all scale-invariant, but which gives better performance on many of our test problems, is $D_k = I$, the identity matrix. Table III shows how these two choices of D_k compare: results from Table II are repeated in the columns headed DEFAULT, while results corresponding to $D_k = I$ appear under D = I.

Table III also gives results from two other test runs. Those under V(LMAX0) = 10**10 show what happens when the bound V(LMAX0) on the 2-norm of the very first step attempted is increased from its default value of 100 to 10^{10}, while those under V(CCONCR) = 10**-8 show what happens when the cosine convergence tolerance is decreased from 10^{-7} to 10^{-8}. In both of these test runs the default D_k was used.

Table II

Default NLSOL Test Summary

PROBLEM	LS	N	P	NF	NG	C	F	COSMAX
ROSNBROK	0	2	2	15	13	R	0.973E-32	0.999E+00
ROSNBROK	1	2	2	45	36	R	0.973E-32	0.999E+00
ROSNBROK	2	2	2	156	135	R	0.973E-32	0.999E+00
HELIX	0	3	3	15	13	R	0.323E-18	0.100E+01
HELIX	1	3	3	12	10	R	0.221E-23	0.777E+00
HELIX	2	3	3	125	37	X	0.864E+04	0.744E+00
SINGULAR	0	4	4	18	18	R	0.273E-18	0.962E-04
SINGULAR	1	4	4	22	22	R	0.806E-19	0.551E-04
SINGULAR	2	4	4	30	26	R	0.849E-19	0.522E-04
WOODS	0	7	4	56	43	R	0.390E-21	0.707E+00
WOODS	1	7	4	64	44	R	0.143E-24	0.772E+00
WOODS	2	7	4	75	51	R	0.523E-23	0.721E+00
ZANGWILL	0	3	3	3	3	R	0.147E-27	0.896E+00
ENGVALL	0	5	3	16	13	R	0.124E-24	0.999E+00
ENGVALL	1	5	3	20	19	R	0.525E-22	0.999E+00
ENGVALL	2	5	3	27	26	R	0.882E-19	0.999E+00
BRANIN	0	2	2	2	2	R	0.162E-28	0.945E+00
BRANIN	1	2	2	16	15	R	0.100E-27	0.869E+00
BRANIN	2	2	2	15	12	R	0.864E-32	0.869E+00
BEALE	0	3	2	10	9	R	0.893E-26	0.542E+00
BEALE	1	3	2	6	6	R	0.148E-21	0.997E+00
CRAGG	0	5	4	22	21	R	0.289E-18	0.197E+00
CRAGG	1	5	4	75	43	C	0.592E+02	0.415E-07
BOX	0	10	3	20	14	R	0.167E-22	0.921E+00
BOX	1	10	3	19	11	C	0.378E-01	0.385E-11
BOX	2	10	3	23	14	C	0.378E-01	0.104E-07
DAVIDON1	0	15	15	9	8	C	0.710E-04	0.910E-07
FRDSTEIN	0	2	2	8	7	C	0.245E+02	0.309E-08
FRDSTEIN	1	2	2	19	12	C	0.245E+02	0.784E-09
FRDSTEIN	2	2	2	35	19	C	0.245E+02	0.504E-11
WATSON6	0	31	6	11	10	C	0.114E-02	0.822E-07
WATSON9	0	31	9	11	9	C	0.700E-06	0.114E-08
WATSON12	0	31	12	13	12	C	0.236E-09	0.902E-08
WATSON20	0	31	20	10	10	C	0.152E-14	0.624E-07
CHEBQD8	0	8	8	27	17	C	0.176E-02	0.263E-07
CHEBQD8	1	8	8	86	60	C	0.176E-02	0.603E-07
BROWN	0	20	4	22	16	C	0.429E+05	0.149E-07
BROWN	1	20	4	23	19	C	0.429E+05	0.486E-07
BROWN	2	20	4	32	26	C	0.429E+05	0.856E-08
BARD	0	15	3	6	6	C	0.411E-02	0.203E-07
BARD	1	15	3	42	28	C	0.871E+01	0.858E-07
BARD	2	15	3	21	9	C	0.871E+01	0.615E-07
JENNRICH	0	10	2	15	12	C	0.622E+02	0.212E-08
KOWALIK	0	11	4	11	10	C	0.154E-03	0.729E-07
KOWALIK	1	11	4	163	90	C	0.514E-03	0.712E-07
KOWALIK	2	11	4	81	61	C	0.154E-03	0.563E-07
OSBORNE1	0	33	5	23	19	C	0.273E-04	0.645E-08
OSBORNE2	0	65	11	17	16	C	0.201E-01	0.183E-07
OSBORNE2	1	65	11	28	12	C	0.895E+00	0.121E-07
MADSEN	0	3	2	11	11	C	0.387E+00	0.975E-07
MADSEN	1	3	2	12	12	C	0.387E+00	0.185E-07
MADSEN	2	3	2	21	20	C	0.387E+00	0.901E-08
MEYER	0	16	3	281	175	C	0.440E+02	0.244E-11

Table III

Some Nondefault Test Runs

PROBLEM	LS	DEFAULT NF	NG	C	D = I NF	NG	C	V(LMAX0) = 10**10 NF	NG	C	V(CCONCR) = 10**-8 NF	NG	C	NOTE
ROSNBROK	0	15	13	R	17	14	R	15	13	R	15	13	R	
ROSNBROK	1	45	36	R	28	23	R	54	39	R	45	36	R	
ROSNBROK	2	156	135	R	54	45	R	178	140	R	156	135	R	
HELIX	0	15	13	R	8	8	R	15	13	R	15	13	R	
HELIX	1	12	10	R	14	11	R	13	11	R	12	10	R	
HELIX	2	125	37	X	17	14	R	17	13	R	125	37	X	
SINGULAR	0	18	18	R	18	18	R	18	18	R	18	18	R	
SINGULAR	1	22	22	R	22	22	R	22	22	R	22	22	R	
SINGULAR	2	30	26	R	28	26	R	25	25	R	30	26	R	
WOODS	0	56	43	R	56	42	R	56	43	R	56	43	R	
WOODS	1	64	44	R	70	46	R	67	50	R	64	44	R	
WOODS	2	75	51	R	61	47	R	67	49	R	75	51	R	
ZANGWILL	0	3	3	R	3	3	R	2	2	R	3	3	R	
ENGVALL	0	16	13	R	16	14	R	16	13	R	16	13	R	
ENGVALL	1	20	19	R	20	18	R	20	19	R	20	19	R	
ENGVALL	2	27	26	R	28	25	C	38	28	R	27	26	R	1
BRANIN	0	2	2	R	2	2	R	2	2	R	2	2	R	
BRANIN	1	16	15	R	17	15	R	16	15	R	16	15	R	
BRANIN	2	15	12	R	18	16	R	22	21	R	15	12	R	
BEALE	0	10	9	R	10	8	R	10	9	R	10	9	R	
BEALE	1	6	6	R	8	8	R	6	6	R	6	6	R	
CRAGG	0	22	21	R	21	20	R	22	21	R	22	21	R	
CRAGG	1	75	43	C	48	43	R	46	39	C	76	44	C	2
BOX	0	20	14	R	6	6	R	6	6	R	20	14	R	
BOX	1	19	11	C	27	15	C	40	21	C	19	11	C	
BOX	2	23	14	C	29	16	C	6	6	C	25	15	C	
DAVIDON1	0	9	8	C	3	3	R	3	3	R	10	9	C	
FRDSTEIN	0	8	7	C	9	9	C	8	7	C	8	7	C	
FRDSTEIN	1	19	12	C	19	14	C	15	14	C	19	12	C	
FRDSTEIN	2	35	19	C	37	24	C	20	18	C	35	19	C	
WATSON6	0	11	10	C	8	8	C	11	10	C	12	11	C	
WATSON9	0	11	9	C	11	9	C	11	9	C	11	9	C	
WATSON12	0	13	12	C	13	11	C	13	12	C	13	12	C	
WATSON20	0	10	10	C	9	9	C	10	10	C	200	96	F	3
CHESQD8	0	27	17	C	20	14	C	27	17	C	28	18	C	
CHESQD8	1	86	60	C	74	63	C	109	67	C	92	61	X	3
BROWN	0	22	16	C	15	14	C	22	16	C	42	17	X	3
BROWN	1	23	19	C	16	16	C	21	17	C	38	20	X	3
BROWN	2	32	26	C	26	24	C	26	23	C	32	26	C	
BARD	0	6	6	C	6	6	C	6	6	C	7	7	C	
BARD	1	42	28	C	47	27	C	33	24	C	54	36	C	
BARD	2	21	9	C	29	17	C	8	7	C	30	15	C	4
JENNRICH	0	15	12	C	16	13	C	15	12	C	15	12	C	
KOWALIK	0	11	10	C	16	12	C	11	10	C	12	11	C	
KOWALIK	1	163	90	C	200	72	F	163	90	C	172	96	C	
KOWALIK	2	81	61	C	103	72	C	79	62	C	82	62	C	
OSBORNE1	0	23	19	C	27	22	C	23	19	C	23	19	C	
OSBORNE2	0	17	16	C	17	14	C	17	16	C	18	17	C	
OSBORNE2	1	28	12	C	24	13	C	28	12	C	30	13	C	
BOGGIN	0	11	11	C	11	11	C	11	11	C	12	12	C	
BOGGIN	1	12	12	C	13	13	C	12	12	C	13	13	C	
BOGGIN	2	21	20	C	22	21	C	22	21	C	21	20	C	
MEYER	0	281	175	C	350	198	F	288	193	C	281	175	C	

Notes on Table III.

Note 1: For problem ENGVALL with LS = 2, a local minimizer x^* having $f(x^*) = 56.1$ was found in the D = I run.

Note 2: For problem CRAGG with LS = 1, a different local minimizer x^*, one having $f(x^*) = 249$, was found in the V(LMAX0) = 10.10 run than in the DEFAULT run.

Note 3: For problems WATSON20, CHEBQD8, and BROWN, a cosine convergence tolerance of 10^{-8} appears too tight for the double-precision arithmetic of an IBM 370 computer. X-convergence did not occur on WATSON20 because one of the x components hovered about zero. The run with V(CCOSCR) = 10.-8 achieved $f(x) = 6.46 \times 10^{-18}$ on this problem (and had $f(x) = 6.53 \times 10^{-18}$ after 20 function and 16 gradient evaluations).

Note 4: For problem BARD with LS = 2, the D = I run found the solution obtained in the DEFAULT run with LS = 0.

Table IV summarizes test runs with three variants of NL2SOL, all of which used the default choice of D_k and the same IV and V inputs as were used for Table II. The columns headed PURE GN show what happens if the augmented model is never used, while those headed PURE S show what happens if it is always used (after the first iteration). Finally, the columns headed NO SIZING give the results obtained when adaptive modelling is allowed but no sizing is performed. We feel that Table IV makes a good case for the use of adaptive modelling with sizing in NL2SOL.

Table IV

Variations on NL230L

PROBLEM	LS	DEFAULT NF	DEFAULT NG	DEFAULT C	PURE GN NF	PURE GN NG	PURE GN C	PURE S NF	PURE S NG	PURE S C	NO SIZING NF	NO SIZING NG	NO SIZING C	NOTE
ROSNBROK	0	15	13	R	20	16	R	24	21	R	20	16	R	
ROSNBROK	1	45	36	R	38	33	R	65	55	R	45	35	R	
ROSNBROK	2	156	135	R	113	103	R	200	45	F	122	111	R	
HELIX	0	15	13	R	10	10	R	22	19	R	10	10	R	
HELIX	1	12	10	R	12	11	R	25	16	R	13	10	R	
HELIX	2	125	37	X	16	14	R	38	23	R	140	39	X	
SINGULAR	0	18	18	R	18	18	R	29	29	R	18	18	R	
SINGULAR	1	22	22	R	22	22	R	35	35	R	22	22	R	
SINGULAR	2	30	26	R	30	26	R	43	42	R	30	26	R	
WOODS	0	56	43	R	77	67	R	47	34	R	80	50	R	
WOODS	1	64	44	R	80	65	R	47	40	R	116	67	R	
WOODS	2	75	51	R	74	55	R	62	47	R	80	58	R	
ZANGWILL	0	3	3	R	3	3	R	3	3	R	3	3	R	
ENGVALL	0	16	13	R	15	13	R	19	17	R	16	13	R	
ENGVALL	1	20	19	R	14	14	R	25	22	R	18	17	R	
ENGVALL	2	27	26	R	28	27	R	38	36	R	27	26	R	
BRANIN	0	2	2	R	2	2	R	2	2	R	2	2	R	
BRANIN	1	16	15	R	16	15	R	24	23	R	17	15	R	
BRANIN	2	15	12	R	15	12	R	38	35	R	15	12	R	
BEALE	0	10	9	R	10	9	R	19	14	R	10	9	R	
BEALE	1	6	6	R	6	6	R	14	13	R	6	6	R	
CRAGG	0	22	21	R	21	20	R	38	35	R	22	21	R	1
CRAGG	1	75	43	C	153	100	C	200	120	C	179	86	C	
BOX	0	20	14	R	22	16	R	46	26	R	19	14	R	
BOX	1	19	11	C	20	12	C	61	52	R	20	12	C	
BOX	2	23	14	C	25	16	C	38	24	C	25	16	C	
DAVIDON1	0	9	8	C	9	8	C	9	8	C	9	8	C	
FRDSTEIN	0	8	7	C	26	15	C	8	8	C	8	7	C	
FRDSTEIN	1	19	12	C	37	22	C	24	18	C	20	13	C	
FRDSTEIN	2	35	19	C	46	22	C	44	29	C	36	20	C	
WATSON6	0	11	10	C	13	12	C	15	11	C	12	11	C	
WATSON9	0	11	9	C	11	9	C	21	14	C	12	10	C	
WATSON12	0	13	12	C	13	12	C	21	17	C	14	11	C	
WATSON20	0	10	10	C	10	10	C	16	15	C	10	10	C	
CHEBQD8	0	27	17	C	43	29	C	22	18	C	25	16	C	
CHEBQD8	1	86	60	C	118	78	C	131	90	C	147	98	C	
BROWN	0	22	16	C	128	84	C	19	17	C	30	21	C	
BROWN	1	23	19	C	153	89	C	24	23	C	171	86	C	
BROWN	2	32	26	C	82	59	C	33	28	C	200	91	F	
BARD	0	6	6	C	6	6	C	10	10	C	6	6	C	
BARD	1	42	28	C	42	28	C	76	34	C	43	29	C	2
BARD	2	21	9	C	21	9	C	83	26	C	21	9	C	2
JENNRICH	0	15	12	C	22	12	C	13	12	C	17	13	C	
KOWALIK	0	11	10	C	26	25	C	20	15	C	12	11	C	
KOWALIK	1	163	90	C	108	78	C	200	63	F	99	73	C	
KOWALIK	2	81	61	C	109	85	C	200	76	F	184	151	I	
OSBORNE1	0	23	19	C	17	15	C	34	31	C	17	15	C	
OSBORNE2	0	17	16	C	17	15	C	16	15	C	17	16	C	
OSBORNE2	1	28	12	C	14	12	C	18	10	C	30	14	C	
MADSEN	0	11	11	C	45	45	C	12	12	C	13	13	C	
MADSEN	1	12	12	C	42	42	C	16	16	C	14	14	C	
MADSEN	2	21	20	C	56	55	C	23	23	C	23	21	C	
MEYER	0	281	175	C	282	183	C	321	183	C	189	138	C	

Notes on Table IV

Note 1: Each of the runs listed for problem CRAGG with LS = 1 found a different local minimizer x^*. The DEFAULT run found $f(x^*) = 59.2$; the PURE GN run found $f(x^*) = 9.98 \times 10^4$; the PURE S run found $f(x^*) = 4.37$; and the NO SIZING run found $f(x^*) = 1.11 \times 10^4$.

Note 2: While the other runs of problem BARD found the same local minimizer as the corresponding DEFAULT run, the PURE S runs gave different results. For LS = 1, the PURE S run found $f(x^*) = 4.11 \times 10^{-3}$ (as did the DEFAULT run with LS = 0), and for LS = 2, it found $f(x^*) = 8.45$.

REFERENCES

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

BARD, Y. (1974), Nonlinear Parameter Estimation, Academic Press, New York.

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.

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 nonlinear least squares problem. in Numerical Solutions 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 MORÉ, J.J. (1977), Quasi-Newton methods, motivation and theory, SIAM Rev. 19, pp. 46-89.

DENNIS, J.E. and WELSCH, R.E. (1978), Techniques for nonlinear least squares and robust regression. Comm. Statist. B7, pp. 345-359.

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. (1971), A modified Marquardt subroutine for nonlinear least squares, A.E.R.E. Harwell report R6799.

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 equation J. Assoc. Comput. Mach. 10, pp. 550-556.

GAY, D. (1979), Computing optimal locally constrained steps, in preparation

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 non-linear least-squares problem. NPL Report NAC 71.

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.

JENNRICH, R.I. and SAMPSON, P.F. (1968), Application of step-wise 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.

MEYER, R.R. (1970), Theoretical and computational aspects of nonlinear regressi in Nonlinear Programming edited by J.B. Rosen, O.L. Mangasarian, and K. Ri Academic Press, New York.

MORÉ, J.J. (1978), The Levenberg-Marquardt algorithm: implementation and theory in Lecture Notes in Mathematics No. 630 Numerical Analysis edited by G. Wa Springer-Verlag, Berlin, Heidelberg and New York.

MORÉ, J.J. (1979), Implementation and Testing of Optimization Software, LAMP Report 79/NA4 , University of Cambridge, UK.

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 FORTRAN subroutine for unconstrained minimization, requiring first derivatives of the objective function, Report AERE-R.6469, A.E.R.E. Harwell, Oxfordshire, England.

PRATT, J. W. (1977), When to stop a quasi-Newton search for a maximum likelihood estimate. Working paper 77-16, Harvard Business School, Soldiers Field Rd., Boston Mass.

RAO, C.R. (1965), Linear Statistical Inference and Its Applications, John Wiley & Sons, 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 separable non-linear least squares problems. Inst. för tillämpad 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 tillämpad matematik, Box 5073 Stockholm 5, ITM Arbetsrapport nr. 24.

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

on,

A P P E N D I X.

NL2SOL Usage Summary

0. Contents

1. Purpose
2. Method
3. Calling Sequence
4. Example
5. Return Codes
6. IV Values
7. V Values of Primary Interest
8. Finite-Difference Jacobians -- NL2SNO
9. Restarting
10. Scaling
11. LMAX0: Limiting the First Step Length
12. Local Solutions
13. Printed Output
14. Changing Computers
15. Using Reverse Communication -- NL2ITR
16. STOPX
17. Other V Input Values
18. Storage Requirements
19. References
20. Acknowledgement

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 $F(x) = \frac{1}{2}\sum_{i=1}^{n} R_i(x)^2$.

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 secant (quasi-Newton) updating method. Once the iterates come sufficiently close to a (local) solution, they usually converge quite rapidly. To promote convergence from poor starting guesses, NL2SOL uses a model/trust-region technique along with an adaptive choice of the model Hessian. Consequently, the algorithm sometimes reduces to a Gauss-Newton or 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, CALCR, CALCJ, IV, V, UIPARM, URPARM, UFPARM)
```

Note: In the double-precision version of NL2SOL, all quantities 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^*. When NL2SOL returns after converging or reaching the iteration limit (i.e., returns with IV(1) = 3, 4, 5, 6, or 8), X contains the best parameter estimate found.

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

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

When CALCR is called, NF is the invocation count for CALCR; it is included for possible use with CALCJ. If X is out of bounds (e.g. if R(X) would overflow), then CALCR should set NF to 0, which will cause a shorter step to be attempted. CALCR should not change N, P or X and should be declared EXTERNAL in the calling program. R should be declared REAL R(N) .

CALCJ (input subroutine) computes the Jacobian matrix J = J(X) of first partials, $J_{ij} = \frac{\partial R_i}{\partial x_j}(X)$, when invoked by:

```
CALL CALCJ(N, P, X, NF, J, UIPARM, URPARM, UFPARM)
```

When CALCJ is called, NF is the invocation count for CALCR at the time when R(X) was evaluated. Except when J is restored after a covariance matrix has been computed with IV(COVREQ) = 1 or 2 (see §6), the X passed to CALCJ is the one passed to CALCR on either its most recent invocation or the one prior to it. Thus if CALCR saves intermediate results for use by CALCJ, then it is possible to tell from NF whether they are valid for the current X (or which copy is valid if two are kept). If J cannot be computed at X, then CALCJ should set NF to 0. CALCJ should not change N, P, or X and should be declared EXTERNAL in the calling program. J should be declared REAL J(N,P) .

IV (I/O INTEGER array of length P + 60) on input contains certain values (such as limits on the number of iterations and calls on CALCR) that control the behavior of NL2SOL and on output contains various counts and other items of interest: see §§5 and 6. If IV(1) = 0 on input, then default values are supplied for the input components of both IV and V. The caller may supply nondefault values for selected components of IV and V by CALLing DFAULT(IV, V) and then assigning the appropriate nondefault values before calling NL2SOL.

V (I/O REAL array of length 96 + N*(P+3) + P*(7P + 43)/2) on input contains certain values (such as convergence tolerances) that

control the behavior of NL2SOL and on output contains various items of interest (such as F(X) and R(X)): see §§7 and 17.

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

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

UFPARM (subroutine), like UIPARM, is passed without change to CALCR and CALCJ. If there is no need for such a subroutine, then on many systems it suffices to pass an arbitrary variable or constant for UFPARM. But if an actual subroutine is passed, then it must be declared EXTERNAL in the calling program.

4. Example

Let $n = 3$, $p = 2$, and $R(x) = \begin{pmatrix} x_1^2 + x_2^2 + x_1 x_2 \\ \sin x_1 \\ \cos x_2 \end{pmatrix}$. (This problem is due to Madsen, Reference 3.) 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 version of NL2SOL.

```
      INTEGER IV(62)
      REAL V(168), X(2)
      EXTERNAL MADR, MADJ
      X(1) = 3.0
      X(2) = 1.0
      IV(1) = 0
      CALL NL2SOL(3, 2, X, MADR, MADJ, IV, V, 0, 0., MADR)
      STOP
      END
      SUBROUTINE MADR(N, P, X, NF, R, UIPARM, URPARM, UFPARM)
      INTEGER N, P, NF, 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(N, P, X, NF, J, UIPARM, URPARM, UFPARM)
      INTEGER N, P, NF, UIPARM(1)
      REAL X(P), J(N,P), URPARM(1)
      EXTERNAL UFPARM
      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 UIPARM and URPARM and MADR is passed for UFPARM.

When the above is executed, NL2SOL prints the initial X vector, a summary of the iterations performed, the final X vector, and some statistics (including the final F(X) and a covariance matrix). If REAL is changed to DOUBLE PRECISION and the above is run on an IBM 370 computer, then NL2SOL reports variability convergence (IV(1) = 6 -- see §5) after 7 calls on CALCR and CALCJ and returns X(1) = -0.156234, X(2) = 0.698698, and F(X) = 0.386616 .

In this example, it is possible to obtain a slightly smaller value of F(X) by decreasing the variability convergence tolerance from its default value of 10^{-4}. If the statement IV(1) = 0 in the main program is replaced by

```
      CALL DFAULT(IV, V)
      V(42) = 0.0
```

then variability convergence testing is turned off. When this modified version of the example is run on an IBM 370 with REAL changed to DOUBLE PRECISION, NL2SOL reports cosine convergence (IV(1) = 4) after 11 calls on CALCR and CALCJ and returns X(1) = -0.155437, X(2) = 0.694564, and F(X) = 0.386600 .

5. Return Codes

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

- 3 = X convergence: see V(XCONCR) in §7.
- 4 = cosine convergence: see V(CCONCR) in §7.
- 5 = residual convergence: see V(RCONCR) in §7.
- 6 = variability convergence: see V(VCONCR) in §7.
- 7 = function evaluation limit reached: see IV(MXFCAL) in §6.
- 8 = iteration limit reached: see IV(MXITER) in §6.
- 9 = STOPX returned .TRUE. (external interupt): see §16.
- 11 = F(X) overflows at the initial X.
- 12 = bad parameters passed to ASSESS (which should not occur).
- 13 = J(X) could not be computed (i.e., CALCJ set NF to 0).
- 14 = one of the inequalities $NN \geq N \geq P \geq 1$ is violated. (NN is only of interest to those who exercise the reverse communication option -- see §15.)
- 15 = NL2SOL was restarted (see §9) with NN, N, or P changed.
- 16 = IV(INITS) is out of range: see §6.
- 17 = IV(1) was out of range (i.e., was negative or greater than 10) when NL2SOL was called.
- 18 or more = V(IV(1)) is out of range: see §§7 and 17.

Just before NL2SOL returns, a brief description of the return code

is printed (unless all printing is turned off by IV(PRUNIT) = 0).

6. IV Values

IV Input Values (Supplied by DFAULT)

IV(1)..... IV(1) should have a value between 0 and 10 when NL2SOL is called. 0 and 10 both mean that this is a fresh start; 0 means DFAULT(IV, V) should be invoked to supply default values to the input components of IV and V, while 10 means that the caller has already supplied these values. IV(1) input values between 3 and 9 mean that NL2SOL should restart: see §9. Default = 10.

IV(COVPRT)... IV(14) = 1 means print a covariance matrix at the solution. This matrix is computed as IV(COVREQ) dictates just before a return with IV(1) = 3, 4, 5, or 6. IV(COVPRT) = 0 means do not print a covariance matrix. Default = 1.

IV(COVREQ)... IV(15) ≠ 0 means compute a covariance matrix just before a return with IV(1) = 3, 4, 5, or 6. In this case, an approximate covariance matrix is obtained in one of several ways. Let $k = |IV(COVREQ)|$ and let $\sigma = 2F(X)/\max\{1,N-P\}$, where $2F(X)$ is the residual sum of squares. If $k = 1$ or 2, then a finite-difference Hessian approximation H is obtained. If H is positive-definite (or, for $k = 3$, if the Jacobian matrix $J = J(X)$ is nonsingular), then one of the following is computed:

$$k = 1 \longrightarrow \sigma \cdot H^{-1}(J^TJ)H^{-1}$$

$$k = 2 \longrightarrow \sigma \cdot H^{-1}$$

$$k = 3 \longrightarrow \sigma \cdot (J^TJ)^{-1} .$$

If IV(COVREQ) > 0, then both function and gradient values (calls on CALCR and CALCJ) are used in computing H (with step sizes determined by V(DELTAO) -- see §7), while if IV(COVREQ) < 0, then only function values (calls on CALCR) are used (with step sizes determined by V(DLTFDC)). If IV(COVREQ) = 0, then no attempt is made to compute a covariance matrix (unless IV(COVPRT) = 1, in which case NL2SOL assumes IV(COVREQ) = 1 and NL2SNO assumes IV(COVREQ) = -1). Default = 1.

IV(INITS).... IV(16) tells how the S matrix of Ref. 1 should be initialized: 0 means set S to 0 and start with the Gauss-Newton model; 1 and 2 mean that the caller has supplied the initial S, storing its lower triangle row-wise in V starting at V(P + 87); IV(INITS) = 1 means start with the Gauss-Newton model, while IV(INITS) = 2 means start with the augmented model. Default = 0.

IV(MXFCAL)... IV(17) gives the maximum number of function evaluations (calls on CALCR, excluding those used to compute the covariance matrix) allowed. If this number does not suffice, then NL2SOL returns with IV(1) = 7. Default = 200.

IV(MXITER)... IV(18) gives the maximum number of iterations allowed. It also indirectly limits the number of gradient evaluations (calls on

CALCJ, excluding those used to compute the covariance matrix) to IV(MXITER) + 1. If IV(MXITER) iterations do not suffice, then NL2SOL returns with IV(1) = 8. Default = 150.

IV(OUTLEV)... IV(19) controls the number and length of iteration summary lines printed. IV(OUTLEV) = 0 means do not print any summary lines. Otherwise, print a summary line after each |IV(OUTLEV)| iterations. Long summary lines are printed if IV(OUTLEV) > 0, short lines if IV(OUTLEV) < 0. See §13 for more details. Default = 1.

IV(PARPRT)... IV(20) = 1 means print any nondefault V values on a fresh start or any changed V (input) values on a restart. IV(PARPRT) = 0 means skip this printing. Default = 1.

IV(PRUNIT)... IV(21) is the output unit number on which all printing is done. IV(PRUNIT) = 0 means suppress all printing. (Setting IV(PRUNIT) to 0 is the only way to suppress the one-line termination message printed before NL2SOL returns.) Default = standard output unit (unit 6 on most systems); the default for IV(PRUNIT) is actually I1MACH(1): see §14.

IV(SOLPRT)... IV(22) = 1 means print the X returned (along with the corresponding gradient and scale vector D). IV(SOLPRT) = 0 means skip this printing. Default = 1.

IV(STATPR)... IV(23) = 1 means print summary statistics upon returning. These consist of the function value (half the residual sum of squares) at X, the variability of the last step (see V(VCONCR) in §7), the number of function and gradient evaluations (call on CALCR and CALCJ respectively, excluding any calls used in computing the covariance), the 2-norm of the gradient at X, the corresponding V(COSMAX) (see V(COSMAX)), and the number o calls (if positive) on CALCR and CALCJ used in trying to comp covariance matrices. IV(STATPR) = 0 means skip this printing Default = 1.

IV(XOPRT)... IV(24) = 1 means print the initial X and scale vector D if this is a fresh start. IV(XOPRT) = 0 means skip this printin Default = 1.

IV Output Values of Primary Interest

IV(1)..... IV(1) is the return code: see §5.

IV(COVMAT)... IV(26) tells whether a covariance matrix was computed. If IV(COVMAT) is positive, then the lower triangle of the covari ance matrix is stored row-wise in V starting at V(IV(COVMAT)) If IV(COVMAT) = 0, then no attempt was made to compute a co- variance matrix. If IV(COVMAT) = -1, then the finite-differ Hessian H was indefinite (or, for |IV(COVREQ)| = 3, the curr Jacobian matrix is singular; see IV(COVREQ) above). And if IV(COVMAT) = -2, then a successful finite-difference step co not be found for some component of X (i.e., CALCR set NF to for each of two trial steps). Note that IV(COVMAT) is reset 0 after each successful iteration, so that if a lower functi

value is found after a restart, then a new attempt will be made to compute a covariance matrix.

IV(D)..... IV(27) is the starting subscript in V of the current scale vector D (see V(DO) in §7).

IV(G)..... IV(28) is the starting subscript in V of the current gradient vector $g = J(X)^T R(X)$.

IV(NFCALL)... IV(6) is the number of calls so far made on CALCR (i.e., the number of function evaluations, including those used in computing covariance matrices).

IV(NFCOV).... IV(40) is the number of calls made on CALCR when computing covariance matrices.

IV(NGCALL)... IV(43) is the number of calls so far made on CALCJ (i.e., the number of gradient evaluations, including those used in computing covariance matrices).

IV(NGCOV).... IV(41) is the number of calls made on CALCJ when computing covariance matrices.

IV(NITER).... IV(44) is the number of iterations performed.

IV(R)..... IV(50) is the starting subscript in V of the residual vector R(X).

7. V Values of Primary Interest

Many of the V input components described here and in §17 must lie within a certain range of values. If such a component falls outside the range indicated below (and in §17) at the beginning of its description, then module PARCHK will print an error message (unless IV(PRUNIT) = 0) and will force NL2SOL to return immediately with IV(1) ≥ 18.

Frequent reference is made below to two quantities: MACHEP and the scale vector D. MACHEP is the unit roundoff for the floating point arithmetic being used -- see §14. The scale vector D is the diagonal of the (diagonal) scale matrix D_k discussed in §§5 and 7 of [1]; this scale matrix is denoted by diag(D) below.

V Input Values of Primary Interest (Supplied by DFAULT)

V(CCONCR)... V(29) ∈ [0, 1] is the cosine convergence tolerance. Let J_i denote the i^{th} column of the $n \times p$ Jacobian matrix J and let

$$\text{cosmax}(R,J) = \max\left\{ \frac{|R^T J_i|}{\|R\|_2 \cdot \|J_i\|_2} : \|J_i\|_2 > \text{JTOL}(i),\ 1 \le i \le p \right\},$$

where JTOL is described with V(JTINIT) below. If NL2SOL finds an X giving $\text{cosmax}(R(X),J(X)) \le$ V(CCONCR), then it returns with IV(1) = 4. Default = $\max(10^{-7}, 1000 \cdot \text{MACHEP})$.

V(DELTA0)... V(31) ∈ [MACHEP, 1] helps pick the finite-difference steps

used in computing H when IV(COVREQ) = 1 or 2. The step used for component X(i) is

$$V(DELTA0) \cdot \max\{|X(i)|, 1/D(i)\} \cdot \operatorname{sign}(X(i)),$$

where D is the current scale vector. (If this step results in CALCR setting NF to 0, then -0.5 times this step is also tried.) Default = $MACHEP^{(1/2)}$.

V(DFAC)..... V(32) $\in$ [0, 1] and V(DO) are used in updating the scale vector D -- see V(DO) below. Default = 0.6.

V(DINIT).... V(33), if nonnegative, is the value to which all components of the scale vector D are initialized. Default = 0.

V(DLTFDC)... V(34) $\in$ [MACHEP, 1] helps pick the step sizes used in computing H when IV(COVREQ) = -1 or -2. For differences involving X(i), the step first tried is

$$V(DLTFDC) \cdot \max\{|X(i)|, 1/D(i)\}.$$

(If this step is too large, i.e., if CALCR sets NF to 0 when this step is first tried, then -0.5 times this step is also tried.) Default = $MACHEP^{(1/3)}$.

V(DLTFDJ)... V(35) $\in$ [MACHEP, 1] helps pick the step sizes that NL2SNO uses when computing its finite-difference approximation to the Jacobian matrix (see §8). For differences involving X(i), the step first tried is $V(DLTFDJ) \cdot \max\{|X(i)|, 1/D(i)\}$. (If this step is too large, i.e., if CALCR sets NF to 0, then smaller steps are tried until the step size is shrunk below 1000·MACHEP.) Default = $MACHEP^{(1/2)}$.

V(DO)..... V(36) and V(DFAC) are used in updating the scale vector D. If V(DO) > 0, then at the start of each iteration, D(i) is set to

$$\max\{[\|J_i\|_2^2 + \max\{S_{ii}, 0\}]^{1/2}, V(DFAC)\cdot D(i), JTOL(i), V(DO)\},$$

where J_i is the i^{th} column of the current Jacobian matrix, S is the S matrix of [1], and JTOL is the array described with V(JTINIT) below. If $-1 < V(DO) < 0$, then D is set to the above values (after any initialization due to V(DINIT)) on the first iteration and is not changed again. If V(DO) = 0, then all components of D are set to 1 (regardless of V(DINIT)), which usually gives good performance on well-scaled problems. If $V(DO) \leq -1$, then it is assumed that the caller has chosen D and has stored it in V, starting at V(96 + 2N + P[7P + 41]/2). Default = 10^{-3}.

V(JTINIT)... V(38) ≥ 0. For $1 \leq i \leq P$, JTOL(i) is a tolerance used to decide whether the i-th column of the Jacobian matrix should be considered to be zero. If V(JTINIT) > 0, then all components of the JTOL array are set to V(JTINIT), and if V(JTINIT) = 0, then it is assumed that the caller has stored JTOL in V starting at V(87). Default = 10^{-9}.

V(LMAX0)..., V(39) > 0 gives the maximum 2-norm allowed for the very first

step that NL2SOL attempts. On problems where this step would otherwise be inordinately large, it is very useful to assign a modest value to V(LMAX0). Default = 100.

V(RCONCR)... $V(40) > 0$ is the residual convergence tolerance. If $\|R(X)\|_2 \leq V(RCONCR)$, then NL2SOL returns with IV(1) = 5. Default = 10^{-9}.

V(VCONCR)... $V(42) \geq 0$ is the variability convergence tolerance. If $\hat{H}$ is the current Hessian approximation, then the variability of the current step Δx is

$$V(VARIAB) = \max\{1, N-P\} \cdot \Delta x^T \hat{H} \Delta x \,/\, (2F(X)),$$

where $2F(X)$ is the current residual sum of squares. If $V(VARIAB) \leq V(VCONCR)$, then NL2SOL returns with IV(1) = 6. Default = 10^{-4}.

V(XCONCR)... $V(26) \geq 0$ is the X convergence tolerance. If a step Δx is tried that yields a much smaller function decrease than expected, and if $V(RELDX) \leq V(XCONCR)$, where V(RELDX) is the maximum relative change in any component of X [which, for $X = X_o + \Delta x$, is computed as

$$V(RELDX) = \max\left\{\frac{|X(i) - X_o(i)|}{|X(i)| + |X_o(i)|} : 1 \leq i \leq P\right\},$$

then NL2SOL returns with IV(1) = 3. Default = 1000·MACHEP.

V Output Values of Primary Interest

V(COSMAX)... V(43) = cosmax(R(X), J(X)) -- see V(CCONCR) above.

V(DGNORM)... $V(1) = \|\mathrm{diag}(D)J(X)^T R(X)\|_2$, where D is the current scale vector and $J(X)^T R(X) = \nabla F(X)$.

V(DSTNRM)... $V(2) = \|\mathrm{diag}(D)\Delta x\|_2$, where Δx is the most recently tried step.

V(F)..... $V(10) = F(X) = \|R(X)\|_2^2 / 2$.

V(RELDX)... V(17) is the maximum relative change in X caused by the most recent step -- see V(XCONCR) above.

V(VARIAB)... V(50) is the variability of the most recent step -- see V(VCONCR) above.

8. Finite-Difference Jacobians -- NL2SNO

Those who do not wish to code a subroutine CALCJ for (analytically) computing the Jacobian matrix may avoid doing so by calling NL2SNO instead of NL2SOL. NL2SNO computes an approximate Jacobian matrix by forward differences (using a step size determined by V(DLTFDJ) -- see §7). The calling sequence for NL2SNO amounts to the one for NL2SOL with CALCJ omitted:

CALL NL2SNO(N, P, X, CALCR, IV, V, UIPARM, URPARM, UFPARM)

The parameters for NL2SNO are the same as the corresponding ones for NL2SOL. One minor exception occurs with the handling of IV(COVREQ): if IV(COVPRT) = 1 and IV(COVREQ) = 0, then NL2SNO sets IV(COVREQ) = -1; otherwise NL2SNO sets IV(COVREQ) to $-|IV(COVREQ)|$. Thus NL2SNO uses function values only in computing covariance matrices and V(DELTA0) is not used.

9. Restarting

After any return with $3 \le IV(1) \le 9$, it is possible to change some of the IV and V input components (such as the convergence tolerances and the iteration and function evaluation limits) and call NL2SOL (or NL2SNO) again with IV(1) unchanged. This causes the algorithm to be resumed at the point where it was interrupted. (It is even possible to save IV, V, and X and then restart in a subsequent run.)

10. Scaling

Problems sometimes arise which are poorly scaled in the sense that the various components of X are expressed in widely differing units. With the default choice of the scale matrix D (see V(D0) and the beginning of §7), the behavior of NL2SOL is largely insensitive to this kind of poor scaling. On well scaled problems, its performance can often be improved by choosing D to be the identity matrix (i.e., setting V(D0) = 0). Sometimes it may also be worthwhile to fix D(i), $1 \le i \le P$, at the 2-norm of the i-th column of the initial Jacobian matrix (by setting V(D0) = -0.5).

11. LMAX0: Limiting the First Step Length

On some problems it is necessary to give V(LMAX0) = V(39) 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 V(LMAX0).

Note that if Δx is the very first step attempted, then $\|\Delta x\|_2$ rather than $\|\mathrm{diag}(D)\Delta x\|_2$ is bounded above by V(LMAX0), because it was felt that the caller might have a better idea of how $\|\Delta x\|_2$ should be limited. As a result, V(LMAX0) is a scale-sensitive quantity.

12. Local 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

13. Printed Output

Any printing is done by one of two modules: ITSMRY and PARCHK.

PARCHK reports any V input components that are out of range and optionally lists any such components that have nondefault or changed values (on a fresh start or restart respectively). ITSMRY does the remaining printing. Various IV input components control what printing is done -- see §6.

If IV(OUTLEV) > 0, then ITSMRY produces an iteration summary which includes the following values: IT, the current iteration number; NF, the number of function evaluations (calls on CALCR), excluding any extra ones needed for computing covariance matrices and, in the case of NL2SNO, excluding the extra ones needed to compute finite-difference Jacobian matrices; F, the current function value (half the residual sum of squares); DF, the difference between the previous and current function values; COSMAX, the current cosmax(R(X), J(X)) -- see V(CCONCR) in §7; VAR, the current step variability -- see V(VCONCR) in §7; MODEL, which tells which models were used in computing the current step [G = the Gauss-Newton model; S = the augmented model; G-S means the Gauss-Newton model was tried first and a switch was then made to the augmented model; S-G, G-S-G, and S-G-S have analogous meanings]; LAMBDA, the current Levenberg-Marquardt parameter λ (which is nonnegative if the step Δx just taken satisfies $[\hat{H} + \lambda \cdot \mathrm{diag}(D)^2]\Delta x = -\nabla F(X - \Delta x)$, where $\hat{H}$ is the current Hessian approximation, and is negative if the special case discussed in [2] was detected); RELDX, the current value of V(RELDX) -- see V(XCONCR) in §7; G, the current value of $\|\nabla F(X)\|_2 = \|J(X)^T R(X)\|_2$; SIZE, the sizing factor just used in updating the S matrix (see [1]); and D*STEP, the current value of V(DSTNRM) -- see §7. These summary lines are 118 characters long (including the carriage control character). If IV(OUTLEV) < 0, then lines of maximum length 69 (or 56 if IV(COVPRT) = 0) are generated, and the iteration summary includes only the first six items described above (i.e., IT, NF, F, DF, COSMAX, and VAR).

14. Changing Computers

The NL2SOL distribution tape contains both single- and double-precision versions of the NL2SOL source code, so it should be unnecessary to change precisions. (On computers having only 32 or 36 bits per REAL word, double precision often gives better performance.)

Only the functions IMDCON and RMDCON contain machine-dependent constants. To change from one computer to another, it should suffice to change the DATA statements in these functions. The DATA statement in IMDCON sets MDCON(1) to the output unit number that DFAULT supplies to IV(PRUNIT). The machine-dependent DATA statement in RMDCON provides three values: BIG, ETA, and MACHEP. BIG is the largest floating-point number such that a FORTRAN program can compute SQRT(0.999*BIG)**2 [i.e., DSQRT(0.999D0*BIG)**2 in DOUBLE PRECISION] without overflowing. Similarly, ETA is the smallest floating-point number such that SQRT(1.001*ETA)**2 [or DSQRT(1.001D0*ETA)**2 respectively] does not underflow. MACHEP is the unit roundoff, i.e., the smallest floating-point number such that 1 + MACHEP yields a stored floating point number greater than 1 and 1 - MACHEP yields a stored number less than 1. (Some computers feature registers that carry more bits than can be stored; MACHEP should only reflect the accuracy of numbers that can be stored.)

The test program supplied on the NL2SOL distribution tape places the further restriction on BIG and ETA that EXP(1.999*ALOG(SQRT(0.999*BIG))) and EXP(1.999*ALOG(SQRT(1.001*ETA))) [DEXP(1.999D0*DLOG(DSQRT(0.999D0*BIG))) and DEXP(1.999D0*DLOG(DSQRT(1.001D0*ETA))) in DOUBLE PRECISION] not overflow or underflow. DATA statements giving suitable values for BIG, ETA, and MACHEP for a variety of computers appear as comments in RMDCON.

Intrinsic functions are explicitly declared in the NL2SOL source code. On certain computers (e.g. Univac), it may be necessary to comment out these declarations. So that this may be done automatically by a simple program, such declarations are preceded by a comment having C/+ in columns 1-3 and blanks in columns 4-72 and are followed by a comment having C/ in columns 1 and 2 and blanks in columns 3-72.

15. Using Reverse Communication -- NL2ITR

Instead of writing subroutines CALCR and CALCJ to compute the residual vector R(X) and Jacobian matrix J(X), one can call NL2ITR and provide R and J by reverse communication. The calling sequence is:

```
CALL NL2ITR(D, IV, J, N, NN, P, R, V, X)
```

Parameters IV, N, P, V, and X are the same as the corresponding parameters to NL2SOL, with the following exceptions: V need only contain 96 + 2N + P[7P + 41]/2 elements, since the storage that NL2SOL and NL2SNO allocate for D, J, and R at the end of V is not needed; and components IV(D), IV(J), and IV(R) are not referenced. D is the scale vector (dimensioned D(P)). NN is the (integer) lead dimension for the J array, which is dimensioned J(NN,P); NN must satisfy NN ≥ N.

When NL2ITR is first called (with IV(1) = 0 or 10), J must have been set to J(X), R to R(X). When NL2ITR wants R to be evaluated at a new X, it returns with IV(1) = 1; the caller should then set R to R(X) (unless X is out of range, in which case the caller should set IV(TOOBIG), i.e., IV(2), to 1) and call NL2ITR again. Similarly, when NL2ITR wants J to be evaluated at X, it returns with IV(1) = 2, and the caller should then set J to J(X) and call NL2ITR again. (If J cannot be evaluated at X, the caller may set IV(NFGCAL), i.e., IV(7), to 0; this will cause NL2ITR to give the error return IV(1) = 13.)

16. STOPX

It is possible to arrange for NL2SOL (NL2SNO and NL2ITR) to be interrupted when used in an interactive environment. 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 and only if the "break" (i.e., "interrupt") key has been pressed since the last call on STOPX. Once this is done, NL2SOL will return with IV(1) = 9 when the "break" key is pressed before some other return has occurred. It is then possible to change some of the IV and V input components and restart -- see §9,

17. Other V Input Values