Distribution Category:
Mathematics and Computers
(UC-32)

ANL-80-74

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

USER GUIDE FOR MINPACK-1

by

Jorge J. Moré, Burton S. Garbow, Kenneth E. Hillstrom

Applied Mathematics Division

August 1980

TABLE OF CONTENTS

ABSTRACT

MINPACK-1 is a package of Fortran subprograms for the numerical solution of systems of nonlinear equations and nonlinear least squares problems. This report provides an overview of the algorithms and software in the package and includes the documentation and program listings.

Preface

The MINPACK Project is a research effort whose goal is the development of a systematized collection of quality optimization software. The first step towards this goal has been realized in MINPACK-1, a package of Fortran programs for the numerical solution of systems of nonlinear equations and nonlinear least squares problems.

The design of the algorithms and software in MINPACK-1 has several objectives; the main ones are reliability, ease of use, and transportability.

At the algorithmic level, reliability derives from the underlying algorithms having a sound theoretical basis. Entirely satisfactory global convergence results are available for the MINPACK-1 algorithms and, in addition, their properties allow scale invariant implementations.

At the software level, reliability derives from extensive testing. The heart of the testing aids is a large collection of test problems (Moré, Garbow, and Hillstrom [1978]). These test problems have been used to measure the performance of the software on the following computing systems: IBM 360/370, CDC 6000-7000, Univac 1100, Cray-1, Burroughs 6700, DEC PDP-10, Honeywell 6000, Prime 400, Itel AS/6, and ICL 2980. At Argonne, software performance has been further measured with the help of WATFIV and BRNANL (Fosdick [1974]). WATFIV detects run-time errors such as undefined variables and out-of-range subscripts, while BRNANL provides execution counts for each block of a program and, in particular, has established that the MINPACK-1 test problems execute every non-trivial program block.

Reliability further implies efficient and robust implementations. For example, MINPACK-1 programs access matrices sequentially along columns (rather than rows), since this improves efficiency, especially on paged systems. Also, there are extensive checks on the input parameters, and computations are formulated to avoid destructive underflows and overflows. Underflows can then be safely ignored; overflows due to the problem should of course be investigated.

Ease of use derives from the design of the user interface. Each algorithmic path in MINPACK-1 includes a core subroutine and a driver with a simplified calling sequence made possible by assuming default settings for certain parameters and by returning a limited amount of information; many applications do not require full flexibility and in these cases the drivers can be invoked. On the other hand, the core subroutines enable, for example, scaling of the variables and printing of intermediate results at specified iterations.

Ease of use is also facilitated by the documentation. Machine-readable documentation is provided for those programs normally called by the user. The documentation includes discussions of all calling sequence parameters and an actual example illustrating the use of the corresponding algorithm. In addition, each program includes detailed prologue comments on its purpose and the roles of its parameters; in-line comments introduce major blocks in the body of the program.

To further clarify the underlying structure of the algorithms, the programs have been formatted by the TAMPR system of Boyle and Dritz [1974]. TAMPR produces implementations in which the loops and logical structure of the programs are clearly delineated. In addition, TAMPR has been used to produce the single precision version of the programs from the master (double precision) version.

Transportability requires that a satisfactory transfer to a different computing system be possible with only a small number of changes to the software. In MINPACK-1, a change to a new computing system only requires changes to one program in each precision; all other programs are written in a portable subset of ANSI standard Fortran acceptable to the PFORT verifier (Ryder [1974]). This one machine-dependent program provides values of the machine precision, the smallest magnitude, and the largest magnitude. Most of the values for these parameters were obtained from the corresponding PORT library program (Fox, Hall, and Schryer [1978]); in particular, values are provided for all of the computing systems on which the programs were tested.

MINPACK-1 is fully supported. Comments, questions, and reports of poor or incorrect performance of the MINPACK-1 programs should be directed to

Burton S. Garbow
Applied Mathematics Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439
Phone: (312) 972-7184

Of particular interest would be reports of performance of the MINPACK-1 package on machines not covered in the testing.

The MINPACK-1 package consists of the programs, their documentation, and the testing aids. The package comprises approximately 28,000 card images and is transmitted on magnetic tape. The tape is available from the following two sources.

National Energy Software Center
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439
Phone: (312) 972-7250

IMSL
Sixth Floor-NBC Building
7500 Bellaire Blvd.
Houston, TX 77036
Phone: (713) 772-1927

The package includes both single and double precision versions of the programs, and for those programs normally called by the user machine-readable documentation is provided in both single and double precision forms. An implementation guide (Garbow, Hillstrom, and Moré [1980]) is also included with the tape.

Acknowledgments

The MINPACK-1 testing was conducted by the following individuals; their assistance and suggestions were invaluable to the project.

Poul Arendal, The World Bank (Burroughs)
Richard Bartels, University of Waterloo (Honeywell)
Mary Ann Berg, University of Illinois at Urbana-Champaign (CDC)
W. Robert Boland, Air Force Weapons Laboratory (CDC)
Roger Crane, RCA Laboratories (IBM)
Dona Crawford, Sandia Laboratories, Livermore (CDC)
John Dennis, Rice University (Itel)
Jeremy DuCroz, Numerical Algorithms Group Ltd. (ICL)
Jay Fleisher, The University of Wisconsin (Univac)
Fred Fritsch, Lawrence Livermore Laboratory (CDC,Cray)
Patrick Gaffney, Union Carbide Corporation (CDC,Cray,DEC,IBM)
David Gay, Massachusetts Institute of Technology (IBM)
Kathie Hiebert, Sandia Laboratories, Albuquerque (CDC)
L. W. Lucas, Naval Weapons Center (Univac)
Dan O'Reilly, Data Resources, Inc. (Burroughs)
Gerald Ruderman, Management Decision Systems, Inc. (Prime)
Nora Sabelli, University of Illinois at Chicago Circle (IBM)
Susan Sherry, Federal Reserve Board (IBM)
Danny Sorensen, University of Kentucky (IBM)
Jesse Wang, Argonne National Laboratory (IBM)

Many others have contributed to the package in various ways; in particular, we acknowledge Beverly Arnoldy, Jim Boyle, Ken Brown, Wayne Cowell, Jim Cody, Tom Coleman, Bill Davidon, Jack Dongarra, Dudley Goetschel, Jerry Kreuser, James Lyness, Mike Minkoff, Larry Nazareth, Mike Powell, Rich Raffenetti, Bob Schnabel, Greg Shubert, Brian Smith, David Thuente, and Richard Wilk.

Special thanks go to Jim Pool, the originator of the project, and to Paul Messina, the head of the Applied Mathematical Sciences section of the Applied Mathematics Division at Argonne. Finally, thanks to Judy Beumer for her usual outstanding job of typing this report.

CHAPTER 1
Introduction to MINPACK-1

The purpose of this chapter is to provide an overview of the algorithms and software in MINPACK-1. Most users need only be acquainted with the first six sections of this chapter; the remaining two sections describe lower-level software called from the main programs.

1.1 Systems of Nonlinear Equations

If n functions $f_1,f_2,\ldots,f_n$ of the n variables $x_1,x_2,\ldots,x_n$ are specified, then MINPACK-1 subroutines can be used to find values for $x_1,x_2,\ldots,x_n$ that solve the system of nonlinear equations

$$f_i(x_1,x_2,\ldots,x_n) = 0 , \qquad 1 \le i \le n .$$

To solve this system we have implemented a modification of Powell's hybrid algorithm. There are two variants of this algorithm. The first variant only requires that the user calculate the functions f_i, while the second variant requires that the user calculate both the functions f_i and the n by n Jacobian matrix

$$\left(\frac{\partial f_i(x)}{\partial x_j}\right), \qquad 1 \le i \le n , \quad 1 \le j \le n .$$

1.2 Nonlinear Least Squares Problems

If m functions $f_1,f_2,\ldots,f_m$ of the n variables $x_1,x_2,\ldots,x_n$ are specified with $m \ge n$, then MINPACK-1 subroutines can be used to find values for $x_1,x_2,\ldots,x_n$ that solve the nonlinear least squares problem

$$\min\left\{ \sum_{i=1}^{m} f_i(x)^2 : x \in R^n \right\} .$$

To solve this problem we have implemented a modification of the Levenberg-Marquardt algorithm. There are three variants of this algorithm. The first variant only requires that the user calculate the functions f_i, while the second variant requires that the user calculate both the functions f_i and the m by n Jacobian matrix

$$\left(\frac{\partial f_i(x)}{\partial x_j}\right), \qquad 1 \le i \le m \ , \ 1 \le j \le n \ .$$

The third variant also requires that the user calculate the functions and the Jacobian matrix, but the latter only one row at a time. This organization only requires the storage of an n by n matrix (rather than m by n), and is thus attractive for nonlinear least squares problems with a large number of functions and a moderate number of variables.

1.3 Derivative Checking

The main advantage of providing the Jacobian matrix is increased reliability; for example, the algorithm is then much less sensitive to functions subject to errors. However, providing the Jacobian matrix is an error-prone task. To help identify errors, MINPACK-1 also contains a subroutine CHKDER that checks the Jacobian matrix for consistency with the function values.

1.4 Algorithmic Paths: Core Subroutines and Easy-to-Use Drivers

There are five general algorithmic paths in MINPACK-1. Each path includes a core subroutine and an easy-to-use driver with a simplified calling sequence made possible by assuming default settings for certain parameters and by returning a limited amount of information; many applications do not require full flexibility and in these cases easy-to-use drivers can be invoked. On the other hand, the core subroutines enable, for example, scaling of the variables and printing of intermediate results at specified iterations.

1.5 MINPACK-1 Subroutines: Systems of Nonlinear Equations

The MINPACK-1 subroutines for the numerical solution of systems of nonlinear equations are HYBRD1, HYBRD, HYBRJ1, and HYBRJ. These subroutines provide alternative ways to solve the system of nonlinear equations

$$f_i(x_1,x_2,\ldots,x_n) = 0 \ , \qquad 1 \le i \le n$$

by a modification of Powell's hybrid algorithm. The principal requirements of the subroutines are as follows (see also Figure 1).

HYBRD1, HYBRD

The user must provide a subroutine to calculate the functions $f_1, f_2, \ldots, f_n$. The Jacobian matrix is then calculated by a forward-difference approximation or by an update formula of Broyden. HYBRD1 is the easy-to-use driver for the core subroutine HYBRD.

HYBRJ1, HYBRJ

The user must provide a subroutine to calculate the functions $f_1, f_2, \ldots, f_n$ and the Jacobian matrix

$$\left(\frac{\partial f_i(x)}{\partial x_j}\right), \qquad 1 \le i \le n, \quad 1 \le j \le n .$$

(Subroutine CHKDER can be used to check the Jacobian matrix for consistency with the function values.) HYBRJ1 is the easy-to-use driver for the core subroutine HYBRJ.

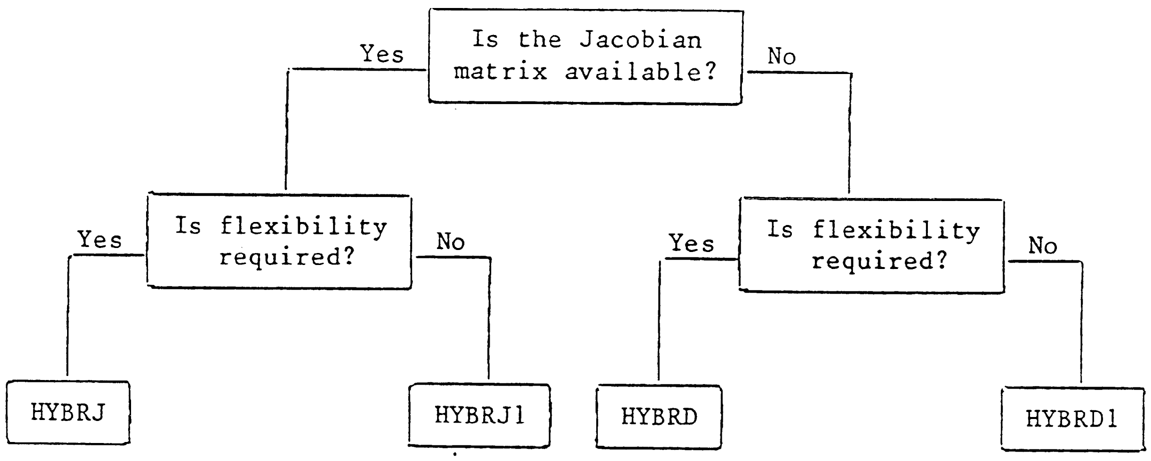

Figure 1
Decision Tree for Systems of Nonlinear Equations

1.6 MINPACK-1 Subroutines: Nonlinear Least Squares Problems

The MINPACK-1 subroutines for the numerical solution of nonlinear least squares problems are LMDIF1, LMDIF, LMDER1, LMDER, LMSTR1, and LMSTR. These subroutines provide alternative ways to solve the nonlinear least squares problem

$$\min\left\{\sum_{i=1}^{m} f_i(x)^2 : x \in R^n\right\}$$

by a modification of the Levenberg-Marquardt algorithm. The principal requirements of the subroutines are as follows (see also Figure 2).

LMDIF1, LMDIF

The user must provide a subroutine to calculate the functions $f_1,f_2,\ldots,f_m$. The Jacobian matrix is then calculated by a forward-difference approximation. LMDIF1 is the easy-to-use driver for the core subroutine LMDIF.

LMDER1, LMDER

The user must provide a subroutine to calculate the functions $f_1,f_2,\ldots,f_m$ and the Jacobian matrix

$$\left(\frac{\partial f_i(x)}{\partial x_j}\right), \qquad 1 \le i \le m \,, \quad 1 \le j \le n \,.$$

(Subroutine CHKDER can be used to check the Jacobian matrix for consistency with the function values.) LMDER1 is the easy-to-use driver for the core subroutine LMDER.

LMSTR1, LMSTR

The user must provide a subroutine to calculate the functions $f_1,f_2,\ldots,f_m$ and the rows of the Jacobian matrix

$$\left(\frac{\partial f_i(x)}{\partial x_j}\right), \qquad 1 \le i \le m \,, \quad i \le j \le n \,,$$

one row per call. (Subroutine CHKDER can be used to check the row of the Jacobian matrix for consistency with the corresponding function value.) LMSTR1 is the easy-to-use driver for the core subroutine LMSTR.

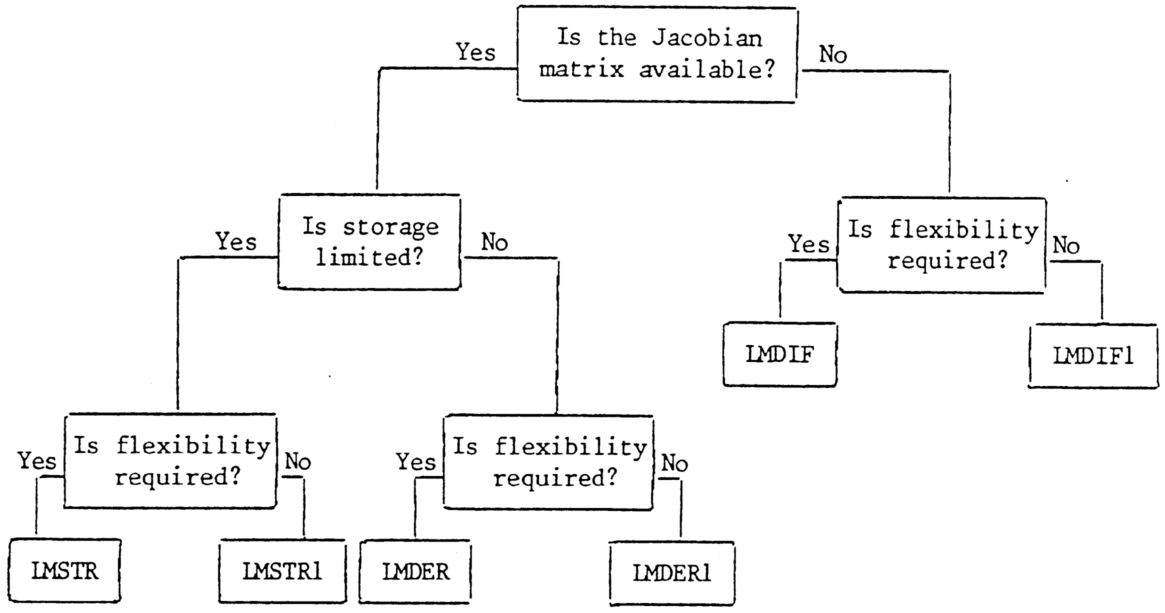

Figure 2
Decision Tree for Nonlinear Least Squares Problems

1.7 Machine-Dependent Constants

There are three machine-dependent constants that have to be set before the single or double precision version of MINPACK-1 can be used; for most machines the correct values of these constants are encoded into DATA statements in functions SPMPAR (single precision) and DPMPAR (double precision). These constants are:

$\beta^{1-\ell}$, the machine precision ,

$\beta^{e_{min}-1}$, the smallest magnitude ,

$(1 - \beta^{-\ell})\beta^{e_{max}}$, the largest magnitude ,

where ℓ is the number of base β digits on the machine, e_{min} is the smallest machine exponent, and e_{max} is the largest machine exponent.

The most critical of the constants is the machine precision ε_M, since the MINPACK-1 subroutines treat two numbers a and b as equal if they satisfy

$$|b-a| \leq \varepsilon_M |a| ,$$

and the above test forms the basis for deciding that no further improvement is possible with the algorithm.

1.8 MINPACK-1 Internal Subprograms

Most users of MINPACK-1 need only be acquainted with the core subroutines and easy-to-use drivers described in the previous sections. Some users, however, may wish to experiment by modifying an algorithmic path to improve the performance of the algorithm on a particular application. A modification to an algorithmic path can often be achieved by modifying or replacing one of the internal subprograms. Additionally, the internal subprograms may be useful independent of the MINPACK-1 algorithmic paths in which they are employed.

For these reasons brief descriptions of the MINPACK-1 internal subprograms are included below; more complete descriptions can be found in the prologue comments in the program listings of Chapter 5.

DOGLEG

Given the QR factorization of an m by n matrix A, an n by n nonsingular diagonal matrix D, an m-vector b, and a positive number Δ, this subroutine determines the convex combination of the Gauss-Newton and scaled gradient directions that solves the problem

$$\min\{\|Ax-b\| : \|Dx\| \leq \Delta\} .$$

ENORM

This function computes the Euclidean norm of a vector x.

FDJAC1

This subroutine computes a forward-difference approximation to the Jacobian matrix associated with n functions in n variables. It includes a banded Jacobian option.

FDJAC2

This subroutine computes a forward-difference approximation to the Jacobian matrix associated with m functions in n variables.

LMPAR

Given the QR factorization of an m by n matrix A, an n by n nonsingular diagonal matrix D, an m-vector b, and a positive number Δ, this subroutine is used to solve the problem

$$\min\{\|Ax-b\| : \|Dx\| \le \Delta\} .$$

QFORM

Given the QR factorization of a rectangular matrix, this subroutine accumulates the orthogonal matrix Q from its factored form.

QRFAC

This subroutine uses Householder transformations with optional column pivoting to compute a QR factorization of an arbitrary rectangular matrix.

QRSOLV

Given the QR factorization of an m by n matrix A, an n by n diagonal matrix D, and an m-vector b, this subroutine solves the linear least squares problem

$$\binom{A}{D}x \cong \binom{b}{0} .$$

RWUPDT

This subroutine is used in updating the upper triangular part of the QR decomposition of a matrix A after a row is added to A.

R1MPYQ

This subroutine multiplies a matrix by an orthogonal matrix given as a product of Givens rotations.

R1UPDT

This subroutine is used in updating the lower triangular part of the LQ decomposition of a matrix A after a rank-1 matrix is added to A.

CHAPTER 2
Algorithmic Details

The purpose of this chapter is to provide information about the algorithms and to point out some of the ways in which this information can be used to improve their performance. The first two sections are essential for the rest of the chapter since they provide the necessary background, but the other sections are independent of each other.

2.1 Mathematical Background

To describe the algorithms for the solution of systems of nonlinear equations and nonlinear least squares problems, it is necessary to introduce some notation.

Let R^n represent the n-dimensional Euclidean space of real n-vectors

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix},$$

and $\|x\|$ the Euclidean norm of x,

$$\|x\| = \left(\sum_{j=1}^{n} x_j^2 \right)^{\frac{1}{2}} .$$

A function F with domain in R^n and range in R^m is denoted by $F\colon R^n \to R^m$. Such a function can be expressed as

$$F(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{pmatrix},$$

where the component function $f_i\colon R^n \to R$ is sometimes called the i-th residual of F. The terminology derives from the fact that a common problem is to fit a model g(t,x) to data y, in which case the f_i are of the form

$$f_i(x) = y_i - g(t_i, x) ,$$

where y_i is measured at t_i and x is the set of fit parameters.

In this notation a system of nonlinear equations is specified by a function $F\colon R^n \to R^n$, and a solution vector x^* in R^n is such that

$$F(x^*) = 0 .$$

Similarly, a nonlinear least squares problem is specified by a function $F\colon R^n \to R^m$ with $m \geq n$, and a solution vector x^* in R^n is such that

$$\|F(x^*)\| \leq \|F(x)\| \quad \text{for} \quad x \in N(x^*) ,$$

where $N(x^*)$ is a neighborhood of x^*. If $N(x^*)$ is the entire domain of definition of the function, then x^* is a global solution; otherwise, x^* is a local solution.

Some of the MINPACK-1 algorithms require the specification of the Jacobian matrix of the mapping $F\colon R^n \to R^m$; that is, the m by n matrix $F'(x)$ whose (i,j) entry is

$$\frac{\partial f_i(x)}{\partial x_j} .$$

A related concept is the gradient of a function $f\colon R^n \to R$, which is the mapping $\nabla f\colon R^n \to R^n$ defined by

$$\nabla f(x) = \begin{pmatrix} \dfrac{\partial f(x)}{\partial x_1} \\ \dfrac{\partial f(x)}{\partial x_2} \\ \vdots \\ \dfrac{\partial f(x)}{\partial x_n} \end{pmatrix} .$$

Note that the i-th row of the Jacobian matrix $F'(x)$ is the gradient $\nabla f_i(x)$ of the i-th residual.

It is well-known that if x^* is a solution of the nonlinear least squares problem, then x^* solves the system of nonlinear equations

$$\sum_{i=1}^{m} f_i(x)\nabla f_i(x) = 0 .$$

In terms of the Jacobian matrix this implies that

$$F'(x^*)^T F(x^*) = 0 ,$$

and shows that at the solution the vector of residuals is orthogonal to the columns of the Jacobian matrix. This orthogonality condition is also satisfied at maximizers and saddle points, but algorithms usually take precautions to avoid these critical points.

2.2 Overview of the Algorithms

Consider a mapping $F: R^n \to R^m$, where $m = n$ for systems of nonlinear equations and $m \geq n$ for nonlinear least squares problems. The MINPACK-1 algorithms in these two problem areas seek a solution x^* of the problem

$$(1) \qquad \min\{\|F(x)\|: x \in R^n\} .$$

In particular, if $m = n$ it is expected that $F(x^*) = 0$.

Our initial description of the algorithms will be at the macroscopic level where the techniques used in each problem area are similar.

With each algorithm the user provides an initial approximation $x = x_o$ to the solution of the problem. The algorithm then determines a correction p to x that produces a sufficient decrease in the residuals of F at the new point $x+p$; it then sets

$$x_+ = x + p$$

and begins a new iteration with x_+ replacing x.

A sufficient decrease in the residuals implies, in particular, that

$$\|F(x+p)\| < \|F(x)\| ,$$

and thus the algorithms guarantee that

$$\|F(x_+)\| < \|F(x)\| .$$

The correction p depends upon a diagonal scaling matrix D, a step bound Δ, and an approximation J to the Jacobian matrix of F at x. Users of the core subroutines can specify initial values D_o and Δ_o; in the easy-to-use drivers D_o and Δ_o are set internally. If the user is providing the Jacobian matrix, then $J_o = F'(x_o)$; otherwise the algorithm sets J_o to a forward difference approximation to $F'(x_o)$.

To compute p, the algorithm solves (approximately) the problem

$$\min\{\|f+Jp\|: \|Dp\| \leq \Delta\} , \tag{2}$$

where f is the m-vector of residuals of F at x. If the solution of this problem does not provide a suitable correction, then Δ is decreased and, if appropriate, J is updated. A new problem is now solved, and this process is repeated (usually only once or twice) until a p is obtained at which there is sufficient decrease in the residuals, and then x is replaced by x+p. Before the start of the next iteration, D, Δ, and J are also replaced.

The motivation for using (2) to obtain the correction p is that for appropriate choices of J and Δ, the solution of (2) is an approximate solution of

$$\min\{\|F(x+p)\|: \|Dp\| \leq \Delta\} .$$

It follows that if there is a solution x* such that

$$\|D(x-x^*)\| \leq \Delta , \tag{3}$$

then x+p is close to x*. If this is not the case, then at least x+p is a better approximation to x* than x. Under reasonable conditions, it can be shown that (3) eventually holds.

The algorithms for systems of nonlinear equations and for nonlinear least squares problems differ, for example, in the manner in which the correction p is obtained as an approximate solution of (2). The nonlinear equations algorithm obtains a p that minimizes $\|f+Jp\|$ in a two-dimensional subspace of the ellipsoid $\{p: \|Dp\| \leq \Delta\}$. The nonlinear least squares algorithm obtains a p that is the exact solution of (2) with a small (10%) perturbation of Δ. Other differences in the algorithms include convergence criteria (Section 2.3) and the manner in which J is computed (Section 2.4).

It is appropriate to close this overview of the algorithms by discussing two of their limitations. First, the algorithms are limited by the precision of the computations. Although the algorithms are globally convergent under reasonable conditions, the convergence proofs are only valid in exact arithmetic and the algorithms may fail in finite precision due to roundoff. This implies that the algorithms tend to perform better in higher precision. It also implies that the calculation of the function and the Jacobian matrix should be as accurate as possible and that improved performance results when the user can provide the Jacobian analytically.

Second, the algorithms are only designed to find local solutions. To illustrate this point, consider

$$F(x) = x^3 - 3x + 18 .$$

In this case, problem (1) has the global solution $x^* = -3$ with $F(x^*) = 0$ and the local solution $x^* = 1$ with $F(x^*) = 16$; depending on the starting point, the algorithms may converge either to the global solution or to the local solution.

2.3 Convergence Criteria

The convergence test in the MINPACK-1 algorithms for systems of nonlinear equations is based on an estimate of the distance between the current approximation x and an actual solution x^* of the problem. If D is the current scaling matrix, then this convergence test (X-convergence) attempts to guarantee that

(1) $$\|D(x-x^*)\| \leq \text{XTOL}\cdot\|Dx^*\| ,$$

where XTOL is a user-supplied tolerance.

There are three convergence tests in the MINPACK-1 algorithms for nonlinear least squares problems. One test is again for X-convergence, but the main convergence test is based on an estimate of the distance between the Euclidean norm $\|F(x)\|$ of the residuals at the current approximation x and the optimal value $\|F(x^*)\|$ at an actual solution x^* of the problem. This convergence test (F-convergence) attempts to guarantee that

$$(2) \qquad \|F(x)\| \le (1 + \mathrm{FTOL})\cdot\|F(x^*)\| ,$$

where FTOL is a second user-supplied tolerance.

The third convergence test for the nonlinear least squares problem (G-convergence) guarantees that

$$(3) \qquad \max\left\{\frac{|a_i^T f|}{\|a_i\|\|f\|} : 1 \le i \le n\right\} \le \mathrm{GTOL} ,$$

where $a_1, a_2, \ldots, a_n$ are the columns of the current approximation to the Jacobian matrix, f is the vector of residuals, and GTOL is a third user-supplied tolerance.

Note that individual specification of the above three tolerances for the nonlinear least squares problem requires direct user call of the appropriate core subroutine. The easy-to-use driver only accepts the single value TOL. It then internally sets FTOL = XTOL = TOL and GTOL = 0.

The X-convergence condition (1) is a relative error test; it thus fails when $x^* = 0$ unless $x = 0$ also. Also note that if (1) is satisfied with $\mathrm{XTOL} = 10^{-k}$, then the larger components of Dx have k significant digits, but smaller components may not be as accurate. For example, if D is the identity matrix, XTOL = 0.001, and

$$x^* = (2.0,\ 0.003) ,$$

then

$$x = (2.001,\ 0.002)$$

satisfies (1), yet the second component of x has no significant digits. This may or may not be important. However, note that if instead

$$D = \mathrm{diag}(1,1000) ,$$

then (1) is not satisfied even for XTOL = 0.1. These scaling considerations can make it important to choose D carefully. See Section 2.5 for more information on scaling.

Since x^* is unknown, the actual criterion for X-convergence cannot be based on (1); instead it depends on the step bound Δ. That is, the actual convergence test is

$$\Delta \leq \mathrm{XTOL} \cdot \|Dx\| .$$

The F-convergence condition (2) is a relative error test; it thus fails when $F(x^*) = 0$ unless $F(x) = 0$ also. It is for this reason that F-convergence is not tested for systems of nonlinear equations where $F(x^*) = 0$ is the expected result. Also note that if (2) is satisfied with FTOL = 10^{-k}, then $\|F(x)\|$ has k significant digits, but x may not be as accurate. For example, if FTOL = 10^{-6} and

$$F(x) = \begin{pmatrix} x - 1 \\ 1 \end{pmatrix} ,$$

then $x^* = 1$, $\|F(x^*)\| = 1$, and if $x = 1.001$ then (2) is satisfied with FTOL = 10^{-6}, but (1) is only satisfied with XTOL = 10^{-3}.

In many least squares problems, if FTOL = $(\mathrm{XTOL})^2$ then X-convergence implies F-convergence. This result, however, does not hold if $\|F(x^*)\|$ is very small. For example, if

$$F(x) = \begin{pmatrix} x - 1 \\ 0.0001 \end{pmatrix} ,$$

then $x^* = 1$ and $\|F(x^*)\| = 0.0001$, but if $x = 1.001$ then (1) is satisfied with XTOL = 10^{-3} and yet

$$\|F(x)\| \geq 10\|F(x^*)\| .$$

Since $\|F(x^*)\|$ is unknown, the actual criterion for F-convergence cannot be literally (2); instead it is based on estimates of the terms in (2). If f and f_+ are the vectors of residuals at the current solution approximation x and at x+p, respectively, then the (relative) actual reduction is

$$\text{ACTRED} = \left(\|f\| - \|f_+\|\right)/\|f\| ,$$

while the (relative) predicted reduction is

$$\text{PRERED} = \left(\|f\| - \|f+Jp\|\right)/\|f\| .$$

The F-convergence test then requires that

$$\text{PRERED} \le \text{FTOL}$$
$$|\text{ACTRED}| \le \text{FTOL}$$
$$\text{ACTRED} \le 2 \cdot \text{PRERED}$$

all hold.

The X-convergence and F-convergence tests are quite reliable, but it is important to note that their validity depends critically on the correctness of the Jacobian. If the user is providing the Jacobian, he may make an error. (CHKDER can be used to check the Jacobian.) If the algorithm is estimating the Jacobian matrix, then the approximation may be incorrect if, for example, the function is subject to large errors and EPSFCN is chosen poorly. (For more details see Section 2.4.) In either case the algorithm usually terminates suspiciously near the starting point; recommended action if this occurs is to rerun the problem from a different starting point. If the algorithm also terminates near the new starting point, then it is very likely that the Jacobian is being determined incorrectly.

The X-convergence and F-convergence tests may also fail if the tolerances are too large. In general, XTOL and FTOL should be smaller than 10^{-5}; recommended values for these tolerances are on the order of the square root of the machine precision. As described in Section 1.7, the single precision value of the machine precision can be obtained from the MINPACK-1 function SPMPAR and the double precision value from DPMPAR. Note, however, that on some machines the square root of machine precision is larger than 10^{-5}.

The G-convergence test (3) measures the angle between the residual vector and the columns of the Jacobian matrix and thus can be expected to fail if either $F(x^*) = 0$ or any column of $F'(x^*)$ is zero. Also note that there is no clear relationship between G-convergence and either X-convergence or F-convergence. Furthermore, the G-convergence test detects other critical points, namely maximizers and saddle points; therefore, termination with G-convergence should be examined carefully.

An important property of the tests described above is that they are scale invariant. (See Section 2.5 for more details on scaling.) Scale invariance is a feature not shared by many other convergence tests. For example, the convergence test

$$(4) \qquad \|f\| \leq \text{AFTOL} ,$$

where AFTOL is a user-supplied tolerance, is not scale invariant, and this makes it difficult to choose an appropriate AFTOL. As an illustration of the difficulty with this test, consider the function

$$F(x) = (3x - 10)\exp(10x) .$$

On a computer with 15 decimal digits

$$|F(x^*)| \geq 1 ,$$

where x^* is the closest machine-representable number to 10/3, and thus a suitable AFTOL is not apparent.

If the user, however, wants to use (4) as a termination test, then he can do this by setting NPRINT positive in the call to the respective core subroutine. (See Section 2.9 for more information on NPRINT.) This provides him periodic opportunity, through subroutine FCN with IFLAG = 0, to affect the iteration sequence, and in this instance he might insert the following program segment into FCN.

```
      IF (IFLAG .NE. 0) GO TO 10
      FNORM = ENORM(LFVEC,FVEC)
      IF (FNORM .LE. AFTOL) IFLAG = -1
      RETURN
   10 CONTINUE
```

In this program segment it is assumed that LFVEC = N for systems of nonlinear equations and LFVEC = M for nonlinear least squares problems. It is also assumed that the MINPACK-1 function ENORM is declared to the precision of the computation.

2.4 Approximations to the Jacobian Matrix

If the user does not provide the Jacobian matrix, then the MINPACK-1 algorithms compute an approximation J. In the algorithms for nonlinear least squares problems, J is always determined by a forward difference approximation, while in the algorithms for systems of nonlinear equations, J is sometimes determined by a forward-difference approximation but more often by an update formula of Broyden. It is important to note that the update formula is also used in the algorithms for systems of nonlinear equations where the user is providing the Jacobian matrix, since the updating tends to improve the efficiency of the algorithms.

The forward-difference approximation to the j-th column of the Jacobian matrix can be written

$$\frac{F(x+h_j e_j) - F(x)}{h_j} , \tag{1}$$

where e_j is the j-th column of the identity matrix and h_j is the difference parameter. The choice of h_j depends on the precision of the function evaluations, which is specified in the MINPACK-1 algorithms by the parameter EPSFCN. To be specific,

$$h_j = (\text{EPSFCN})^{\frac{1}{2}} |x_j|$$

unless $x_j = 0$, in which case

$$h_j = (\mathrm{EPSFCN})^{\frac{1}{2}} .$$

In the easy-to-use drivers EPSFCN is set internally to the machine precision (see Section 1.7), since these subroutines assume that the functions can be evaluated accurately. In the core subroutines EPSFCN is a user-supplied parameter; if there are errors in the evaluations of the functions, then EPSFCN may need to be much larger than the machine precision. For example, if the specification of the function requires the numerical evaluation of an integral, then EPSFCN should probably be on the order of the tolerance in the integration routine.

One advantage of approximation (1) is that it is scale invariant. (See Section 2.5 for more details on scaling.) A disadvantage of (1) is that it assumes EPSFCN the same for each variable, for each component function of F, and for each vector x. These assumptions may make it difficult to determine a suitable value for EPSFCN. The user who is uncertain of an appropriate value of EPSFCN can run the algorithm with two or three values of EPSFCN and retain the value that gives the best results. In general, overestimates are better than underestimates.

The update formula of Broyden depends on the current approximation x, the correction p, and J. Since

$$F(x+p) - F(x) = \left[\int_0^1 F'(x+\theta p)\,d\theta \right] p ,$$

it is natural to ask that the approximation J_+ at x+p satisfy the equation

$$J_+ p = F(x+p) - F(x) ,$$

and among the possible choices be the one closest to J. To define an appropriate measure of distance, let D be the current diagonal scaling matrix and define the matrix norm

$$\|A\|_D = \left(\sum_{j=1}^{n} \left(\frac{\|a_j\|}{d_j} \right)^2 \right)^{\frac{1}{2}} ,$$

where $a_1, a_2, \ldots, a_n$ are the columns of A. It is now easy to verify that the solution of the problem

$$\min\{\|\tilde{J}-J\|_D : \tilde{J}p = F(x+p)-F(x)\} ,$$

is given by

$$J_+ = J + \frac{\left(F(x+p)-F(x)-Jp\right)(D^TDp)^T}{\|Dp\|^2} .$$

There are many properties of this formula that justify its use in algorithms for systems of nonlinear equations, but a discussion of these properties is beyond the scope of this work.

2.5 Scaling

Scale invariance is a desirable feature of an optimization algorithm. Algorithms for systems of nonlinear equations and nonlinear least squares problems are scale invariant if, given problems related by the change of scale

$$\tilde{F}(x) = \alpha F(D_V x)$$
$$\tilde{x}_o = D_V^{-1} x_o ,$$

where α is a positive scalar and D_V is a diagonal matrix with positive entries, the approximations x and $\tilde{x}$ generated by the algorithms satisfy

$$\tilde{x} = D_V^{-1} x .$$

Scale invariance is a natural requirement that can have a significant effect on the implementation and performance of an algorithm. To the user scale invariance means, in particular, that he can work with either problem and obtain equivalent results.

The core subroutines in MINPACK-1 are scale invariant provided that the initial choice of the scaling matrix satisfies

(1) $$\tilde{D}_o = \alpha D_V D_o ,$$

where D_o and $\tilde{D}_o$ are the initial scaling matrices of the respective problems defined by F and x_o and by $\tilde{F}$ and $\tilde{x}_o$. If the user of the core subroutines has requested internal scaling (MODE = 1), then the internal scaling matrix is set to

$$\mathrm{diag}(\|a_1\|,\|a_2\|,\ldots,\|a_n\|) ,$$

where a_i is the i-th column of the initial Jacobian approximation, and (1) holds. If the user has stipulated external scaling (MODE = 2), then the initial scaling matrix is specified by the contents of the array DIAG, and scale invariance is only achieved if the user's choice satisfies (1).

There are certain cases in which scale invariance may be lost, as when the Jacobian matrix at the starting point has a column of zeroes and internal scaling is requested. In this case D_0 would have a zero element and be singular, but this possibility is not catered to in the current implementation. Instead, the zero element is arbitrarily set to 1, preserving nonsingularity but giving up scale invariance. In practice, however, these cases seldom arise and scale invariance is usually maintained.

Our experience is that internal scaling is generally preferable for nonlinear least squares problems and external scaling for systems of nonlinear equations. This experience is reflected in the settings built into the easy-to-use drivers; MODE = 1 is specified in the drivers for nonlinear least squares problems and MODE = 2 for systems of nonlinear equations. In the latter case, D_0 is set to the identity matrix, a choice that generally works out well in practice; if this choice is not appropriate, recourse to the core subroutine would be indicated.

It is important to note that scale invariance does not relieve the user of choosing an appropriate formulation of the problem or a reasonable starting point. In particular, note that an appropriate formulation may involve a scaling of the equations or a nonlinear transformation of the variables and that the performance of the MINPACK-1 algorithms can be affected by these transformations. For example, the algorithm for systems of nonlinear equations usually generates different approximations for problems defined by functions $\tilde{F}$ and F, where

$$\tilde{F}(x) = D_E F(x) ,$$
$$\tilde{x}_0 = x_0 ,$$

and D_E is a diagonal matrix with positive entries. The main reason for this is that the algorithm usually decides that x_+ is a better approximation than x if

$$\|F(x_+)\| < \|F(x)\| ,$$

and it is entirely possible that

$$\|\tilde{F}(x_+)\| > \|\tilde{F}(x)\| .$$

The user should thus scale his equations (i.e., choose D_E) so that the expected errors in the residuals are of about the same order of magnitude.

2.6 Subroutine FCN: Calculation of the Function and Jacobian Matrix

The MINPACK-1 algorithms require that the user provide a subroutine with name of his choosing, say FCN, to calculate the residuals of the function $F: R^n \to R^m$, where $m = n$ for systems of nonlinear equations and $m \geq n$ for nonlinear least squares problems. Some of the algorithms also require that FCN calculate the Jacobian matrix of the mapping F.

It is important that the calculation of the function and Jacobian matrix be as accurate as possible. It is also important that the coding of FCN be as efficient as possible, since the timing of the algorithm is strongly influenced by the time spent in FCN. In particular, when the residuals f_i have common subexpressions it is usually worthwhile to organize the computation so that these subexpressions need be evaluated only once. For example, if the residuals are of the form

$$f_i(x) = g(x) + h_i(x) , \quad 1 \leq i \leq m$$

with g(x) common to all of them, then the coding of FCN is best expressed in the following form.

```
$\tau = g(x)$
For $i = 1,2,\ldots,m$
    $f_i(x) = \tau + h_i(x)$ .
```

As another example, assume that the residuals are of the form

$$f_i(x) = \sum_{j=1}^{n} \left(\alpha_{ij}\cos(x_j) + \beta_{ij}\sin(x_j)\right) ,$$

where the α_{ij} and β_{ij} are given constants. The following program segment evaluates the f_i efficiently.

```
For $i = 1,2,\ldots,m$
    $f_i(x) = 0$
For $j = 1,2,\ldots,n$
    $\gamma = \cos(x_j)$
    $\sigma = \sin(x_j)$
    For $i = 1,2,\ldots,m$
        $f_i(x) = f_i(x) + \gamma\alpha_{ij} + \sigma\beta_{ij}$ .
```

If the user is providing the Jacobian matrix of the mapping F, then it is important that its calculation also be as efficient as possible. In particular, when the elements of the Jacobian matrix have common subexpressions, it is usually worthwhile to organize the computation so that these subexpressions need be evaluated only once. For example, if

$$f_i(x) = g(x) + h_i(x) , \qquad 1 \le i \le m ,$$

then the rows of the Jacobian matrix are

$$\nabla f_i(x) = \nabla g(x) + \nabla h_i(x) , \qquad 1 \le i \le m ,$$

and the subexpression $\nabla g(x)$ is thus common to all the rows of the Jacobian matrix.

As another example, assume that

$$f_i(x) = \sum_{j=1}^{n} \left(\alpha_{ij}\cos(x_j) + \beta_{ij}\sin(x_j)\right) ,$$

where the α_{ij} and β_{ij} are given constants. In this case,

$$\frac{\partial f_i(x)}{\partial x_j} = - \alpha_{ij}\sin(x_j) + \beta_{ij}\cos(x_j) ,$$

and the following program segment evaluates the Jacobian matrix efficiently.

```
For j = 1,2,...,n
    $\gamma = \cos(x_j)$
    $\sigma = \sin(x_j)$
    For i = 1,2,...,m
        $\frac{\partial f_i(x)}{\partial x_j} = -\sigma\alpha_{ij} + \gamma\beta_{ij}$ .
```

The previous example illustrates further the possibility of common sub-expressions between the function and the Jacobian matrix. For the nonlinear least squares algorithms advantage can be taken of this, because a call to FCN to evaluate the Jacobian matrix at x is always preceded by a call to evaluate the function at x. This is not the case for the nonlinear equations algorithms.

To specifically illustrate this possibility of sharing information between function and Jacobian matrix, assume that

$$f_i(x) = g(x)^2 + h_i(x) , \qquad 1 \leq i \leq m .$$

Then the rows of the Jacobian matrix are

$$\nabla f_i(x) = 2g(x)\nabla g(x) + \nabla h_i(x) , \qquad 1 \leq i \leq m ,$$

and the coding of FCN is best done as follows.

```
If FUNCTION EVALUATION then
    $\tau = g(x)$
    Save $\tau$ in COMMON
    For i = 1,2,...,m
        $f_i(x) = \tau^2 + h_i(x)$
If JACOBIAN EVALUATION then
    $v = \nabla g(x)$
    For i = 1,2,...,m
        $\nabla f_i(x) = 2\tau v + \nabla h_i(x)$ .
```

2.7 Constraints

Systems of nonlinear equations and nonlinear least squares problems often impose constraints on the solution. For example, on physical grounds it is sometimes necessary that the solution vector have positive components.

At present there are no algorithms in MINPACK that formally admit constraints, but in some cases they can be effectively achieved with ad hoc strategies. In this section we describe two strategies for restricting the solution approximations to a region **D** of R^n.

The user has control over the initial approximation x_0. It may happen, however, that x is in **D** but the algorithm computes a correction p such that x+p is not in **D**. If this correction is permitted, the algorithm may never recover; that is, the approximations may now converge to an unacceptable solution outside of **D**.

The simplest strategy to restrict the corrections is to impose a penalty on the function if the algorithm attempts to step outside of **D**. For example, let μ be any number such that

$$|f_i(x_0)| \leq \mu , \qquad 1 \leq i \leq m ,$$

and in FCN define

$$f_i(x) = \mu , \qquad 1 \leq i \leq m$$

whenever x does not belong to **D**. If FCN is coded in this way, a correction p for which x+p lies outside of **D** will not decrease the residuals and is therefore not acceptable. It follows that this penalty on FCN forces all the approximations x to lie in **D**.

Note that this strategy restricts all the corrections, and as a consequence may lead to very slow convergence if the solution is near the boundary of **D**. It usually suffices to only restrict the initial correction, and users of the core subroutines can do this in several ways.

Recall from Section 2.2 that the initial correction p_0 satisfies a bound of the form

$$\|D_o p_o\| \leq \Delta_o ,$$

where D_o is a diagonal scaling matrix and Δ_o is a step bound. The contents of D_o are governed by the parameter MODE. If MODE = 1 then D_o is internally set, while if MODE = 2 then D_o is specified by the user through the array DIAG. The step bound Δ_o is determined from the parameter FACTOR. By definition

$$\Delta_o = \text{FACTOR}\cdot\|D_o x_o\| ,$$

unless x_o is the zero vector, in which case

$$\Delta_o = \text{FACTOR} .$$

It is clear from this definition that smaller values of FACTOR lead to smaller steps. For a sufficiently small value of FACTOR (usually 0.01 suffices), an improved point x_o+p_o will be found that belongs to **D**.

Be aware that the step restriction is on $D_o p_o$ and not on p_o directly. A small element of D_o, which can be set by internal scaling when MODE = 1, may lead to a large component in the correction p_o. In many cases it is not necessary to control p_o directly, but if this is desired then MODE = 2 must be used.

When MODE = 2, the contents of D_o are specified by the user, and this allows direct control of p_o. If, for example, it is desired to restrict the components of p_o to small relative corrections of the corresponding components of x_o (assumed nonzero), then this can be done by setting

$$D_o = \operatorname{diag}\left(\frac{1}{|\xi_1|}, \frac{1}{|\xi_2|}, \ldots, \frac{1}{|\xi_n|}\right) ,$$

where ξ_i is the i-th component of x_o, and by choosing FACTOR appropriately. To justify this choice, note that p_o satisfies

$$\|D_o p_o\| \leq \Delta_o = \text{FACTOR}\cdot\|D_o x_o\| ,$$

and that the choice of D_o guarantees that

$$\|D_o x_o\| = n^{\frac{1}{2}} .$$

Thus, if ρ_i is the i-th component of p_o, then

$$|\rho_i| \leq n^{\frac{1}{2}} \cdot \text{FACTOR} \cdot |\xi_i| ,$$

which justifies the choice of D_o.

2.8 Error Bounds

A problem of general interest is the determination of error bounds on the components of a solution vector. It is beyond the scope of this work to discuss this topic in depth, so the discussion below is limited to the computation of bounds on the sensitivity of the parameters, and of the covariance matrix. The discussion is in terms of the nonlinear least squares problem, but some of the results also apply to systems of nonlinear equations.

Let $F: R^n \to R^m$ define a nonlinear least squares problem $(m \geq n)$, and let x^* be a solution. Given $\varepsilon > 0$, the problem is to determine sensitivity (upper) bounds $\sigma_1, \sigma_2, \ldots, \sigma_n$ such that, for each i, the condition

$$|x_i - x_i^*| \leq \sigma_i , \qquad \text{with } x_j = x_j^* \text{ for } j \neq i ,$$

implies that

$$\|F(x)\| \leq (1 + \varepsilon)\|F(x^*)\| .$$

Of particular interest are values of σ_i which are large relative to $|x_i|$, since then the residual norm $\|F(x)\|$ is insensitive to changes in the i-th parameter and may therefore indicate a possible deficiency in the formulation of the problem.

A first order estimate of the sensitivity bounds σ_i shows that

(1) $$\sigma_i = \varepsilon^{\frac{1}{2}} \left(\frac{\|F(x^*)\|}{\|F'(x^*) \cdot e_i\|} \right) ,$$

where $F'(x^*)$ is the Jacobian matrix of F at x^* and e_i is the i-th column of the identity matrix. Note that if $\|F'(x^*) \cdot e_i\|$ is small relative to $\|F(x^*)\|$, then the residual norm is insensitive to changes in the i-th parameter.

If x is an approximation to the solution x* and J is an approximation to F'(x*), then the bounds (1) can usually be replaced by

$$(2) \qquad \sigma_i = \varepsilon^{\frac{1}{2}}\left(\frac{\|F(x)\|}{\|Je_i\|}\right) .$$

The MINPACK-1 nonlinear least squares programs (except LMDIF1) return enough information to compute the sensitivity bounds (2). On a normal exit, these programs return F(x) and part of the QR decomposition of J; namely, an upper triangular matrix R and a permutation matrix P such that

$$(3) \qquad JP = QR$$

for some matrix Q with orthogonal columns. The vector F(x) is returned in the array FVEC and the matrix R is returned in the upper triangular part of the array FJAC. The permutation matrix P is defined by the contents of the integer array IPVT; if

$$\text{IPVT} = \big(p(1),p(2),\ldots,p(n)\big) ,$$

then the j-th column of P is the p(j)-th column of the identity matrix.

The norms of the columns of the Jacobian matrix can be computed by noting that (3) implies that

$$Je_{p(j)} = QRe_j ,$$

and hence,

$$\|Je_{p(j)}\| = \|Re_j\| .$$

The following loop uses this relationship to store $\|Je_\ell\|$ in the ℓ-th position of an array FJNORM; with this information it is then easy to compute the sensitivity bounds (2).

```
      DO 10 J = 1, N
         L = IPVT(J)
         FJNORM(L) = ENORM(J,FJAC(1,J))
   10    CONTINUE
```

This loop assumes that ENORM and FJNORM have been declared to the precision of the computation.

In addition to sensitivity bounds for the individual parameters, it is sometimes desirable to determine a bound for the sensitivity of the residual norm to changes in some linear combination of the parameters. Given $\varepsilon > 0$ and a vector v with $\|v\| = 1$, the problem is to determine a bound σ such that

$$\|F(x^*+\sigma v)\| \leq (1 + \varepsilon)\|F(x^*)\| .$$

A first order estimate of σ is now

$$\sigma = \varepsilon^{\frac{1}{2}}\left(\frac{\|F(x^*)\|}{\|F'(x^*)\cdot v\|}\right) ;$$

if $\|F'(x^*)\cdot v\|$ is small relative to $\|F(x^*)\|$, then σ is large and the residual norm is insensitive to changes in the linear combination of the parameters specified by v.

For example, if the level set

$$\{x: \ \|F(x)\| \leq (1 + \varepsilon)\|F(x^*)\|\}$$

is as in Figure 3, then the residual norm, although sensitive to changes in x_1 and x_2, is relatively insensitive to changes along $v = (1,1)$.

If the residual norm is relatively insensitive to changes in some linear combination of the parameters, then the Jacobian matrix at the solution is nearly rank-deficient, and in these cases it may be worthwhile to attempt to determine a set of linearly independent parameters. In some statistical applications, the covariance matrix

$$(J^TJ)^{-1}$$

is used for this purpose.

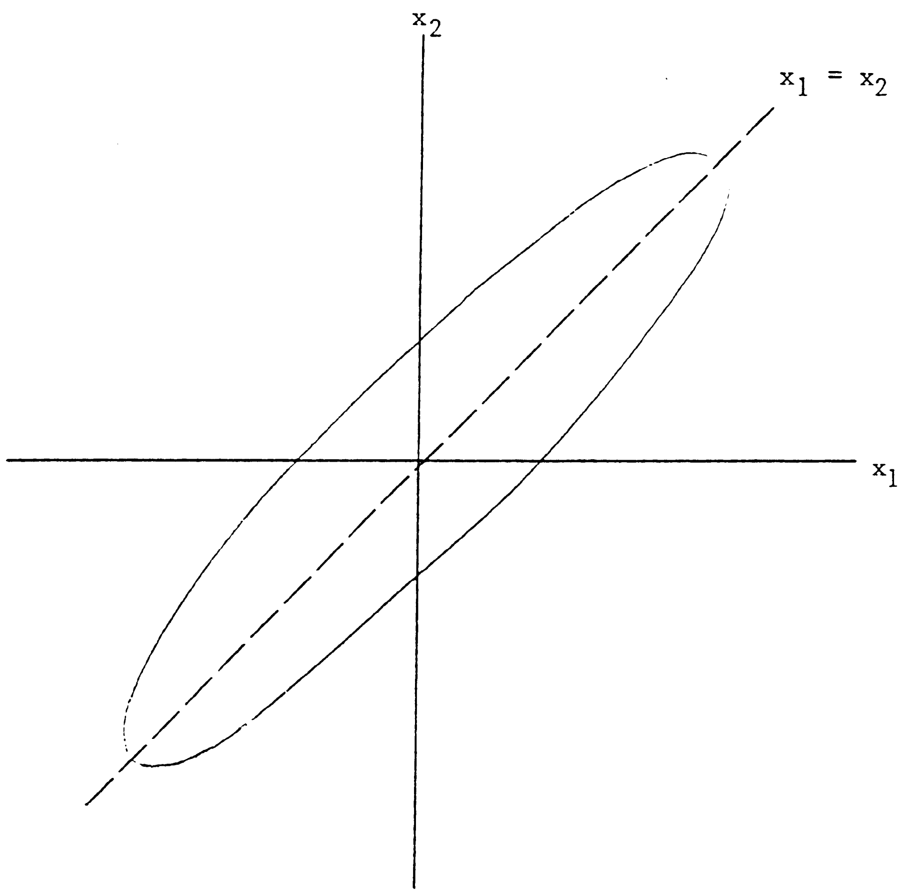

Figure 3

Subroutine COVAR, which appears at the end of this section, will compute the covariance matrix. The computation of the covariance matrix from the QR factorization of J depends on the relationship

$$(J^TJ)^{-1} = P(R^TR)^{-1}P^T \quad , \tag{4}$$

which is an easy consequence of (3). Subroutine COVAR overwrites R with the upper triangular part of $(R^TR)^{-1}$ and then computes the covariance matrix from (4).

Note that for proper execution of COVAR the QR factorization of J must have used column pivoting. This guarantees that for the resulting R

$$|r_{kk}| \geq |r_{ij}| \quad , \qquad k \leq i \leq j \quad , \tag{5}$$

thereby allowing a reasonable determination of the numerical rank of J. Most of the MINPACK-1 nonlinear least squares subroutines return the correct factorization; the QR factorization in LMSTR1 and LMSTR, however, satisfies

$$JP_1 = Q_1R_1$$

but R_1 does not usually satisfy (5). To obtain the correct factorization, note that the QR factorization with column pivoting of R_1 satisfies

$$R_1P_2 = Q_2R_2$$

where R_2 satisfies (5), and therefore

$$J(P_1P_2) = (Q_1Q_2)R_2$$

is the desired factorization of J. The program segment below uses the MINPACK-1 subroutine QRFAC to compute R_2 from R_1.

```
         DO 30 J = 1, N
            JP1 = J + 1
            IF (N .LT. JP1) GO TO 20
            DO 10 I = JP1, N
               FJAC(I,J) = ZERO
   10          CONTINUE
   20       CONTINUE
   30    CONTINUE
         CALL QRFAC(N,N,FJAC,LDFJAC,.TRUE.,IPVT2,N,WA1,WA2,WA3)
         DO 40 J = 1, N
            FJAC(J,J) = WA1(J)
            L = IPVT2(J)
            IPVT2(J) = IPVT1(L)
   40    CONTINUE
```

Note that QRFAC sets the contents of the array IPVT2 to define the permutation matrix P_2, and the final loop in the program segment overwrites IPVT2 to define the permutation matrix P_1P_2.

```
      SUBROUTINE COVAR(N,R,LDR,IPVT,TOL,WA)
      INTEGER N,LDR
      INTEGER IPVT(N)
      DOUBLE PRECISION TOL
      DOUBLE PRECISION R(LDR,N),WA(N)
C     ***********
C
C     SUBROUTINE COVAR
C
C     GIVEN AN M BY N MATRIX A, THE PROBLEM IS TO DETERMINE
C     THE COVARIANCE MATRIX CORRESPONDING TO A, DEFINED AS
C
C                    T
C           INVERSE(A *A) .
C
C     THIS SUBROUTINE COMPLETES THE SOLUTION OF THE PROBLEM
C     IF IT IS PROVIDED WITH THE NECESSARY INFORMATION FROM THE
C     QR FACTORIZATION, WITH COLUMN PIVOTING, OF A. THAT IS, IF
C     A*P = Q*R, WHERE P IS A PERMUTATION MATRIX, Q HAS ORTHOGONAL
C     COLUMNS, AND R IS AN UPPER TRIANGULAR MATRIX WITH DIAGONAL
C     ELEMENTS OF NONINCREASING MAGNITUDE, THEN COVAR EXPECTS
C     THE FULL UPPER TRIANGLE OF R AND THE PERMUTATION MATRIX P.
C     THE COVARIANCE MATRIX IS THEN COMPUTED AS
C
C                      T     T
C           P*INVERSE(R *R)*P  .
C
C     IF A IS NEARLY RANK DEFICIENT, IT MAY BE DESIRABLE TO COMPUTE
C     THE COVARIANCE MATRIX CORRESPONDING TO THE LINEARLY INDEPENDENT
C     COLUMNS OF A. TO DEFINE THE NUMERICAL RANK OF A, COVAR USES
C     THE TOLERANCE TOL. IF L IS THE LARGEST INTEGER SUCH THAT
C
C           ABS(R(L,L)) .GT. TOL*ABS(R(1,1)) ,
C
C     THEN COVAR COMPUTES THE COVARIANCE MATRIX CORRESPONDING TO
C     THE FIRST L COLUMNS OF R. FOR K GREATER THAN L, COLUMN
C     AND ROW IPVT(K) OF THE COVARIANCE MATRIX ARE SET TO ZERO.
C
C     THE SUBROUTINE STATEMENT IS
C
C       SUBROUTINE COVAR(N,R,LDR,IPVT,TOL,WA)
C
C     WHERE
C
C       N IS A POSITIVE INTEGER INPUT VARIABLE SET TO THE ORDER OF R.
C
C       R IS AN N BY N ARRAY. ON INPUT THE FULL UPPER TRIANGLE MUST
C         CONTAIN THE FULL UPPER TRIANGLE OF THE MATRIX R. ON OUTPUT
C         R CONTAINS THE SQUARE SYMMETRIC COVARIANCE MATRIX.
C
C       LDR IS A POSITIVE INTEGER INPUT VARIABLE NOT LESS THAN N
C         WHICH SPECIFIES THE LEADING DIMENSION OF THE ARRAY R.
C
C       IPVT IS AN INTEGER INPUT ARRAY OF LENGTH N WHICH DEFINES THE
```

```
C         PERMUTATION MATRIX P SUCH THAT A*P = Q*R. COLUMN J OF P     COVR0550
C         IS COLUMN IPVT(J) OF THE IDENTITY MATRIX.                   COVR0560
C                                                                     COVR0570
C       TOL IS A NONNEGATIVE INPUT VARIABLE USED TO DEFINE THE        COVR0580
C         NUMERICAL RANK OF A IN THE MANNER DESCRIBED ABOVE.          COVR0590
C                                                                     COVR0600
C       WA IS A WORK ARRAY OF LENGTH N.                               COVR0610
C                                                                     COVR0620
C     SUBPROGRAMS CALLED                                              COVR0630
C                                                                     COVR0640
C       FORTRAN-SUPPLIED ... DABS                                     COVR0650
C                                                                     COVR0660
C     ARGONNE NATIONAL LABORATORY. MINPACK PROJECT. AUGUST 1980.      COVR0670
C     BURTON S. GARBOW, KENNETH E. HILLSTROM, JORGE J. MORE           COVR0680
C                                                                     COVR0690
C     **********                                                      COVR0700
      INTEGER I,II,J,JJ,K,KM1,L                                       COVR0710
      LOGICAL SING                                                    COVR0720
      DOUBLE PRECISION ONE,TEMP,TOLR,ZERO                             COVR0730
      DATA ONE,ZERO /1.0D0,0.0D0/                                     COVR0740
C                                                                     COVR0750
C     FORM THE INVERSE OF R IN THE FULL UPPER TRIANGLE OF R.          COVR0760
C                                                                     COVR0770
      TOLR = TOL*DABS(R(1,1))                                         COVR0780
      L = 0                                                           COVR0790
      DO 40 K = 1, N                                                  COVR0800
         IF (DABS(R(K,K)) .LE. TOLR) GO TO 50                         COVR0810
         R(K,K) = ONE/R(K,K)                                          COVR0820
         KM1 = K - 1                                                  COVR0830
         IF (KM1 .LT. 1) GO TO 30                                     COVR0840
         DO 20 J = 1, KM1                                             COVR0850
            TEMP = R(K,K)*R(J,K)                                      COVR0860
            R(J,K) = ZERO                                             COVR0870
            DO 10 I = 1, J                                            COVR0880
               R(I,K) = R(I,K) - TEMP*R(I,J)                          COVR0890
   10          CONTINUE                                               COVR0900
   20       CONTINUE                                                  COVR0910
   30    CONTINUE                                                     COVR0920
         L = K                                                        COVR0930
   40    CONTINUE                                                     COVR0940
   50 CONTINUE                                                        COVR0950
C                                                                     COVR0960
C     FORM THE FULL UPPER TRIANGLE OF THE INVERSE OF (R TRANSPOSE)*R  COVR0970
C     IN THE FULL UPPER TRIANGLE OF R.                                COVR0980
C                                                                     COVR0990
      IF (L .LT. 1) GO TO 110                                         COVR1000
      DO 100 K = 1, L                                                 COVR1010
         KM1 = K - 1                                                  COVR1020
         IF (KM1 .LT. 1) GO TO 80                                     COVR1030
         DO 70 J = 1, KM1                                             COVR1040
            TEMP = R(J,K)                                             COVR1050
            DO 60 I = 1, J                                            COVR1060
               R(I,J) = R(I,J) + TEMP*R(I,K)                          COVR1070
   60          CONTINUE                                               COVR1080
```

```
  70          CONTINUE                                                 COVR1090
  80       CONTINUE                                                    COVR1100
           TEMP = R(K,K)                                               COVR1110
           DO 90 I = 1, K                                              COVR1120
              R(I,K) = TEMP*R(I,K)                                     COVR1130
  90          CONTINUE                                                 COVR1140
 100       CONTINUE                                                    COVR1150
 110    CONTINUE                                                       COVR1160
C                                                                      COVR1170
C       FORM THE FULL LOWER TRIANGLE OF THE COVARIANCE MATRIX          COVR1180
C       IN THE STRICT LOWER TRIANGLE OF R AND IN WA.                   COVR1190
C                                                                      COVR1200
        DO 130 J = 1, N                                                COVR1210
           JJ = IPVT(J)                                                COVR1220
           SING = J .GT. L                                             COVR1230
           DO 120 I = 1, J                                             COVR1240
              IF (SING) R(I,J) = ZERO                                  COVR1250
              II = IPVT(I)                                             COVR1260
              IF (II .GT. JJ) R(II,JJ) = R(I,J)                        COVR1270
              IF (II .LT. JJ) R(JJ,II) = R(I,J)                        COVR1280
 120          CONTINUE                                                 COVR1290
           WA(JJ) = R(J,J)                                             COVR1300
 130       CONTINUE                                                    COVR1310
C                                                                      COVR1320
C       SYMMETRIZE THE COVARIANCE MATRIX IN R.                         COVR1330
C                                                                      COVR1340
        DO 150 J = 1, N                                                COVR1350
           DO 140 I = 1, J                                             COVR1360
              R(I,J) = R(J,I)                                          COVR1370
 140          CONTINUE                                                 COVR1380
           R(J,J) = WA(J)                                              COVR1390
 150       CONTINUE                                                    COVR1400
        RETURN                                                         COVR1410
C                                                                      COVR1420
C       LAST CARD OF SUBROUTINE COVAR.                                 COVR1430
C                                                                      COVR1440
        END                                                            COVR1450
```

2.9 Printing

No printing is done in any of the MINPACK-1 subroutines. However, printing of certain parameters through FCN can be facilitated with the integer parameter NPRINT that is available to users of the core subroutines. For these subroutines, setting NPRINT positive results in special calls to FCN with IFLAG = 0 at the beginning of the first iteration and every NPRINT iterations thereafter and immediately prior to return. On these calls to FCN, the parameters X and FVEC are available for printing; FJAC is additionally available if using LMDER.

Often it suffices to print some simple measure of the iteration progress, and the Euclidean norm of the residuals is usually a good choice. This norm can be printed by inserting the following program segment into FCN.

```
      IF (IFLAG .NE. 0) GO TO 10
      FNORM = ENORM(LFVEC,FVEC)
      WRITE (---,1000) FNORM
 1000 FORMAT (---)
      RETURN
   10 CONTINUE
```

In this program segment it is assumed that LFVEC = N for systems of nonlinear equations and LFVEC = M for nonlinear least squares problems. It is also assumed that the MINPACK-1 function ENORM is declared to the precision of the computation.

CHAPTER 3
Notes and References

This chapter provides notes relating the MINPACK-1 algorithms and software to other work. The list of references appears at the end.

Powell's Hybrid Method

The MINPACK-1 version of Powell's [1970] hybrid method differs in many respects from the original version. For example, the "special iterations" used in the original algorithm proved to be inefficient and have been replaced. The updating method used is due to Broyden [1965]; the MINPACK-1 algorithm is a scaled version of the original. A comparison of an earlier version of the MINPACK-1 algorithm with other algorithms for systems of nonlinear equations has been made by Hiebert [1980].

The Levenberg-Marquardt Algorithm

There are many versions of the algorithm proposed by Levenberg [1944] and modified by Marquardt [1963]. An advantage of the MINPACK-1 version is that it avoids the difficulties associated with choosing the Levenberg-Marquardt parameter, and this allows a very strong global convergence result. The MINPACK-1 algorithm is based on the work of Hebden [1973] and follows the ideas of Moré [1977]. A comparison of an earlier version of the MINPACK-1 algorithm with other algorithms for nonlinear least squares problems has been made by Hiebert [1979].

Derivative Checking

Subroutine CHKDER is new, but similar routines exist in the Numerical Algorithms Group (NAG) library. An advantage of CHKDER is its generality; it can be used to check Jacobians, gradients, and Hessians (second derivatives). To enable this generality, CHKDER presumes no specific parameter sequence for the function evaluation program, returning control instead to the user. This in turn makes necessary a second call to CHKDER for each check.

MINPACK-1 Internal Subprograms

Subroutines DOGLEG and LMPAR are used to generate search directions in the algorithms for systems of nonlinear equations and nonlinear least squares problems, respectively. The algorithm used in DOGLEG is a fairly straightforward implementation of the ideas of Powell [1970], while LMPAR is a refined version of the algorithm described by Moré [1977]. The LMPAR algorithm is the more complicated; in particular, it requires the solution of a sequence of linear least squares problems of special form. It is for this purpose that subroutine QRSOLV is used.

The algorithm used in ENORM is a simplified version of Blue's [1978] algorithm. An advantage of the MINPACK-1 version is that it does not require machine constants; a disadvantage is that nondestructive underflows are allowed.

The banded Jacobian option in FDJAC1 is based on the work of Curtis, Powell, and Reid [1974].

QRFAC and RWUPDT are based on the corresponding algorithms in LINPACK (Dongarra, Bunch, Moler, and Stewart [1979]).

The algorithm used in R1UPDT is based on the work of Gill, Golub, Murray, and Saunders [1974].

References

Blue, J. L. [1978]. A portable Fortran program to find the Euclidean norm of a vector, ACM Transactions on Mathematical Software 4, 15-23.

Boyle, J. M. and Dritz, K. W. [1974]. An automated programming system to facilitate the development of quality mathematical software, Proceedings IFIP Congress, North-Holland.

Broyden, C. G. [1965]. A class of methods for solving nonlinear simultaneous equations, Math. Comp. 19, 577-593.

Curtis, A. R., Powell, M. J. D., and Reid, J. K. [1974]. On the estimation of sparse Jacobian matrices, J. Inst. Maths Applics 13, 117-119.

Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W. [1979]. LINPACK users' guide, SIAM Publications.

Fosdick, L. D. [1974]. BRNANL, A Fortran program to identify basic blocks in Fortran programs, University of Colorado, Computer Science report CU-CS-040-74.

Fox, P. A., Hall, A. D., and Schryer, N. L. [1978]. The PORT mathematical subroutine library, ACM Transactions on Mathematical Software 4, 104-126.

Garbow, B. S., Hillstrom, K. E., and Moré, J. J. [1980]. Implementation guide for MINPACK-1, Argonne National Laboratory report ANL-80-68.

Gill, P. E., Golub, G. H., Murray, W., and Saunders, M. A. [1974]. Methods for modifying matrix factorizations, Math. Comp. 28, 505-535.

Hebden, M. D. [1973]. An algorithm for minimization using exact second derivatives, Atomic Energy Research Establishment report TP 515, Harwell, England.

Hiebert, K. L. [1979]. A comparison of nonlinear least squares software, Sandia Laboratories report SAND 79-0483, Albuquerque, New Mexico.

Hiebert, K. L. [1980]. A comparison of software which solves systems of nonlinear equations, Sandia Laboratories report SAND 80-0181, Albuquerque, New Mexico.

Levenberg, K. [1944]. A method for the solution of certain nonlinear problems in least squares, Quart. Appl. Math. 2, 164-168.

Marquardt, D. W. [1963]. An algorithm for least-squares estimation of nonlinear parameters, SIAM J. Appl. Math. 11, 431-441.

Moré, J. J. [1977]. The Levenberg-Marquardt algorithm: Implementation and Theory, Numerical Analysis, G. A. Watson, ed., Lecture Notes in Mathematics 630, Springer-Verlag.

Moré, J. J., Garbow, B. S., and Hillstrom, K. E. [1978]. Testing unconstrained optimization software, Argonne National Laboratory, Applied Mathematics Division Technical Memorandum 324 (to appear in ACM Transactions on Mathematical Software).

Powell, M. J. D. [1970]. A hybrid method for nonlinear equations, in Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, ed., Gordon and Breach.

Ryder, B. G. [1974]. The PFORT verifier, Software Practice and Experience 4, 359-377.