

AN ACCELERATED LEVENBERG-MARQUARDT
ALGORITHM FOR NONLINEAR
LEAST SQUARES PROBLEMS

By

WEI YUAN

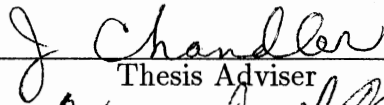
Bachelor of Science
Hunan Normal University
Changsha, Hunan, P.R. of China
1981

Master of Science
Huazhong University of Science and Technology
Wuhan, Hubei, P.R. of China
1984

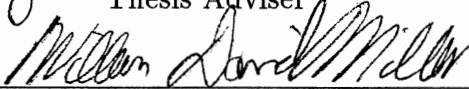
Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
July, 1992

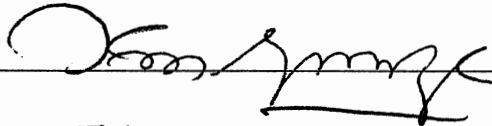
AN ACCELERATED LEVENBERG MARQUARDT
ALGORITHM FOR NONLINEAR
LEAST SQUARES PROBLEMS

Thesis Approved:



Thesis Adviser





Thomas C. Collins

Dean of the Graduate College

PREFACE

This paper presents modifications of the Levenberg-Marquardt method for solving nonlinear least squares problems. This is of practical importance in various fields such as curve-fitting and parameter estimation.

The modifications are made in such a way that, (1) Ruhe's conjugate gradient acceleration is used along with the Levenberg-Marquardt search direction; (2) A modified version of Al-Baali and Fletcher's line search scheme is incorporated into the algorithm in which polynomial interpolations are made to the individual residual functions rather than the overall objective function; (3) Accuracy of the line search is controlled dynamically by letting the accuracy parameters be varied as the norms of the function and gradient change. The algorithm has been implemented using FORTRAN 77, and compared with some other nonlinear least squares codes. Numerical results on a set of test problems indicate that it is efficient as well as robust.

I would like to express my sincere gratitude to my major adviser, Dr. John P. Chandler, who introduced me to this interesting subject, and constantly gave me intelligent guidance. Many thanks also to the other committee members, Dr. K. M. George and Dr. William D. Miller, for their helpful advisement and suggestions.

Special thanks are due to my wife, Guangping Lei, for her moral support and technical assistance with expert L^AT_EX skills during the thesis preparation. My deepest appreciation is extended to my parents whose encouragement and understanding were invaluable throughout the study.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
II. LITERATURE REVIEW	4
III. METHODOLOGY	9
Search Direction	10
Size of the Trust Region	11
The Levenberg-Marquardt Parameter	12
Using Ruhe's <i>c-g</i> Acceleration	13
Line Search	15
Al-Baali and Fletcher's Line Search Scheme	15
Accuracy of the Line Search	18
Size of the Search Interval	19
Initial Steplength	20
IV. NUMERICAL EXPERIMENTS	22
Test Problems	22
Results and Analysis	24
V. CONCLUSIONS	35
A SELECTED BIBLIOGRAPHY	37
APPENDIX A – TEST FUNCTIONS	41
APPENDIX B – PROGRAM LISTING	49

LIST OF TABLES

Table	Page
I. One Line Search for Rosenbrock's Function.	18
II. Summary of 54 Calls to ALML	27
III. Summary of 54 Calls to LMDER	28
IV. Summary of 54 Calls to MARQ (GN).	29
V. Summary of 54 Calls to MARQ (Mod. GN)	30
VI. Summary of 54 Calls to MARQ (Marquardt)	31
VII. Summary of 54 Calls to MARQ (Mod. Marquardt)	32
VIII. Overall Performance Data	33
IX. SUMMARY OF 11 CALLS TO ALML ($\varepsilon_{noise} = 10$)	34
X. Data for Function 8	43
XI. Data for Function 9	44
XII. Data for Function 10.	45
XIII. Data for Function 17.	47
XIV. Data for Function 18.	48

LIST OF FIGURES

Figure	Page
1. One Line Search for Rosenbrock's Function	18

CHAPTER I

INTRODUCTION

This thesis studies nonlinear least squares problems that are to minimize the sum of squares of nonlinear functions of one or more real variables. Nonlinear least squares problems are of practical importance, and typically occur in the field of curve-fitting where a set of m data points

$$\{(t_k, y_k) | k = 1, \dots, m\}$$

and a modeling function $f(t_k, x)$ are given where x is an n -dimensional parameter vector to be determined such that

$$S(x) = \sum_{k=1}^m (y_k - f(t_k, x))^2$$

is minimized (Gauss, 1873). The error $y_k - f(t_k, x)$ is usually called the k -th residual.

Mathematically, the nonlinear least squares problem can be formulated as

$$\min_{x \in \mathbb{R}^n} S(x) = \frac{1}{2} r^t(x) r(x) \quad (1)$$

where $r(x) = (r_1(x), \dots, r_m(x))^t \in \mathbb{R}^m$, $m \geq n$. When $m = n$, the problem can be posed as a system of nonlinear equations

$$r_k(x) = 0, \quad k = 1, \dots, m.$$

The gradient vector $g(x)$ and the Hessian matrix $H(x)$ of $S(x)$,

$$\begin{aligned} g(x) &= \nabla S(x) = J(x)^t r(x), \\ H(x) &= \nabla^2 S(x) = J(x)^t J(x) + \sum_{k=1}^m r_k(x) \nabla^2 r_k(x) \end{aligned} \quad (2)$$

have a special form which can be exploited by algorithms; here $J(x)$ is the $m \times n$ Jacobian matrix of $r(x)$, whose (i, j) element is $\partial r_i / \partial x_j$; and $\nabla^2 r_k(x)$ is the $n \times n$ Hessian matrix of $r_k(x)$, whose (i, j) element is $\partial^2 r_k(x) / \partial x_i \partial x_j$.

Algorithms for solving nonlinear least squares problems have found wide application in various areas such as physics, chemistry, engineering, statistics and lens design, etc., and can be divided into the following two categories:

- (i) general purpose nonlinear unconstrained optimization methods which ignore the special form of the object function;
- (ii) special methods which take into account the special structure of the problem under consideration.

Generally speaking, the specialized methods are superior to the general purpose methods since they take advantage of the special structure of the problem. It should be noted that even if a method from the proper category converges on a given problem, it may diverge or converge too slowly on another problem. A good method should first be reliable. If the method converges to a point x^* , then x^* should be a local minimum for $S(x)$, i.e., $g(x^*) = 0$. Secondly it is obviously desirable that the method converge rapidly. Also it is important for the method to be robust. The robustness of a method is its ability to find solutions, as defined in (Hiebert, 1981).

This thesis develops a hybrid method for nonlinear least squares problems by combining the better characteristics of several earlier methods, and is organized as follows:

Chapter II will give a brief review of some of the existing methods for nonlinear least squares problems and provide some background information for the hybrid method.

The hybrid method will be described in Chapter III which is divided into several sections and subsections.

Chapter IV will give a description of the selected set of test problems and present

some results of numerical experiments, followed by a discussion of these results. The computational performance of the algorithm will be compared with that of some other nonlinear least squares codes such as LMDER and MARQ, etc.

In Chapter V, we will state some conclusions on the new method and make suggestions for further research.

Finally, a relatively large set of test problems and a program listing which implements the new method will be collected in Appendices.

CHAPTER II

LITERATURE REVIEW

When first derivatives are available, most nonlinear least squares methods are based on the use of the first order Taylor approximation of $r(x)$. The most well-known specialized method for nonlinear least squares problems is the Gauss-Newton(GN) method:

$$x^{(k+1)} = x^{(k)} - (J^{(k)t} J^{(k)})^{-1} g^{(k)},$$

where $J^{(k)}$ refers to $J(x^{(k)})$ (likewise $g^{(k)}$ refers to $g(x^{(k)})$, $S^{(k)}$ refers to $S(x^{(k)})$, and $r^{(k)}$ refers to $r(x^{(k)})$, etc.). The GN method can be regarded either as Newton's method simplified by ignoring the second term of the Hessian $H(x)$ in (2), or as a linearization of $r(x)$ in (1) in each step. This linear approximation is good enough when the residual is small or when the function is "nearly linear". If the GN method converges, the rate of convergence depends on the size of residuals and usually is linear for a nonzero residual problem and is superlinear, even quadratic, for a zero residual problem (Fletcher and Xu, 1987). Unlike steepest descent, the GN method is invariant under linear transformations. Besides, the GN method can be implemented so that it has the desirable numerical properties associated with a QR factorization of $J(x)$. Thus, the GN method should always be tried first when many problems of similar type (e.g. fitting exponentials) are going to be solved (Chandler, 1992).

However, the GN method might be ineffective because the linear approximation is poor when the second part of the Hessian $H(x)$ in (2) is not small relative to the first part. The GN method can jam up at a point where the Jacobian $J(x)$ is rank deficient (Fletcher, 1980). Consequently, the modified GN method has been

developed by adding a damping parameter to make it a descent method and to assure convergence. The k -th iteration of the modified GN method looks like

$$x^{(k+1)} = x^{(k)} - \alpha^{(k)}(J^{(k)t}J^{(k)})^{-1}g^{(k)}.$$

The modified GN method has a neighborhood of convergence under mild conditions, but it may still converge too slowly on large residual problems (Ruhe, 1979). A modified GN method for nonlinear regression problems was given by Hartley (1961).

To overcome the difficulties of GN-like methods, the $J(x)^tJ(x)$ matrix can be augmented at crucial steps by a positive definite matrix (usually a diagonal matrix) $\lambda^{(k)}D^{(k)}$ which effectively biases the search direction towards that of scaled steepest descent. In this way, we get the Levenberg-Marquardt method (Levenberg, 1944; Marquardt, 1963):

$$x^{(k+1)} = x^{(k)} - (J^{(k)t}J^{(k)} + \lambda^{(k)}D^{(k)})^{-1}g^{(k)},$$

where the scalar $\lambda^{(k)}$ is called the Levenberg-Marquardt parameter. The definitions of $\lambda^{(k)}$ and $D^{(k)}$ depends on implementations. Among various implementations of the Levenberg-Marquardt method, the one discussed in (Moré, 1978) is efficient and robust, and has strong convergence properties. The major features of this implementation are the proper use of implicitly scaled variables, and the determination of the Levenberg-Marquardt parameter via an efficient scheme due to Hebden (1973). The Levenberg-Marquardt method can be viewed as a model trust region method (Dennis and Schnabel, 1983), which is also called a restricted step method (Fletcher, 1980). Like the GN method, the Levenberg-Marquardt method converges fast in the low-residual cases, and furthermore, it is more reliable and robust than the GN method. Therefore, Levenberg-Marquardt type methods have become the most popular methods used by applied researchers (Martinez, 1987). For example, Meiron (1965) applied a Levenberg-Marquardt type method to automatic lens design, and he considered the relative sensitivities of the variables by assigning a damping factor to each parameter.

The quasi-Newton methods, which approximate the Hessian $H(x)$ with only first-order information, are currently regarded as the best approach for general unconstrained optimization (Xu, 1990). Most popular is the BFGS method with inexact line search, which is superlinearly convergent when the Hessian matrix is positive definite at the solution point, and which generally performs well in practice (Powell, 1976). However, it does not exploit the special structure of the objective function when applied to the nonlinear least squares problem, and it may take many iterations to build up a good estimate of the Hessian matrix (Fletcher and Xu, 1987). Thus, a quasi-Newton method alone is unsatisfactory for nonlinear least squares problems.

An alternative approach is to use quasi-Newton method to approximate the second term in (2) with only first-order information (Nazareth, 1980). In an attempt to combine the better features of the quasi-Newton method and the GN method, Dennis *et al.* (1981) introduced a structured DFP secant method in which local q -superlinear convergence has been proved. Following their idea, Al-Baali and Fletcher (1985) proposed a simple hybrid GN-BFGS method which is more efficient and robust than the structured DFP method. However, Fletcher and Xu (1987) showed by a counterexample that this method does not retain the superlinear convergence property of the BFGS method; they gave a modified hybrid GN-BFGS method which is as efficient and robust as the Al-Baali and Fletcher hybrid method, and is superlinearly convergent if $H(x^*)$ is positive definite. A non-derivative version of the Fletcher and Xu method was presented by Xu (1990). Recently, Yabe and Takahashi (1991) suggested BFGS-like and DFP-like updates in factorized form.

For large-residual problems, or problems in which a good initial starting guess is not available, many iterative methods that attempt to approximate the objective function by using only first-order information tend to suffer from slow convergence or unreliability, while a class of algorithms based on the continuation method is very promising (Salane, 1987; Villiers and Glasser, 1981). The algorithms are based on

solving approximately a sequence of related subproblems with small residuals and a good initial estimate of the solution for each subproblem.

For problems with a large and sparse Jacobian matrix, Martinez (1987) stated that it is difficult to incorporate Hessian approximations in an approach which keeps sparsity, and he developed an “inexact Gauss-Newton” strategy in which the Gauss-Newton equation is partially solved at each iteration using a preconditioned conjugate gradient algorithm, and a two-dimensional trust-region scheme is used. Martinez and Santos (1990) modified the Martinez method by using a simpler curvilinear search instead of the two-dimensional trust-region scheme. Both methods have global convergence properties. Before Martinez and Santos, Witte and Holst (1964) had published a circular arcs method that uses suitable curves to approximate the valleys of the objective function.

Al-Baali and Fletcher (1986) considered sectioning schemes for solving the line search subproblem in general unconstrained optimization by finding an acceptable steplength. To take advantage of the special structure of the nonlinear least squares problem, they also apply polynomial interpolations to the individual residual functions rather than to the overall objective function. This line search scheme was tested with the hybrid GN-BFGS method of Al-Baali and Fletcher (1985) on a set of 55 test problems, and the results showed that substantial gains in efficiency were obtained.

Meyer (1970) gave theorems on the factors which affect the convergence speed of GN-like methods. To compare GN-like methods with general purpose methods, McKeown (1975) constructed a class of test problems for which those factors were known and could be easily controlled. Similar work can also be found in (Ramsin and Wedin, 1977), where about 1000 test problems were generated for testing the GN method, the Levenberg-Marquardt method and a quasi-Newton method. Moré *et al.* (1981) pointed out that many algorithms appeared in the optimization literature were not well tested, because it was often the case that only small number of test

problems were used, and that the starting points were close to the solution. Hence they collected a relatively large but easy-to-use set of test problems and designed guidelines for testing the reliability and robustness of unconstrained optimization software. Hiebert (1981) did a comparison study on mathematical software that solves nonlinear least squares problems. Theoretical and software aspects of the codes, as well as their performance on a set of test problems, were evaluated in her paper.

Ruhe (1979) showed that the GN method with line search behaves asymptotically like steepest descent for a special choice of parameterization. Based on this result, he applied a conjugate gradient (*c-g*) acceleration to the GN method leading to an algorithm which is n -step quadratically convergent, i.e., there exists a constant C such that

$$\|x^{(n)} - x^*\| \leq C\|x^{(0)} - x^*\|^2,$$

provided that $x^{(0)}$ is close enough to the solution x^* . The *c-g* acceleration is easy to implement and only amounts to a negligible portion of extra work. On the test problems given in his paper, his method did converge faster than the GN method. Unfortunately, because of using the GN direction, the Ruhe method inherits the nonrobust property of the GN method.

By using the Levenberg-Marquardt direction, a more robust search direction, and adding Ruhe's conjugate gradient acceleration, plus combining Al-Baali and Fletcher's enhanced line search scheme, it is possible to construct an algorithm which is fast, reliable, and robust. The rest of this thesis will focus on exploring such a possibility. To evaluate the efficiency, reliability and robustness of the resulting algorithm, we will follow Moré *et al.*'s idea and use the set of test problems produced by them.

CHAPTER III

METHODOLOGY

In this chapter, we describe a hybrid method that is based on the Levenberg-Marquardt method and Ruhe's c - g acceleration. The Al-Baali and Fletcher line search scheme is combined into the hybrid method with some modifications.

Given a starting point $x^{(0)}$, the basic structure of the k th iteration of the hybrid method is as follows:

Step 1. Compute a descent search direction $s^{(k)}$ such that

$$s^{(k)t}g^{(k)} < 0; \quad (3)$$

Step 2. Perform a line search along $s^{(k)}$ to find steplength $\alpha^{(k)}$, which is obtained ideally by solving the line search subproblem

$$\alpha^{(k)} = \arg \min_{\alpha > 0} S(x^{(k)} + \alpha s^{(k)}); \quad (4)$$

Step 3. Set $x^{(k+1)} = x^{(k)} + \alpha^{(k)}s^{(k)}$.

The above process can be repeatedly implemented until one of the following conditions is satisfied:

(i) The function reduction is small enough:

$$S^{(k)} - S^{(k+1)} \leq \varepsilon_S \max(1, S^{(k)}), \quad (5)$$

where ε_S is a user-defined tolerance on S . This criterion was used by Al-Baali and Fletcher (1985) and is not scale-invariant, unfortunately.

(ii) The size of the gradient is small enough:

$$\|g^{(k)}\| \leq \varepsilon_g, \quad (6)$$

where ε_g is a user-supplied tolerance on g . This criterion is simple but not scale-invariant. The scale-invariant alternatives can be found in (Marquardt, 1959; Martinez and Santos, 1990).

(iii) Too much work has been done: for example, the number of function evaluations is over $100(n+1)$ (Moré *et al.*, 1980).

For convenience, we denote $S(x^{(k)} + \alpha s^{(k)})$ as $S(\alpha)$, and then

$$S'(\alpha) = s^{(k)t} g(x^{(k)} + \alpha s^{(k)}).$$

Thus (3) is equivalent to $S'(0) < 0$, showing that S is decreasing at $x^{(k)}$ along the direction $s^{(k)}$.

Search Direction

To find a search direction $s^{(k)}$ at the k th iteration, we use the Levenberg-Marquardt correction

$$s^{(k)} = -(J^{(k)t} J^{(k)} + \lambda^{(k)} D^{(k)t} D^{(k)})^{-1} g^{(k)} \quad (7)$$

which is obtained by minimizing the quadratic function

$$q(s) = \|r^{(k)} + J^{(k)} s\|^2 \quad (8)$$

subject to the ellipsoidal constraint

$$\|D^{(k)} s\| \leq \Delta^{(k)}, \quad (9)$$

where $D^{(k)}$ is a diagonal matrix which is used to scale the correction, $\Delta^{(k)}$ is called the size of the trust region, and $\lambda^{(k)}$ is the Levenberg-Marquardt parameter which depends on $\Delta^{(k)}$.

As in (Moré, 1978), the diagonal elements of $D^{(k)}$ are defined by

$$D_{ii}^{(0)} = \begin{cases} \|J_i^{(0)}\|, & \|J_i^{(0)}\| > 0, \\ 1, & \text{otherwise} \end{cases}$$

$$D_{ii}^{(k)} = \max(D_{ii}^{(k-1)}, \|J_i^{(k)}\|), \quad k > 0,$$

where $J_i^{(k)}$ denotes the i -th column of $J^{(k)}$.

Householder transformations (Businger and Golub, 1965) are used to construct a QR factorization of the Jacobian $J(x)$ in order to compute the Levenberg-Marquardt correction in a numerically stable manner. The reader is referred to (Moré, 1978) for more details.

Size of the Trust Region

The size of the trust region $\Delta^{(k)}$ needs to be adjusted on each iteration according to some measure. Moré (1978) suggests the measure

$$\tau = \frac{S^{(k)} - S(x^{(k)} + s^{(k)})}{S^{(k)} - \frac{1}{2}\|r^{(k)} + J^{(k)}s^{(k)}\|^2}. \quad (10)$$

If τ is close to unity or larger, $\Delta^{(k+1)}$ will be given a larger value. If τ is small, a smaller value of $\Delta^{(k+1)}$ will be used. If τ is unacceptably small, $\Delta^{(k)}$ is reduced and $s^{(k)}$ is recomputed.

Since a line search is incorporated in our method, (10) is no longer suitable. Instead, a simpler measure, the relative function reduction, is used. Our new scheme for adjusting $\Delta^{(k)}$ does not require a recomputation of the search direction $s^{(k)}$ on one iteration and is described as follows:

- (i) If the convergence rate is not satisfactory: $\frac{S^{(k-1)} - S^{(k)}}{S^{(k-1)}} < \xi_1$, then set

$$\Delta^{(k+1)} = \Delta^{(k)} \max \left(\xi_2, \min \left(\alpha^{(k)}, \xi_3 \frac{s^{(k)t} g^{(k)}}{\|s^{(k)}\| \|g^{(k)}\|} \right) \right) \quad (11)$$

(ii) else if $s^{(k)}$ is the GN search direction or the convergence rate is good enough:

$$\frac{g^{(k-1)} - g^{(k)}}{g^{(k-1)}} > \xi_4, \text{ then set}$$

$$\Delta^{(k+1)} = \min(\xi_5, \xi_6 \alpha^{(k)}) \|D^{(k)} s^{(k)}\|, \quad (12)$$

where we have set the parameters $\xi_1 = 0.05$, $\xi_2 = 0.2$, $\xi_3 = 1000$, $\xi_4 = 0.1$, $\xi_5 = 5$, and $\xi_6 = 1000$.

In (11), ξ_2 is used to avoid too small size of the trust region. If $\alpha^{(k)}$, the steplength of the last iteration, is small, then $\Delta^{(k+1)}$ will be small. If $s^{(k)}$, the search direction of the last iteration, is nearly orthogonal to $-g^{(k)}$, then $\Delta^{(k+1)}$ will also be small. The steplength of the last iteration is taken into account in (12) where the size of the trust region is incremented if $\alpha^{(k)}$ is not too small.

The Levenberg-Marquardt Parameter

Once $\Delta^{(k)}$ is chosen, the Levenberg-Marquardt parameter $\lambda^{(k)}$ needs to be determined. There are two possible solutions to (8) and (9): either $\lambda^{(k)} = 0$ and $s^{(k)}$ becomes the GN direction, or $\lambda^{(k)} > 0$ and

$$\|D^{(k)} s^{(k)}\| = \Delta^{(k)}. \quad (13)$$

In the remainder of this subsection we assume that $\lambda^{(k)} > 0$.

In practice, it would be expensive and unnecessary to require (13) to hold strictly. As a matter of fact, $\lambda^{(k)}$ is accepted as the Levenberg-Marquardt parameter if it satisfies

$$|\phi(\lambda^{(k)})| \leq \varepsilon_\Delta \quad (14)$$

where

$$\phi(\lambda) = \|D^{(k)}(J^{(k)t} J^{(k)} + \lambda D^{(k)t} D^{(k)})^{-1} g^{(k)}\| - \Delta^{(k)}$$

and $\varepsilon_\Delta \in (0, 1)$ is the relative error in $\|D^{(k)}(J^{(k)t} J^{(k)} + \lambda D^{(k)t} D^{(k)})^{-1} g^{(k)}\|$.

Since ϕ is a convex function, the Newton method could be used to find such a $\lambda^{(k)}$. But there exists a much more efficient scheme because of the particular structure of $\phi(\lambda)$. $\phi(\lambda)$ can be approximated by

$$\tilde{\phi}(\lambda) = \frac{a}{b + \lambda} - \Delta^{(k)},$$

that fits the values $\phi(\lambda)$ and $\phi'(\lambda)$. This leads to the following iterative scheme due to Hebden (1973).

Step 1. Given $\lambda_0 > 0$ and set $u_0 = \|D^{(k)-1}g^{(k)}\|/\Delta^{(k)}$;

If $J^{(k)}$ is not rank deficient, set $l_0 = -\phi(0)/\phi'(0)$; otherwise set $l_0 = 0$;

Step 2. If $\lambda_j \notin (l_j, u_j)$, then let $\lambda_j = \max(0.001u_j, \sqrt{l_j u_j})$;

If λ_j satisfies (14), then set $\lambda^{(k)} = \lambda_j$, terminate;

Step 3. Set $l_{j+1} = \max(l_j, \lambda_j - \phi(\lambda_j)/\phi'(\lambda_j))$;

If $\phi'(\lambda_j) < 0$, then let $u_{j+1} = \lambda_j$;

Step 4. Set $\lambda_{j+1} = \lambda_j - (1 + \phi(\lambda_j)/\Delta^{(k)})(\phi(\lambda_j))/\phi'(\lambda_j)$ and $j = j + 1$;

repeat Step 2.

It can be shown that the above scheme is quadratically convergent. According to Moré (1978), less than two iterations on the average are required when $\varepsilon_\Delta = 0.1$.

Using Ruhe's c - g Acceleration

When the Levenberg-Marquardt parameter $\lambda^{(k)}$ is not zero, the Levenberg-Marquardt search direction defined in (7) is always used. Note that the Levenberg-Marquardt correction $s^{(k)}$ becomes the GN direction when $\lambda^{(k)} = 0$. To accelerate convergence, each GN step can be followed by at most $n - 1$ of Ruhe's c - g steps. A Ruhe c - g accelerated step is defined by

$$s^{(k)} = s^{(k)} + \beta^{(k)} s^{(k-1)}, \quad (15)$$

where $s^{(k)}$ on the RHS is a GN direction, $s^{(k-1)}$ is either a GN direction or a Ruhe direction, and

$$\beta^{(k)} = \frac{\|J^{(k)}s^{(k)}\|^2}{\|J^{(k-1)}s^{(k-1)}\|^2}. \quad (16)$$

Since the QR decomposition of $J(x)$ is used, the $Q^t S^{(k)}$ is always available at each iteration. Assuming $s^{(k)}$ is a GN direction, it can be easily seen that $\|s^{(k)}\| = \|b\|$, where b is a vector with the first n components of $Q^t S^{(k)}$. Thus, the extra computational cost of (16) is negligible. It was noticed by Ramsin and Wedin (1977) that the quotient $\beta^{(k)}$ in (16) can be used to estimate the limiting convergence factor of the undamped GN method. The larger $\beta^{(k)}$ is, the slower the GN method would be.

Based on his experiments, Ruhe (1979) reported that the $c-g$ acceleration never was slower. Unfortunately, our numerical tests indicated that Ruhe's $c-g$ acceleration does not always accelerate the convergence and sometimes makes the convergence slower, in particular for the small residual problems. Therefore, the use of Ruhe's acceleration must be carefully controlled.

It is usually redundant to use $c-g$ acceleration when the GN step is good. Thus, we turn off the $c-g$ acceleration switch if sufficient function reduction is achieved in the previous GN step

$$\frac{S^{(k-1)} - S^{(k)}}{S^{(k-1)}} > \eta, \quad (17)$$

where η is a preset positive parameter.

The switch of Ruhe's acceleration is also turned off when the last Ruhe step is worse than the most recent GN step to a certain degree in the sense of relative function reduction.

Another measure to decide if Ruhe's acceleration should be employed is the angle between the current GN direction and the steepest direction. If the angle is small, Ruhe's acceleration should be used; otherwise the acceleration should not be used.

Having the above three conditions, we are now able to switch automatically

between the Levenberg-Marquardt direction (including the GN direction) and the Ruhe direction.

Line Search

In practice, it is not efficient to carry out an exact line search on each iteration. Therefore an approximate solution of (4) which satisfies certain conditions is accepted. Among various possibilities (Fletcher, 1980), we prefer that which requires the acceptable point $\alpha^{(k)}$ to satisfy both

$$S(\alpha) \leq S(0) + \alpha\rho S'(0) \quad (18)$$

and

$$|S'(\alpha)| \leq -\sigma S'(0), \quad (19)$$

where $\rho \in (0, \frac{1}{2})$ and $\sigma \in (\rho, 1)$ are preset parameters. These conditions are intended to ensure that a sufficient decrease in S is gained on each iteration and that the slope $S'(\alpha)$ is not too bad relative to $S'(0)$. Al-Baali and Fletcher (1986) proposed a special-purpose line search algorithm for nonlinear least squares which guarantees to find an acceptable point that satisfies (18) and (19) in a finite number of steps. In the hybrid method, we will employ Al-Baali and Fletcher's line search algorithm with some modifications.

Al-Baali and Fletcher's Line Search Scheme

Al-Baali and Fletcher's algorithm is based on a sectioning scheme that reduces uniformly an interval known to bracket acceptable points. The scheme is used in conjunction with polynomial interpolation.

When interpolating, we define an interval

$$T(a, \alpha) = \begin{cases} [a + \tau_1(\alpha - a), \alpha - \tau_2(\alpha - a)], & \text{if } a < \alpha, \\ [\alpha + \tau_2(a - \alpha), a - \tau_1(a - \alpha)], & \text{if } \alpha < a, \end{cases} \quad (20)$$

where $0 < \tau_1 \leq \tau_2 \leq \frac{1}{2}$ are preset parameters.

When extrapolating, we define an interval

$$E(\alpha, b) = \begin{cases} [\alpha + \tau_5(b - \alpha), b - \tau_6(b - \alpha)], & \text{if } \alpha < b, \\ [b + \tau_6(\alpha - b), \alpha - \tau_5(\alpha - b)], & \text{if } b < \alpha, \end{cases} \quad (21)$$

where $0 < \tau_3 \leq \tau_4 \leq \frac{1}{2}$ are preset parameters. In (21), We have simplified Al-Baali and Fletcher's definition and found that the simplified form works better than theirs in our numerical experiments.

In the sectioning scheme, sequences $\{a_j\}$, $\{b_j\}$, $\{\alpha_j\}$ are generated within the interval $(0, \mu]$, where μ is the size of the search interval. a_j is always the current best point that satisfies (18) but not (19), and α_j is the current trial point, and b_j either fails to satisfy (18), or $S(b_j) \geq S(a_j)$ or both. It has been proved by Al-Baali and Fletcher (1986) that the interval $(\min(a_j, b_j), \max(a_j, b_j))$ always contains an interval of acceptable points. For the sectioning scheme to be more efficient, it is important to incorporate polynomial interpolation. For nonlinear least squares, it is possible to take greater advantage of the special structure of the objective function S by approximating each individual residual $r_k(\alpha)$ by a quadratic polynomial $q_k(\alpha; r_k(a), r'_k(a), r_k(b))$ which interpolates the values $r_k(a)$, $r'_k(a)$, and $r_k(b)$. Hence $S(\alpha)$ can be approximated by the quartic polynomial

$$Q(\alpha; a, b) = \frac{1}{2} \sum_{k=1}^m (q_k(\alpha; r_k(a), r'_k(a), r_k(b)))^2 \quad (22)$$

Given $a_1 = 0$, $b_1 = \mu$ and $\alpha_1 \in (0, \mu]$, Al-Baali and Fletcher's line search scheme can be expressed as below.

Step 1. Evaluate $S(\alpha_j)$;

if $S(\alpha_j) \leq \bar{S}$ then terminate;

if α_j does not satisfies (18) or $S(\alpha_j) \geq S(a_j)$ then

choose α_{j+1} which minimizes $Q(\alpha; a_j, \alpha_j)$ in $T(a_j, \alpha_j)$,

set $b_{j+1} = \alpha_j$ and $j = j + 1$;

```

repeat Step 1;
endif

Step 2. Evaluate  $S'(\alpha_j)$ ;
if  $\alpha_j$  satisfies (19), then terminate;
if  $(b_j - a_j)S'(\alpha_j) < 0$  then
    choose  $\alpha_{j+1}$  which minimizes  $Q(\alpha; \alpha_j, a_j)$  in  $E(\alpha_j, b_j)$ ;
else
    choose  $\alpha_{j+1}$  which minimizes  $Q(\alpha; \alpha_j, a_j)$  in  $T(a_j, \alpha_j)$ ;
    set  $b_{j+1} = a_j$ ;
endif
set  $a_{j+1} = \alpha_j$  and  $j = j + 1$ ;
repeat Step 1.

```

Since (22) is a quartic polynomial, it is not straightforward to find its minimizer. Fortunately, the convergence result allows any point in $T(a_j, \alpha_j)$ or $E(\alpha_j, b_j)$ to be selected (Al-Baali and Fletcher, 1986), so an approximation of the minimizer of (22) is accepted. This can be done by using the Newton method with termination criterion (Al-Baali and Fletcher, 1986)

$$|\Delta\alpha| \leq \sigma|\alpha_j - a_j|.$$

Now let us take the Rosenbrock function (see APPENDIX A for definition) for a numerical example of the line search scheme. Choosing the parameters $\rho = 0.01$ and $\sigma = 0.1$, the base point at the 8-th iteration is $x^{(8)} = (-.28276, -.27225)$, and the search direction is a Ruhe direction $s^{(8)} = (2.56552, -1.1367)$. The line search found an acceptable point in 4 iterations with 4 function calls and 3 Jacobian evaluations. The iterative process is shown in TABLE I and illustrated in Figure 1 where the first trial point is not drawn.

TABLE I
ONE LINE SEARCH FOR ROSENBROCK'S FUNCTION

j	a_j	b_j	α_j	$2S(\alpha_j)$	$S'(\alpha_j)$	(18) hold?	(19) hold?
0			0.00000	14.05	-14.355		
1	0.00000	48.94600	0.87260	2592		no	
2	0.00000	0.87260	0.04363	13.69	6.1305	yes	no
3	0.04363	0.00000	0.02182	13.65	-4.0870	yes	no
4	0.02182	0.04363	0.03272	13.62	0.9943	yes	yes

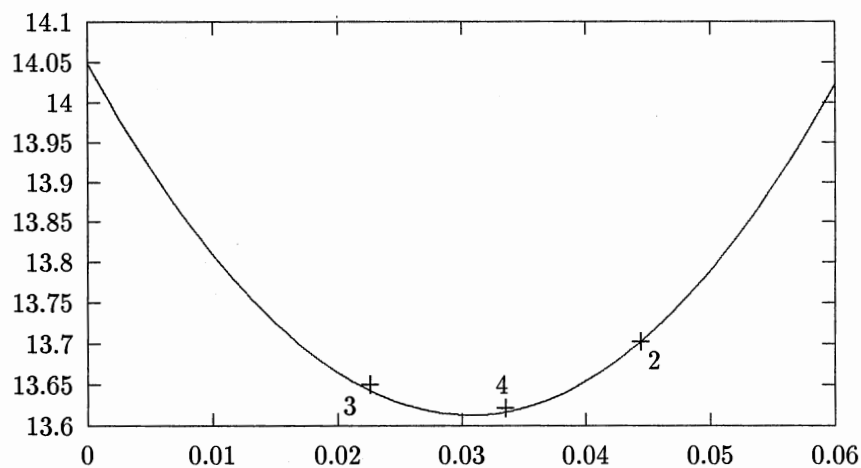


Figure 1. One Line Search for Rosenbrock's Function

Accuracy of the Line Search

Accuracy of the line search can be controlled by the parameters ρ and σ . The larger ρ is and the smaller σ is, the more accurate the line search is. Constant values of $\rho = 0.01$ and $\sigma = 0.1$ are typically used by many researchers (Al-Baali and Fletcher, 1986; Fletcher and Xu, 1987). We found, however, that the parameter choice of $\rho = 0.01$ and $\sigma = 0.1$ is poor in our numerical tests. Instead, we choose the values of ρ and σ dynamically in such a way that ρ is smaller and σ is larger when we believe

the current point is remote from the solution. This is done by defining

$$\rho^{(k)} = \max\left(\rho_1, \frac{\rho_2}{1 + \omega^{(k)}}\right)$$

and

$$\sigma^{(k)} = \min(\sigma_1 + \omega^{(k)}, \sigma_2)$$

where $0 < \rho_1 < \rho_2 < \sigma_1 < \sigma_2 < 1$ are preset parameters, and

$$\omega^{(k)} = \frac{\|g^{(k)}\|}{\max(1, \|S^{(k)}\|)},$$

where $\omega^{(k)}$ is used to estimate how far the current point is from the solution.

Size of the Search Interval

From (18) it follows that

$$\alpha \leq \frac{S(\alpha) - S(0)}{\rho S'(0)}.$$

Let $\bar{S} < S(0)$ be a value of S for which any $\alpha \in \{\alpha | S(\alpha) \leq \bar{S}\}$ is accepted. Then the line search can be restricted within the interval $(0, \mu]$, where the size of the interval

$$\mu = \frac{\bar{S} - S(0)}{\rho S'(0)} \quad (23)$$

In (23), $\bar{S} = 0$ is obviously an acceptable value for a nonlinear least squares problem and (23) becomes

$$\mu = \frac{-S(0)}{\rho S'(0)}. \quad (24)$$

But this often results in an overestimation of the value of μ . We wish to choose \bar{S} such that the value of μ is reasonably small. In fact, we usually cannot expect to gain much decrease in the objective function if the last step is poor. Therefore, we may accept any value of α that satisfies

$$\frac{S(0) - S(\alpha)}{S(0)} \leq \min\left(1, \theta \frac{S^{(k-1)} - S(0)}{S^{(k-1)}}\right),$$

where $\theta > 1$ is a preset parameter. Thus we can choose the value of \bar{S} such that

$$\frac{S(0) - \bar{S}}{S(0)} = \min \left(1, \theta \frac{S^{(k-1)} - S(0)}{S^{(k-1)}} \right).$$

Therefore, (23) can be rewritten as

$$\mu = - \min \left(1, \theta \frac{S^{(k-1)} - S(0)}{S^{(k-1)}} \right) \frac{S(0)}{\rho S'(0)}. \quad (25)$$

This value of μ is smaller than that defined in (24) when the relative reduction in S in the last step is small. (25) works well in practice if the value of θ is properly chosen. The value $\theta=30$ is recommended.

Initial Steplength

For a line search scheme to be efficient, it is important to make a good estimate for the initial steplength. Al-Baali and Fletcher (1986) suggested the choice

$$\alpha_1 = \begin{cases} \min(\alpha_f, \alpha_l, \mu), & \text{if } \max(\alpha_f, \alpha_l) \geq 1, \\ \min(\max(\alpha_f, \alpha_l), \mu), & \text{otherwise,} \end{cases} \quad (26)$$

where

$$\alpha_f = \min \left(1, -2 \max \left(S^{(k-1)} - S^{(k)}, 10\varepsilon_f \max(1, S^{(k)}) \right) / S'(0) \right),$$

and

$$\alpha_l = \frac{-r(0)^t r'(0)}{\|r'(0)\|^2}$$

which is the minimizer of $\|r(0) + \alpha r'(0)\|^2$. Note that $\alpha_l = 1$ when the search direction is the GN direction.

It is observed in numerical tests that if the final steplengths are the same in the two consecutive iterations, the final steplength in the following step tends to be unchanged or only changed a little. Another observation is that when the current step is the GN step, the final steplength is likely to be one or near one. Thus we modify Al-Baali and Fletcher's estimation as follows:

(i) If the current step is GN step, then

$$\text{set } \alpha_1 = 1;$$

(ii) else if $\alpha^{(k-1)} = \alpha^{(k-2)}$, then

$$\text{set } \alpha_1 = \alpha^{(k-1)};$$

(iii) else

use α_1 defined in (26) with modification that

$$\alpha_f = \min \left(1, \max \left(\gamma, \max \left(S^{(k-1)} - S^{(k)}, 30\varepsilon_f \max(1, S^{(k)}) \right) / S'(0) \right) \right),$$

where γ is a preset positive parameter which is used to avoid too small an initial steplength.

Numerical tests show that the new strategy for estimating α_1 is better than that of Al-Baali and Fletcher. Using the new strategy, the cumulative number of function evaluations, gradient evaluations and iterations required to solve a set of test problems are reduced by 24%, 19%, and 13%, respectively, compared with using Al-Baali and Fletcher's estimation.

CHAPTER IV

NUMERICAL EXPERIMENTS

The test code, ALML, examined in this chapter is a Fortran 77 implementation of the hybrid method described in Chapter IV. The performance of ALML will be compared with that of LMDER, a MINPACK code (Moré *et al.*, 1984) which implements the Levenberg-Marquardt method. According to Hiebert (1981), LMDER was the most robust among the codes tested. LMDER is available by ftp or by email at the Internet address *netlib@research.att.com* (Dongarra and Gross, 1987). We will also compare ALML with MARQ (Jackson, 1974) which allows the user to select the GN method, the modified GN method, the Marquardt method or the modified Marquardt method (Chandler, 1992). In addition, the overall performance data of two more nonlinear least squares codes NLSQ1 and NLSQ2 are abstracted from (Moré *et al.*, 1981) for comparison. The development of NLSQ1 and NLSQ2 received considerable attention, and both of them appeared in optimization libraries. However, which methods these two codes implement were not mentioned in (Moré *et al.*, 1981).

All of the numerical results are obtained in double precision on an Everex 386SX/16 computer, using the Microsoft Fortran Optimizing Compiler Version 5.00, and using the analytical Jacobian. All of the numerical tests are carried out on the same set of test problems that are described below.

Test Problems

The evaluation of our code requires a suitable set of test problems. A relatively large collection of 18 test functions used and referenced by (Moré *et al.*, 1981) is used

in our numerical experiments. For easy reference, this set is listed in Appendix A with the following format:

Function number. Name of the function (reference)

- Dimensions (m, n)
- Function definition ($r_k(x)$)
- Standard starting point ($x^{(0)}$)
- Minima (x^* , $\|r(x^*)\|^2$)

Following Moré *et al.*'s suggestion on testing the robustness with respect to the starting point, in addition to the standard starting point $x^{(0)}$, two more extended starting points $10x^{(0)}$ and $100x^{(0)}$ are used, except for functions involving rapidly increasing expressions such as exponentials. In addition, some of the test functions have varied dimensions. These increase the number of test problems to 54.

The test set represents various problems, including linear (function 1, 2 and 3), almost linear (function 16) and rank-deficient linear (function 2 and 3) problems, problems involving exponentials and triangles (function 5, 10, 12, 13, 14, 17 and 18), a singular problem (function 6) and data-fitting problems (function 8, 9, 10, 17, and 18). Several problems (function 1, 2, 3, 10, 13, 14 and 15) have large residuals.

To evaluate the objective function $S(x)$ requires an evaluation of the residual vector $r(x)$, and to evaluate the gradient vector $g(x)$ requires an evaluation of the Jacobian matrix $J(x)$. Note that $r(x)$ and $J(x)$ often contain common subexpressions such as exponential and trigonometric subexpressions. It is efficient to evaluate such subexpressions only once. In our program, any evaluation of $J(x)$ is always preceded by an evaluation of $r(x)$, and the evaluations of $r(x)$ and $J(x)$ of each problem are organized in a single subroutine by using ENTRY statements. Thus the values of the

common subexpressions, calculated during the evaluation of $r(x)$, can be stored by a `SAVE` statement, and can be used in the calculation of $J(x)$ for the same value of x .

To prevent destructive overflows of exponential functions due to too large a steplength, ALML has an option that asks the user to check the exponent in the subprogram of function evaluation before evaluation of the power. If the exponent is greater than a preset bound, then the subprogram return a flag to ALML and ALML will reduce the steplength.

Results and Analysis

For ALML, stopping criteria (5) and (6) were used and the parameter selections

$$\varepsilon_S = \varepsilon_g = 10^{-10}, \quad \eta = 0.2, \quad \gamma = 0.05,$$

$$\rho_1 = 0.0001, \quad \rho_2 = 0.05, \quad \sigma_1 = 0.6, \quad \sigma_2 = 0.8,$$

$$\tau_1 = \tau_3 = 0.05, \quad \text{and} \quad \tau_2 = \tau_4 = 0.5$$

were made, in addition to those given in Chapter IV.

LMDER calculates the actual relative change in the function and the predicted relative change based on a linear model. LMDER uses a test on the ratio of these two changes and a test on the norm of the scaled gradient defined by $\max_{1 \leq i \leq n} (|g_i/D_{ii}|)/\|S\|$. Several other stopping criteria are also used in LMDER. The reader is referred to (Moré *et al.*, 1980) for precise descriptions. As for ALML, the tolerances *ftol* and *gtol* were also set to 10^{-10} for LMDER.

MARQ checks the relative change in x from iterate to iterate. The tolerance *RELMIN* was set to the default value 10^{-6} . A driver routine was written to test MARQ on the problem set. Since MARQ does not provide a correct counter of function evaluations when using the analytic Jacobian, some necessary modifications on MARQ were made.

ALML, LMDER and MARQ were run on the same set of test problems described in the last section and Appendix A, without checking overflows. The results are shown in TABLE II through TABLE VII, where NPROB, NFEV, and NJEV are the problem number, number of function evaluations and gradient evaluations, respectively. In TABLE II and TABLE III, INFO indicates the stopping status. For example, INFO=1 means that the relative error in the function is at most the tolerance; INFO=4 shows that the norm of the (scaled) gradient is small enough; and INFO=fail indicates that the number of function evaluations has reached $100(n + 1)$ before convergence. In TABLE IV through TABLE VII, KFLAG also indicates the stopping status. $KFLAG > 0$ signals a normal exit and $KFLAG < 0$ flags an abnormal exit.

From TABLE II and TABLE III, the following main differences between ALML and LMDER are observed.

- LMDER fails on two problems, while ALML solves all the test problems. So ALML is reliable as well as robust.
- On most easy problems, ALML and LMDER have no significant differences, while on more difficult functions such as function 9, 10 and 14, the performance of ALML is substantially better than that of LMDER.
- ALML and LMDER reach different local solutions on function 10 with starting point $100x^{(0)}$, and function 16 with $m = n = 40$.
- For most problems where ALML and LMDER reach the same local solution, the $\|r^*\|$ obtained by ALML agrees to seven significant digits to that obtained by LMDER, with the exception of some zero-residual problems. LMDER was much more accurate on those problems.

The CPU time required by ALML for one run (54 calls) was about 73 seconds,

while that by LMDER was about 142 seconds. It is clear that ALML outperforms LMDER. As in Ruhe's study, we did not observe a clear-cut occurrence of the n -cyclic quadratic convergence of the Ruhe steps.

Based on TABLE IV through TABLE VII, we would like to mention the following points about the four methods implemented by MARQ:

- The performance of the GN method is satisfactory on most small residual problems, in particular on function 4 and function 7. However, this method suffers from overflows on five problems and did not converge to the solution on quite a few problems. The modified GN method behaves similarly but with fewer overflow cases.
- Both the Marquardt method and the modified Marquardt method do not encounter any overflow, but they fail to find the solution within $100(n+1)$ function evaluations for several test problems including function 4 with the third starting point.
- All of the four methods produce an erroneous solution on one or more problems. All of them have trouble on function 10 and function 14.

From the previous tables, TABLE VIII is obtained, which contains the cumulative number of function evaluations (NFE), gradient evaluations (NJE), and iterations (NIT). TABLE VIII also contains the cumulative number of problems that fail to converge before reaching $100(n+1)$ function evaluations (NFC), problems that converge to erroneous solutions (NWR), and overflowed problems (NOV). The first two types of failures occur with a message and are not nearly as serious as overflows. The overall performance data of NLSQ1 and NLSQ2 are added in TABLE VIII. Although the stopping criteria for NLSQ1 and NLSQ2 are unknown, Moré *et al.* (1981) mentioned that they did not use small tolerances. The test results of NLSQ1 and NLSQ2 were

TABLE II
SUMMARY OF 54 CALLS TO ALML

NPROB	N	M	NFEV	NJEV	NITR	INFO	$\ r^*\ $	$\ g^*\ $
1	5	10	2	2	2	4	.2236068D+01	.2D-14
1	5	50	3	3	3	4	.6708204D+01	.1D-13
2	5	10	3	2	2	1	.1463850D+01	.2D-08
2	5	50	3	2	2	1	.3482630D+01	.2D-06
3	5	10	2	2	2	4	.1909727D+01	.7D-10
3	5	50	3	2	2	1	.3691729D+01	.7D-07
4	2	2	29	16	16	4	.0000000D+00	.0D+00
4	2	2	4	4	4	4	.3557152D-13	.8D-12
4	2	2	5	4	4	4	.2781248D-11	.6D-10
5	3	3	12	10	9	1	.6751559D-13	.1D-11
5	3	3	17	14	11	1	.1114203D-21	.2D-20
5	3	3	27	19	18	1	.1035907D-22	.2D-21
6	4	4	13	12	12	1	.2316163D-05	.2D-07
6	4	4	16	15	15	1	.1181716D-05	.6D-08
6	4	4	20	19	19	1	.1413674D-05	.8D-08
7	2	2	35	21	18	1	.6998875D+01	.7D-03
7	2	2	36	25	21	1	.6998875D+01	.1D-01
7	2	2	37	27	25	1	.6998875D+01	.3D-01
8	3	15	7	6	6	1	.9063596D-01	.1D-08
8	3	15	19	16	12	4	.4174770D+01	.2D-11
8	3	15	11	8	7	4	.4174769D+01	.6D-13
9	4	11	14	12	10	1	.1753584D-01	.4D-06
9	4	11	17	14	12	1	.1753584D-01	.6D-06
9	4	11	94	59	44	1	.3205220D-01	.7D-10
10	3	16	18	12	11	1	.9377945D+01	.5D-03
10	3	16	131	95	84	1	.9377945D+01	.1D-02
10	3	16	3	2	2	4	.6237599D+05	.3D-13
11	6	31	8	7	7	1	.4782959D-01	.1D-05
11	6	31	13	12	12	1	.4782959D-01	.8D-06
11	6	31	17	16	16	1	.4782959D-01	.8D-06
11	9	31	7	6	6	1	.1183115D-02	.3D-11
11	9	31	13	12	12	1	.1183115D-02	.1D-10
11	9	31	18	16	16	1	.1183115D-02	.6D-10
11	12	31	7	6	6	1	.2173104D-04	.1D-13
11	12	31	24	19	18	1	.2173104D-04	.8D-13
11	12	31	21	17	17	1	.2173104D-04	.4D-12
12	3	10	7	6	6	1	.5210805D-19	.4D-19
13	2	10	62	46	43	1	.1115178D+02	.1D+00
14	4	20	58	42	35	1	.2929543D+03	.9D-01
14	4	20	60	43	41	1	.2929543D+03	.1D+00
14	4	20	63	46	41	1	.2929543D+03	.1D+00
15	1	8	1	1	1	4	.1886238D+01	.0D+00
15	1	8	31	30	30	1	.1884248D+01	.3D-03
15	1	8	49	48	48	1	.1884248D+01	.3D-03
15	8	8	59	42	30	1	.5930325D-01	.7D-04
15	9	9	15	10	9	1	.1462322D-09	.4D-09
15	10	10	49	33	23	1	.8064710D-01	.2D-04
16	10	10	22	17	12	1	.7839973D-10	.2D-09
16	10	10	38	32	29	1	.1574083D-11	.5D-11
16	10	10	89	67	50	1	.7352992D-08	.2D-07
16	30	30	25	17	10	1	.3297763D-08	.2D-07
16	40	40	17	11	8	1	.1000000D+01	.4D-11
17	5	33	12	8	8	1	.7392493D-02	.2D-08
18	11	65	18	14	12	1	.2003440D+00	.9D-07

TABLE III
SUMMARY OF 54 CALLS TO LMDER

NPROB	N	M	NFEV	NJEV	INFO	$\ r^*\ $
1	5	10	3	2	2	.2236068D+01
1	5	50	3	2	3	.6708204D+01
2	5	10	3	2	1	.1463850D+01
2	5	50	3	2	1	.3482630D+01
3	5	10	3	2	1	.1909727D+01
3	5	50	3	2	1	.3691729D+01
4	2	2	21	16	4	.0000000D+00
4	2	2	8	5	2	.0000000D+00
4	2	2	6	4	2	.0000000D+00
5	3	3	11	8	2	.9936522D-16
5	3	3	20	15	2	.1044677D-18
5	3	3	19	16	2	.3138778D-28
6	4	4	74	62	2	.1316411D-33
6	4	4	74	65	2	.4024338D-35
6	4	4	91	82	2	.1388366D-41
7	2	2	14	8	1	.6998875D+01
7	2	2	19	12	1	.6998875D+01
7	2	2	24	17	1	.6998875D+01
8	3	15	6	5	1	.9063596D-01
8	3	15	37	36	1	.4174769D+01
8	3	15	14	13	1	.4174769D+01
9	4	11	18	16	1	.1753584D-01
9	4	11	78	70	1	.3205219D-01
9	4	11	500	378	fail	.1753584D-01
10	3	16	126	116	3	.9377945D+01
10	3	16	400	348	fail	.7957519D+03
10	3	16	288	252	3	.9377945D+01
11	6	31	8	7	1	.4782959D-01
11	6	31	14	13	1	.4782959D-01
11	6	31	15	14	1	.4782959D-01
11	9	31	9	8	3	.1183115D-02
11	9	31	20	16	3	.1183115D-02
11	9	31	18	15	1	.1183115D-02
11	12	31	10	9	3	.2173104D-04
11	12	31	13	12	3	.2173104D-04
11	12	31	34	28	3	.2173104D-04
12	3	10	7	6	2	.5210805D-19
13	2	10	21	12	1	.1115178D+02
14	4	20	266	247	1	.2929543D+03
14	4	20	56	44	1	.2929543D+03
14	4	20	253	236	1	.2929543D+03
15	1	8	1	1	4	.1886238D+01
15	1	8	29	28	1	.1884248D+01
15	1	8	47	46	1	.1884248D+01
15	8	8	40	21	1	.5930324D-01
15	9	9	12	9	2	.2538747D-15
15	10	10	25	12	1	.8064710D-01
16	10	10	15	13	2	.1723530D-14
16	10	10	13	8	2	.6363234D-14
16	10	10	22	20	2	.8373908D-14
16	30	30	19	14	2	.1279451D-12
16	40	40	19	14	2	.1387379D-12
17	5	33	18	15	1	.7392493D-02
18	11	65	17	13	1	.2003440D+00

TABLE IV
SUMMARY OF 54 CALLS TO MARQ
(GN)

NPROB	N	M	NFEV	NJEV	KFLAG	$\ r^*\ $
1	5	10	5	2	2	.2236068D+01
1	5	50	4	2	1	.6708204D+01
2	5	10	5	2	2	.1463850D+01
2	5	50	5	2	2	.3482630D+01
3	5	10	4	2	1	.1909727D+01
3	5	50	5	2	2	.3691729D+01
4	2	2	5	3	1	.0000000D+00
4	2	2	5	3	1	.0000000D+00
4	2	2	5	3	1	.0000000D+00
5	3	3	12	10	1	.1325265D-13
5	3	3	11	9	1	.6740326D-13
5	3	3	11	9	1	.8682404D-16
6	4	4	22	20	1	.1154021D-10
6	4	4	24	22	1	.7212621D-10
6	4	4	28	26	1	.2817430D-10
7	2	2	45	43	1	.0000000D+00
7	2	2	21	19	1	.0000000D+00
7	2	2	67	65	1	.0000000D+00
8	3	15	8	6	1	.9063596D-01
8	3	15	9	6	2	.5006651D+01
8	3	15	8	5	2	.5057227D+01
9	4	11	65	63	1	.2058336D-01
9	4	11	98	95	2	.3205219D-01
9	4	11	501	499	-7	.4236203D-01
10	3	16	18	16	1	.9377945D+01
10	3	16	13	11	1	.3765455D+05
10	3	16	8	7	-4	.6237599D+05
11	6	31	11	9	1	.4782959D-01
11	6	31	16	14	1	.4782959D-01
11	6	31	18	16	1	.4782959D-01
11	9	31	7	5	1	.1183115D-02
11	9	31	14	12	1	.1183115D-02
11	9	31	17	15	1	.1183115D-02
11	12	31	8	6	1	.2173104D-04
11	12	31	14	12	1	.2173104D-04
11	12	31	18	16	1	.2173104D-04
12	3	10	8	6	1	.2626850D-16
13	2	10				(overflow)
14	4	20	501	499	-7	.2230001D+05
14	4	20	501	499	-7	.4482580D+03
14	4	20	501	499	-7	.1010313D+04
15	1	8	4	1	2	.1886238D+01
15	1	8	29	27	1	.1884248D+01
15	1	8	46	44	1	.1884248D+01
15	8	8				(overflow)
15	9	9				(overflow)
15	10	10				(overflow)
16	10	10	5	3	1	.1751103D+04
16	10	10	8	5	2	.1000000D+01
16	10	10	4	2	1	.1451099D+04
16	30	30	5	3	1	.1337077D+05
16	40	40	9	6	2	.2249781D+04
17	5	33	9	7	1	.7392493D-02
18	11	65				(overflow)

TABLE V
SUMMARY OF 54 CALLS TO MARQ
(MOD. GN)

NPROB	N	M	NFEV	NJEV	KFLAG	r*
1	5	10	6	2	2	.2236068D+01
1	5	50	12	2	2	.6708204D+01
2	5	10	6	2	2	.1463850D+01
2	5	50	6	2	2	.3482630D+01
3	5	10	8	2	1	.1909727D+01
3	5	50	6	2	2	.3691729D+01
4	2	2	48	17	1	.0000000D+00
4	2	2	17	6	1	.0000000D+00
4	2	2	14	5	1	.0000000D+00
5	3	3	27	9	1	.1047519D-21
5	3	3	42	13	1	.2276381D-19
5	3	3	47	15	1	.1585245D-14
6	4	4	42	20	1	.1154021D-10
6	4	4	46	22	1	.7212621D-10
6	4	4	54	26	1	.2817430D-10
7	2	2	71	8	1	.7614827D+01
7	2	2	79	13	1	.7597293D+01
7	2	2	83	18	1	.7600990D+01
8	3	15	14	6	1	.9063596D-01
8	3	15	87	18	2	.4199227D+01
8	3	15	58	7	2	.4174769D+01
9	4	11	31	11	1	.1753584D-01
9	4	11	501	144	-7	.9052202D-01
9	4	11	501	141	-7	.2974790D+02
10	3	16	21	7	1	.9377945D+01
10	3	16				(overflow)
10	3	16	401	264	-7	.31727910+05
11	6	31	24	8	1	.4782959D-01
11	6	31	32	13	1	.4782959D-01
11	6	31	36	15	1	.4782959D-01
11	9	31	17	6	1	.1183115D-02
11	9	31	28	12	1	.1183115D-02
11	9	31	39	17	1	.1183115D-02
11	12	31	19	6	1	.2173104D-04
11	12	31	43	17	1	.2173104D-04
11	12	31	43	18	1	.2173104D-04
12	3	10	15	6	1	.2626850D-16
13	2	10	63	9	1	.5789415D+02
14	4	20	501	171	-7	.2930032D+03
14	4	20	502	174	-7	.2930275D+03
14	4	20	501	173	-7	.2930851D+03
15	1	8	4	1	2	.1886238D+01
15	1	8	56	27	1	.1884248D+01
15	1	8	90	44	1	.1884248D+01
15	8	8	66	11	1	.1715847D+00
15	9	9	70	14	1	.1501435D+00
15	10	10	61	9	1	.1660925D+00
16	10	10	269	94	1	.2220446D-15
16	10	10	66	22	1	.7771561D-15
16	10	10	6	2	1	.1451099D+04
16	30	30				(overflow)
16	40	40				(overflow)
17	5	33	31	12	1	.7392493D-02
18	11	65	38	15	1	.2003440D+00

TABLE VI
SUMMARY OF 54 CALLS TO MARQ
(MARQUARDT)

NPROB	N	M	NFEV	NJEV	KFLAG	$ r^* $
1	5	10	7	5	1	.2236068D+01
1	5	50	7	5	1	.6708204D+01
2	5	10	7	5	1	.1463850D+01
2	5	50	7	5	1	.3482630D+01
3	5	10	7	5	1	.1909727D+01
3	5	50	7	5	1	.3691729D+01
4	2	2	37	24	1	.2542278D-11
4	2	2	99	66	1	.2542167D-11
4	2	2	301	201	-7	.5140837D+02
5	3	3	13	10	1	.9573746D-14
5	3	3	15	11	1	.1152019D-12
5	3	3	21	14	1	.3154385D-18
6	4	4	24	22	1	.2170043D-11
6	4	4	26	24	1	.9363012D-11
6	4	4	28	26	1	.5592794D-10
7	2	2	31	20	1	.6998875D+01
7	2	2	40	24	1	.6998875D+01
7	2	2	49	27	1	.6998875D+01
8	3	15	10	8	1	.9063596D-01
8	3	15	119	8	-1	.4176093D+01
8	3	15	27	4	2	.4777408D+01
9	4	11	35	32	1	.1753584D-01
9	4	11	87	64	1	.1753584D-01
9	4	11	492	331	1	.1753584D-01
10	3	16	402	268	-7	.7689756D+02
10	3	16	402	268	-7	.3765455D+05
10	3	16	401	264	-7	.3172791D+05
11	6	31	13	11	1	.4782959D-01
11	6	31	16	14	1	.4782959D-01
11	6	31	18	16	1	.4782959D-01
11	9	31	16	14	1	.1183115D-02
11	9	31	22	17	1	.1183115D-02
11	9	31	24	18	1	.1183115D-02
11	12	31	21	18	1	.2173104D-04
11	12	31	23	20	1	.2173104D-04
11	12	31	28	22	1	.2173104D-04
12	3	10	14	11	1	.2749341D+00
13	2	10	73	28	1	.1115178D+02
14	4	20	505	89	-7	.7918841D+03
14	4	20	502	94	-7	.6017687D+03
14	4	20	506	103	-7	.4605683D+03
15	1	8	4	1	2	.1886238D+01
15	1	8	30	28	1	.1884248D+01
15	1	8	46	44	1	.1884248D+01
15	8	8	59	31	1	.5930324D-01
15	9	9	13	10	1	.4925404D-14
15	10	10	29	14	1	.8064710D-01
16	10	10	12	10	1	.1110223D-15
16	10	10	26	18	1	.8042798D-14
16	10	10	41	33	1	.6262679D-10
16	30	30	13	11	1	.4664734D-13
16	40	40	15	12	1	.2719469D-12
17	5	33	52	37	1	.7392493D-02
18	11	65	15	12	1	.2003440D+00

TABLE VII
SUMMARY OF 54 CALLS TO MARQ
(MOD. MARQUARDT)

NPROB	N	M	NFEV	NJEV	KFLAG	r*
1	5	10	12	5	1	.2236068D+01
1	5	50	12	5	1	.6708204D+01
2	5	10	13	5	1	.1463850D+01
2	5	50	12	5	1	.3482630D+01
3	5	10	13	5	1	.1909727D+01
3	5	50	12	5	1	.3691729D+01
4	2	2	65	25	1	.9138813D-10
4	2	2	146	51	1	.2497260D-09
4	2	2	301	102	-7	.1824937D+02
5	3	3	27	10	1	.5028442D-12
5	3	3	42	16	1	.8204625D-12
5	3	3	128	50	1	.1811305D-12
6	4	4	46	22	1	.2170043D-11
6	4	4	50	24	1	.9363012D-11
6	4	4	54	26	1	.5592794D-10
7	2	2	43	14	1	.6998875D+01
7	2	2	53	17	1	.6998875D+01
7	2	2	66	22	1	.6998875D+01
8	3	15	18	8	1	.9063596D-01
8	3	15	37	8	2	.4174770D+01
8	3	15	11	4	2	.4174769D+01
9	4	11	32	12	1	.1753584D-01
9	4	11	93	35	1	.1753584D-01
9	4	11	466	171	1	.1753584D-01
10	3	16	401	161	-7	.4661629D+02
10	3	16	401	159	-7	.8216228D+03
10	3	16	402	137	-7	.3188472D+05
11	6	31	25	10	1	.4782959D-01
11	6	31	32	13	1	.4782959D-01
11	6	31	36	15	1	.4782959D-01
11	9	31	30	14	1	.1183115D-02
11	9	31	41	18	1	.1183115D-02
11	9	31	47	20	1	.1183115D-02
11	12	31	40	18	1	.2173104D-04
11	12	31	79	25	1	.2173104D-04
11	12	31	74	30	1	.2173104D-04
12	3	10	29	11	1	.2749341D+00
13	2	10	35	11	1	.1115178D+02
14	4	20	501	166	-7	.1165153D+04
14	4	20	502	163	-7	.1693845D+04
14	4	20	501	166	-7	.2929694D+03
15	1	8	4	1	2	.1886238D+01
15	1	8	58	28	1	.1884248D+01
15	1	8	90	44	1	.1884248D+01
15	8	8	83	30	1	.5930324D-01
15	9	9	23	10	1	.1067782D-13
15	10	10	49	17	1	.8064710D-01
16	10	10	22	9	1	.3179195D-14
16	10	10	42	16	1	.2799225D-11
16	10	10	72	28	1	.2937650D-12
16	30	30	25	11	1	.3965151D-11
16	40	40	26	11	1	.1109114D-09
17	5	33	81	33	1	.7392493D-02
18	11	65	43	19	1	.2003440D+00

collected by Moré *et al.* (1981) in two tables similar to TABLE II. Because of using different stopping criteria, NLSQ1 and NLSQ2 exceed the limitation on function evaluations ($100(n + 1)$) on a few test problems. To make the results of NLSQ1 and NLSQ2 comparable with the other codes, we have abandoned the portion of the number of the function evaluations that is over $100(n+1)$ and deducted proportionally from the number of gradient evaluations.

TABLE VIII
OVERALL PERFORMANCE DATA

CODE	NFE	NJE	NIT	NFC	NWR	NOV
ALML	1384	1047	919	0	0	0
LMDER	2887	2437	2437	2	0	0
MARQ(GN) [†]	2765	2658	2658	4	6	5
MARQ(Mod. GN) [†]	4447	1414	1414	6	9	3
MARQ(Marquardt)	4837	2482	2482	7	3	0
MARQ(Mod. Marquardt)	5546	2041	2041	7	1	0
NLSQ1	2935	2530	2530	1	0	0
NLSQ2	4366	3480	3480	4	0	0

[†] Performance data of overflowed problems were not counted

Compared with LMDER, the cumulative number of function evaluations, gradient evaluations, and iterations required by ALML are reduced by 52%, 57%, and 63%, respectively. Another interesting fact observed from TABLE VIII is that the average number of function evaluations and gradient evaluations per line search are as low as 1.51 and 1.14, respectively, noting that one line search is carried out for each iteration. On the basis of TABLE VIII, it can now be concluded that ALML is the best among the codes listed here, at least on the set of test problems we used.

To produce more large-residual problems, some data-fitting problems are ex-

tended by adding “noise” to the data points as follows:

$$y_i + (-1)^i \varepsilon_{noise},$$

where ε_{noise} is a constant. ALML were tested on several problems and results are given in TABLE IX.

TABLE IX
SUMMARY OF 11 CALLS TO ALML
($\varepsilon_{noise} = 10$)

NPROB	N	M	NFEV	NJEV	NITR	INFO	$\ r^*\ $	$\ g^*\ $
8	3	15	31	20	7	1	.3680492D+02	.9D-03
8	3	15	27	15	11	1	.3943284D+02	.2D-10
8	3	15	16	6	5	1	.3943284D+02	.6D-12
9	4	11	265	186	113	1	.3062098D+02	.3D-06
9	4	11	500	363	232	fail	.2859849D+02	.2D+00
9	4	11	500	354	205	fail	.2859803D+02	.1D+01
10	3	16	17	11	11	1	.4037313D+02	.3D-01
10	3	16	109	69	36	1	.3771211D+05	.3D+05
10	3	16	2	2	2	4	.6237885D+05	.0D+00
17	5	33	600	481	353	fail	.5566754D+02	.1D+01
18	11	65	1200	909	610	fail	.7934415D+02	.4D-01

CHAPTER V

CONCLUSIONS

This study provides an alternative method for nonlinear least squares problems. The method is a combination of the Levenberg-Marquardt method, Ruhe's $c-g$ acceleration and a modified version of Al-Baali and Fletcher's line search scheme. By combining the better features of the Levenberg-Marquardt method and the Ruhe method, and by adding the power of the enhanced line search scheme which takes advantage of the special structure of the objective function, the hybrid method exhibits efficiency, reliability and robustness. Also, our estimation of the size of the search interval, our strategy for choosing initial steplength, and our scheme for controlling accuracy of the line search contribute to the better performance of the method. The results of numerical tests show that the new method is very competitive with a well-developed implementation of the Levenberg-Marquardt method.

Further research might be done in the following aspects:

- In practice, it is often the case that explicit expressions for the Jacobian $J(x)$ cannot be given. Even if analytical expressions for $J(x)$ are available, it may take the user a lot of time to code them. Thus a non-derivative version of ALML is needed, and some of the ideas of Xu (1990) may be helpful. Hiebert (1981) tested several nonlinear least squares codes including LMDER and she found very little difference between using the analytic Jacobian and an approximate Jacobian. Based on this result, we can expect that ALML using an approximate Jacobian will also be effective.

- It is desirable to test the codes on more large residual problems such as the signomial problems, the exponential problems and the trigonometric problems introduced by Al-Baali and Fletcher (1985).
- Since our scheme for adjusting the size of the trust region is primitive, there is still much room for refinement.
- It might be interesting to combine the curvilinear search scheme (Martinez and Santos, 1990) into the algorithm to take the place of the iterative scheme for finding the Levenberg-Marquardt parameter since it is quite expensive,
- Finally, it is possible to improve the algorithm under the framework provided by the continuation approach (Salane, 1987).

A SELECTED BIBLIOGRAPHY

1. Al-Baali, M. and Fletcher, R., Variational Methods for Non-Linear Least-Squares, *Journal of the Operational Research Society*, Vol. 36, pp. 405-421, 1985.
2. Al-Baali, M. and Fletcher, R., An Efficient Line Search for Nonlinear Least Squares, *Journal of Optimization Theory and Applications*, Vol. 48, pp. 359-377, 1986.
3. Bard, Y., Comparison of Gradient Methods for the Solution of Nonlinear Parameter Estimation Problems, *SIAM Journal of Numerical Analysis*, Vol. 7, pp. 157-186, 1970.
4. Betts, J. T., Solving the Nonlinear Least Square Problem, *Journal of Optimization Theory and Applications*, Vol. 18, pp. 469-483, 1976.
5. Box, M. J., A Comparison of Several Current Optimization Methods, and the Use of Transformations in Constrained Problems, *Computer Journal*, Vol. 9, pp. 66-77, 1966.
6. Brent, R. P., Algorithms for Minimization without Derivatives, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
7. Brown, K. M., A Quadratically Convergent Newton-like Method Based upon Gaussian Elimination, *SIAM Journal of Numerical Analysis*, Vol. 6, pp. 560-569, 1969.
8. Brown, K. M. and Dennis, J. E., New Computational Algorithms for Minimizing a Sum of Squares of Nonlinear Functions, Report No. 71-76, Yale University, Department of Computer Science, New Haven, Connecticut, 1971.
9. Businger, P. and Golub, G. H., Linear Least-squares Solution by Householder Transformations, *Numerische Mathematik*, Vol. 7, pp. 269-276, 1965.
10. Chandler, J. P., Private Communication, Oklahoma State University, 1992.
11. Dennis, J. E., Jr., Gay, D. M. and Welsch, R. E., An Adaptive Nonlinear Least Squares Algorithm, *ACM Transactions on Mathematical Software*, Vol. 7, pp. 348-369, 1981.

12. Dennis, J. E., Jr., Martinez, H. J., and Tapia, R. A., A Convergence Theory for the Structured BFGS Secant Method with an Application to Nonlinear Least Squares, Rice University, Mathematical Sciences Department, Technical Report No. 87-15, 1987.
13. Dennis, J. E. and Schnabel R. B., Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, NJ, 1983.
14. Dongarra, J. J. and Gross E., Distribution of Mathematical Software Via Electronic Mail, Comm. ACM, Vol. 30, pp. 403-407, 1987.
15. Fletcher, R., Function Minimization without Evaluating Derivatives - A Review, Computer Journal, Vol. 8, pp. 33-41, 1965.
16. Fletcher, R., Practical Methods of Optimization, Vol. 1: Unconstrained Optimization, Wiley, Chichester, England, 1980.
17. Fletcher, R. and Powell, M. J. D., A Rapidly Convergent Descent Method for Minimization, Computer Journal, Vol. 6, pp. 163-168, 1963.
18. Fletcher, R. and Xu, C., Hybrid Methods for Nonlinear Least Squares, IMA Journal of Numerical Analysis, Vol. 7, pp. 371-389, 1987.
19. Freudenstein, F. and Roth, B., Numerical Solutions of Systems of Nonlinear Equations, Journal of ACM, Vol. 10, pp. 550-556, 1963.
20. Gauss, C. F., Theoria Combinationum Observationum Erroribus Minimis Obnoxiae, Werke, Vol. 4, pp. 3-93, 1873.
21. Hartley, H. O., The Modified Gauss-Newton Method for the Fitting of Non-Linear Regression Functions by Least Squares, Technometrics, Vol. 3, pp. 269-280, 1961.
22. Hebden, M. D., An algorithm for Minimization Using Exact Second Derivatives, Atomic Energy Research Establishment Report TP515, Harwell, England, 1973.
23. Hiebert, K. L., An Evaluation of Mathematical Software That Solves Nonlinear Least Squares Problems, ACM Transactions on Mathematical Software, Vol. 7, pp. 1-16, 1981.
24. Jackson, L. W., A Comparison of Selected Gradient Methods for Solving the Nonlinear Least Squares Problem, Master's Report, Computing and Information Sciences Department, Oklahoma State University, 1974.
25. Jennrich, R. I. and Sampson, P. F., Application of Stepwise Regression to Non-linear Estimation, Technometrics, Vol. 10, pp. 63-72, 1968.
26. Kowalik, J. S. and Osborne, M. R., Methods for Unconstrained Optimization Problems, Elsevier North-Holland, New York, 1968.

27. Levenberg, K., A Method for the Solution of Certain Nonlinear Problems in Least Squares, *Quarterly of Applied Mathematics*, Vol. 2, pp. 164-168, 1944.
28. Luenberger, D. G., Introduction to linear and nonlinear programming, Addison Wesley, Reading Mass., 1973.
29. Marquardt, D. W., Solution of Nonlinear Chemical Engineering Models, *Chemical Engineering Progress*, Vol. 55, pp. 65-70, 1959.
30. Marquardt, D. W., An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM Journal of Applied Mathematics*, Vol. 11, pp. 431-441, 1963.
31. Marquardt, D. W., Bennett, R. G. and Burrell, E. J., Least Squares Analysis of Electron Paramagnetic Resonance Spectra, *Journal of Molecular Spectroscopy*, Vol. 7, pp. 269-279, 1961.
32. Martinez, J. M. An Algorithm for Solving Sparse Nonlinear Least Squares Problems, *Computing*, Vol. 39, pp. 307-325, 1987.
33. Martinez, J. M. and Santos R. F., An Algorithm for Solving Nonlinear Least-Squares Problems with a New Curvilinear Search, *Computing*, Vol. 44, pp. 83-90, 1990.
34. McKeown, J. J., Specialized Versus General-Purpose Algorithms for Minimizing Functions that are Sums of Squared Terms, *Mathematical Programming*, Vol. 9, pp. 57-68, 1975.
35. Meiron, J., Damped Least-Squares Method for Automatic Lens Design, *Journal of the Optical Society of America*, Vol. 55, pp. 1105-1109, 1965.
36. Meyer, R. R., Theoretical and Computational Aspects of Nonlinear Regression, in Nonlinear Programming, edited by J. B. Rosen, O. L. Mangasarian, and K. Ritter, Academic Press, New York, pp. 465-486, 1970.
37. Moré, J. J., The Levenberg-Marquardt algorithm: Implementation and Theory, in Lecture Notes in Mathematics, No. 630-Numerical Analysis, edited by G. Watson, Springer-Verlag, New York, pp. 105-116, 1978.
38. Moré, J. J., Garbow, B. S. and Hillstom, K. E., User guide for MINPACK-1, Argonne National Laboratories, Report ANL-80-74, Argonne, IL60439, 1980.
39. Moré, J. J., Garbow, B. S. and Hillstom, K. E., Testing Unconstrained Optimization Software, *ACM Transactions on Mathematical Software*, Vol. 7, pp. 17-40, 1981.
40. Moré, J. J., Garbow, B. S. and Hillstom, K. E., The MINPACK Project, in Sources and Development of Mathematical Software, edited by W. R. Cowell, Prentice-Hall, Englewood Cliffs, NJ, 1984.

41. Nazareth, L., Some recent approaches to solving large residual nonlinear least squares problems, *SIAM review*, Vol.22, pp. 1-11, 1980.
42. Osborne, M. J., Some Aspects of Nonlinear Least Squares Calculations, in Numerical Methods for Nonlinear Optimization, edited by F. A. Lootsma, Gordon & Breach, New York, pp. 87-114, 1972.
43. Powell, M. J. D., An iterative Method for Finding Stationary Values of a Function of Several Variables, *Computer Journal*, Vol. 5, pp. 147-151, 1962.
44. Powell, M. J. D., Some Global Convergence Properties of a Variable Metric Algorithm for Minimization without Exact Line Searches, in Nonlinear Programming, edited by R. W. Cottle and C. E. Lemke, SIAM-AMS Proceedings, Vol. 9, Philadelphia, Pennsylvania, 1976.
45. Rosenbrock, H. H., An Automatic Method for Finding the Greatest or Least Values of a Function , *Computer Journal*, Vol. 3, pp. 175-184, 1960.
46. Ramsin, H. and Wedin, P. A., A Comparison of Some Algorithms for the Non-Linear Least Squares Problem, *BIT*, Vol. 17, pp. 72-90, 1977.
47. Ruhe A., Accelerated Gauss-Newton Algorithms for Nonlinear Least Squares, *BIT*, Vol. 19, pp. 356-367, 1979.
48. Salane E. D., A Continuation Approach for Solving Large-Residual Nonlinear Least Squares Problems, Vol. 8, pp. 655-671, 1987.
49. Villiers, N. D. and Glasser, D., A Continuation Method for Nonlinear Regression, *SIAM Journal of Numerical Analysis*, Vol. 18, pp. 1139-1154, 1981.
50. Witte, B. F. W. and Holst, W. R., Two New Direct Minimum Search Procedures for Functions of Several Variables, Proceedings-Spring Joint Computer Conference, pp. 195-209, 1964.
51. Xu, C., Hybrid Method for Nonlinear Least-Square Problems without Calculating Derivatives, *Journal of Optimization Theory and Applications*, Vol. 65, pp. 555-574, 1990.
52. Yabe, H. and Takahashi, T., Factorized Quasi-Newton Methods for nonlinear least squares problems, *Mathematical Programming*, Vol. 51, pp. 75-100, 1991.

APPENDIX A
TEST FUNCTIONS

1. Linear function – full rank (Moré *et al.*, 1981)

- $m \geq n$
- $r_i(x) = \begin{cases} x_i - \frac{2}{m} \sum_{j=1}^n x_j - 1, & i \leq n \\ -\frac{2}{m} \sum_{j=1}^n x_j - 1, & i > n \end{cases}$
- $x^{(0)} = (1, \dots, 1)$
- $\|r^*\|^2 = m - n$ at $(-1, \dots, -1)$

2. Linear function – rank 1 (Moré *et al.*, 1981)

- $m \geq n$
- $r_i(x) = i \sum_{j=1}^n jx_j - 1$
- $x^{(0)} = (1, \dots, 1)$
- $\|r^*\|^2 = \frac{m(m-1)}{2(2m-1)}$ at any point where $\sum_{j=1}^n jx_j = \frac{3}{2m+1}$

3. Linear function – rank 1 with zero columns and rows (Moré *et al.*, 1981)

- $m \geq n$
- $r_i(x) = \begin{cases} (i-1) \sum_{j=2}^{n-1} jx_j - 1, & 1 < i < m \\ -1, & i = 1, m \end{cases}$
- $x^{(0)} = (1, \dots, 1)$

- $\|r^*\|^2 = \frac{m^2 + 3m - 6}{2(2m - 3)}$ at any point where $\sum_{j=2}^{m-1} jx_j = \frac{3}{2m - 3}$

4. Rosenbrock function (Rosenbrock, 1960)

- $m = n = 2$
- $r_1(x) = 10(x_2 - x_1^2)$
 $r_2(x) = 1 - x_1$
- $x^{(0)} = (-1.2, 1)$
- $\|r^*\|^2 = 0$ at $(1, 1)$

5. Helical valley function (Fletcher and Powell, 1963)

- $m = n = 3$
- $r_1(x) = 10(x_3 - 10t)$
 $r_2(x) = 10(\sqrt{x_1^2 + x_2^2} - 1)$
 $r_3(x) = x_3$

where

$$t = \begin{cases} \frac{1}{2\pi} \arctan \frac{x_2}{x_1}, & x_1 > 0 \\ \text{sign}(0.25, x_2), & x_1 = 0 \\ \frac{1}{2\pi} \arctan \frac{x_2}{x_1} + 0.5, & x_1 < 0 \end{cases}$$

- $x^{(0)} = (-1, 0, 0)$
- $\|r^*\|^2 = 0$ at $(1, 0, 0)$

6. Powell singular function (Powell, 1962)

- $m = n = 4$
- $r_1(x) = x_1 + 10x_2$
 $r_2(x) = \sqrt{5}(x_3 - x_4)$
 $r_3(x) = (x_2 - 2x_3)^2$
 $r_4(x) = \sqrt{10}(x_1 - x_4)^2$

- $x^{(0)} = (3, -1, 0, 1)$
- $\|r^*\|^2 = 0$ at $(0, 0, 0, 0)$

7. Freudenstein and Roth function (Freudenstein and Roth, 1963)

- $m = n = 2$
- $r_1(x) = -13 + x_1 + ((5 - x_2)x_2 - 2)x_2$
 $r_2(x) = -29 + x_1 + ((1 + x_2)x_2 - 14)x_2$
- $x^{(0)} = (0.5, -2)$
- $\|r^*\|^2 = 0$ at $(5, 4)$
 $\|r^*\|^2 = 48.9842\dots$ at $(11.41\dots, -0.8968\dots)$

8. Bard function (Bard, 1970)

- $n = 3, m = 15$
- $r_i(x) = y_i - \left(x_1 + \frac{i}{(16 - i)x_2 + \min(i, 16 - i)x_3} \right)$,
 where y_i 's are as in TABLE X.
- $x^{(0)} = (1, 1, 1)$
- $\|r^*\|^2 = 8.21487\dots 10^{-3}$
 $\|r^*\|^2 = 17.4286\dots$ at $(0.8406\dots, -\infty, -\infty)$

TABLE X

DATA FOR FUNCTION 8

i	y_i	i	y_i	i	y_i	i	y_i	i	y_i
1	0.14	4	0.25	7	0.35	10	0.58	13	1.34
2	0.18	5	0.29	8	0.39	11	0.73	14	2.10
3	0.22	6	0.32	9	0.37	12	0.96	15	4.39

9. Kowalik and Osborne function (Kowalik and Osborne, 1968)

- $m = 4, n = 11$
- $r_i(x) = y_i - \frac{x_1 u_i (u_i + x_2)}{u_i (u_i + x_3) + x_4}$,
where y_i 's and u_i 's are as in TABLE XI.
- $x^{(0)} = (0.25, 0.39, 0.415, 0.39)$
- $\|r^*\|^2 = 3.07505 \dots 10^{-4}$
 $\|r^*\|^2 = 1.02734 \dots 10^{-3}$ at $(+\infty, -14.07 \dots, -\infty, -\infty)$

TABLE XI
DATA FOR FUNCTION 9

i	y_i	u_i	i	y_i	u_i	i	y_i	u_i
1	0.1957	4.0000	5	0.0844	0.2500	9	0.0323	0.0833
2	0.1947	2.0000	6	0.0627	0.1670	10	0.0235	0.0714
3	0.1735	1.0000	7	0.0456	0.1250	11	0.0246	0.0625
4	0.1600	0.5000	8	0.0342	0.1000			

10. Meyer function (Meyer, 1970)

- $n = 3, m = 16$
- $r_i(x) = x_1 e^{\frac{x_2}{5i+45+x_3}} - y_i$,
where y_i 's are as in TABLE XII.
- $x^{(0)} = (0.02, 4000, 250)$
- $\|r^*\|^2 = 87.9458 \dots$

TABLE XII
DATA FOR FUNCTION 10

i	y_i	i	y_i	i	y_i	i	y_i	i	y_i
1	34780	5	16370	8	9744	11	6005	14	3820
2	28610	6	13720	9	8261	12	5147	15	3307
3	23650	7	11540	10	7030	13	4427	16	2872
4	19630								

11. Watson function (Kowalik and Osborne, 1968)

- $2 \leq n \leq 31, m = 31$

$$\bullet r_i(x) = \begin{cases} \sum_{j=2}^n (j-1) \left(\frac{i}{29}\right)^{j-2} x_j - \left(\sum_{j=1}^n \left(\frac{i}{29}\right)^{j-1} x_j\right)^2 - 1, & i \leq 29 \\ x_1, & i = 30 \\ x_2 - x_1^2 - 1, & i = 31 \end{cases}$$

- $x^{(0)} = (0, \dots, 0)$

- $\|r^*\|^2 = 2.28767 \dots 10^{-3}$ if $n = 6$
- $\|r^*\|^2 = 1.39976 \dots 10^{-6}$ if $n = 9$
- $\|r^*\|^2 = 4.72238 \dots 10^{-10}$ if $n = 12$

12. Box three-dimensional function (Box, 1966)

- $n = 3, m \geq n$
- $r_i(x) = e^{-0.1ix_1} - e^{-0.1ix_2} - x_3(e^{-0.1i} - e^{-i})$
- $x^{(0)} = (0, 10, 20)$
- $\|r^*\|^2 = 0$ at $(1, 10, 1), (10, 1, -1)$ and wherever $x_1 = x_2$ and $x_3 = 0$

13. Jennrich and Sampson function (Jennrich and Sampson, 1968)

- $n = 2, m \geq n$

- $r_i(x) = 2 + 2i - (e^{ix_1} + e^{ix_2})$
- $x^{(0)} = (0.3, 0.4)$
- $\|r^*\|^2 = 124.362 \dots$ at $x_1 = x_2 = 0.2578 \dots$ for $m = 10$

14. Brown and Dennis function (Brown and Dennis, 1971)

- $n = 4, m \geq n$
- $r_i(x) = (x_1 + \frac{i}{5}x_2 - e^{\frac{i}{5}})^2 + (x_3 + x_4 \sin \frac{i}{5} - \cos \frac{i}{5})^2$
- $x^{(0)} = (25, 5, -5, -1)$
- $\|r^*\|^2 = 85822.2 \dots$ if $m = 20$

15. Chebyquad function (Fletcher, 1965)

- $m \geq n$
- $r_i(x) = \frac{1}{n} \sum_{j=1}^n T_i(x_j) - \int_0^1 T_i(x) dx,$
where T_i is the i th Chebyshev polynomial shifted to the interval $[0,1]$, and hence

$$\int_0^1 T_i(x) dx = \begin{cases} 0, & \text{for } i \text{ odd} \\ \frac{1}{1-i^2}, & \text{for } i \text{ even} \end{cases}$$

- $x^{(0)} = (\frac{1}{n+1}, \dots, \frac{n}{n+1})$
- $\|r^*\|^2 = 0$ for $m = n, 1 \leq n \leq 7$, and $n = 9$
 $\|r^*\|^2 = 3.51687 \dots 10^{-3}$ for $m = n = 8$
 $\|r^*\|^2 = 6.50395 \dots 10^{-3}$ for $m = n = 10$

16. Brown almost-linear function (Brown, 1969)

- $n = m$
- $r_i(x) = \begin{cases} x_i + \sum_{j=1}^n x_j - (n+1), & 1 \leq i < n \\ \prod_{j=1}^n x_j - 1, & i = n \end{cases}$

- $x^{(0)} = (\frac{1}{2}, \dots, \frac{1}{2})$
- $\|r^*\|^2 = 0$ at $(\alpha, \dots, \alpha, \alpha^{1-n})$,
where $\alpha \neq 1$ and α satisfies $n\alpha^n - (n+1)\alpha^{n-1} + 1 = 0$
 $\|r^*\|^2 = 1$ at $(0, \dots, 0, n+1)$

17. Osborne 1 function (Osborne, 1972)

- $n = 5, m = 33$
- $r_i(x) = y_i - (x_1 + x_2 e^{-10(i-1)x_4} + x_3 e^{-10(i-1)x_5})$,
where y_i 's are as in TABLE XIII.
- $x^{(0)} = (0.5, 1.5, -1, 0.01, 0.02)$
- $\|r^*\|^2 = 5.46489 \dots 10^{-5}$

TABLE XIII
DATA FOR FUNCTION 17

i	y_i	i	y_i	i	y_i	i	y_i	i	y_i
1	0.844	8	0.850	15	0.628	22	0.490	29	0.424
2	0.908	9	0.818	16	0.603	23	0.478	30	0.420
3	0.932	10	0.784	17	0.580	24	0.467	31	0.414
4	0.925	11	0.751	18	0.558	25	0.457	32	0.411
5	0.925	12	0.718	19	0.538	26	0.448	33	0.406
6	0.908	13	0.685	20	0.522	27	0.438		
7	0.881	14	0.658	21	0.506	28	0.431		

18. Osborne 2 function (Osborne, 1972)

- $n = 11, m = 65$
- $r_i(x) = y_i - (x_1 e^{-\frac{i-1}{10}x_5} + x_2 e^{-(\frac{i-1}{10}-x_9)^2x_6} + x_3 e^{-(\frac{i-1}{10}-x_{10})^2x_7} + x_4 e^{-(\frac{i-1}{10}-x_{11})^2x_8})$,
where y_i 's are as in TABLE XIV.

TABLE XIV
DATA FOR FUNCTION 18

i	y_i	i	y_i	i	y_i	i	y_i	i	y_i
1	1.366	14	0.665	27	0.612	40	0.429	53	0.597
2	1.191	15	0.616	28	0.558	41	0.523	54	0.625
3	1.112	16	0.606	29	0.533	42	0.562	55	0.739
4	1.013	17	0.602	30	0.495	43	0.607	56	0.710
5	0.991	18	0.626	31	0.500	44	0.653	57	0.729
6	0.885	19	0.651	32	0.432	45	0.672	58	0.720
7	0.831	20	0.724	33	0.395	46	0.708	59	0.636
8	0.847	21	0.649	34	0.375	47	0.633	60	0.581
9	0.786	22	0.649	35	0.372	48	0.668	61	0.428
10	0.725	23	0.694	36	0.391	49	0.645	62	0.292
11	0.746	24	0.644	37	0.396	50	0.632	63	0.162
12	0.679	25	0.624	38	0.405	51	0.591	64	0.098
13	0.608	26	0.661	39	0.428	52	0.559	65	0.054

- $x^{(0)} = (1.3, 0.65, 0.65, 0.7, 0.6, 3, 5, 7, 2, 4.5, 5.5)$
- $\|\gamma^*\|^2 = 4.01377 \dots 10^{-2}$

APPENDIX B
PROGRAM LISTING

```
PROGRAM ALML
C*****
C
C PROGRAM
C -----
C ALML (ACCELERATED LEVENBERG-MARQUARDT METHOD WITH LINE SEARCH)
C
C DESCRIPTION
C -----
C THIS PROGRAM TESTS CODES FOR THE NONLINEAR LEAST SQUARES PROBLEMS
C OF M RESIDUALS IN N VARIABLES. THE DRIVER READS IN DATA, CALLS
C THE NONLINEAR LEAST-SQUARES SOLVER, AND FINALLY PRINTS OUT
C INFORMATION ON THE PERFORMANCE OF THE SOLVER.
C
C DIMENSION LIMITATION
C -----
C M<=MM, N<=NN, WHERE M IS # OF THE RESIDUES AND N IS # OF THE
C VARIABLES. THE CURRENT VALUE OF MM AND NN ARE 65 AND 40,
C RESPECTIVELY. TO CHANGE THE DIMENSION LIMITATION, MODIFY THE
C PARAMETER(MM=65,NN=40,...) STATEMENTS THROUGHOUT THE PACKAGE
C
C SUBPROGRAMS CALLED
C -----
C SOLVER ..... NONLINEAR LEAST-SQUARES SOLVER
C SETPAR ..... SET/INPUT PARAMETERS <----- ENTRY OF SUBROUTINE IO
C SETPRB ..... SET/INPUT PROBLEM INFO <---- ENTRY OF SUBROUTINE IO
C SETTAB ..... SET THE SUMMARY TABLE <----- ENTRY OF SUBROUTINE IO
C PRTAB ..... PRINT THE TABLE <----- ENTRY OF SUBROUTINE IO
C
C INPUT/OUTPUT
C -----
C SEE SUBROUTINE IO
C
C PROGRAMMED BY
C -----
C WEI YUAN, COMPUTER SCIENCE DEPT., OKLAHOMA STATE UNIV., 1992
C
C*****
C --- USING DOUBLE PRECISION ...
C      IMPLICIT REAL*8 (A-H,O-Z)
C
C --- CURRENTLY ALLOWED: 65 RESIDUALS, 40 VARIABLES ...
C      PARAMETER(MM=65, NN=40)
C
C --- ARRAY: VARIABLE VECTOR...
C      DIMENSION X(NN)
C
C --- FUNCTIONS CALLED: A NONLINEAR LEAST SQUARES SOLVER ...
C      INTEGER SOLVER
C
C --- SET PARAMETERS ...
C      CALL SETPAR
```

```

C --- BEGIN THE MAIN LOOP ...
10  CALL SETPRB(M,N,NTRIES)
C      ... SET/INPUT PROBLEM INFORMATION

C ----- BEGIN THE INNER LOOP ...
      DO 20 K = 1, NTRIES
        INFO=SOLVER(M,N,K,X)
C      ... CALL THE SOLVER SOLVE THE LEAST SQUARES PROBLEM
        CALL SETTAB(M,N,INFO)
C      ... STORE OUTPUTS INTO THE TABLES & INCREMENT THE SUMS

C ----- ON NEXT TRY, INITIAL POINT WILL BE TIMED BY 10*(K-1)
20  CONTINUE

C --- END OF THE MAIN LOOP ...
      IF(NTRIES.GT.0) GOTO 10

C --- PRINT THE SUMMARY TABLES ...
      CALL PRTAB
      END

      INTEGER FUNCTION SOLVER(M,N,K,X)
C*****
C
C  PURPOSE
C  -----
C  THIS IS A NON-LINEAR LEAST-SQUARES SOLVER IMPLEMENTING AN
C  ACCELERATED LEVENBERG-MARQUARDT METHOD WITH LINE SEARCH
C
C  FORMAL PARAMETERS
C  -----
C  M ..... NUMBER OF RESIDUAL FUNCTIONS
C  N ..... NUMBER OF VARIABLES
C  K ..... NUMBER OF TRIES TO THIS SOLVER ON SAME PROBLEM
C  X ..... VARIABLE VECTOR
C
C  RETURN VALUES
C  -----
C  1 ... RELATIVE FUNCTION REDUCTION IS LESS THAN FTOL
C  4 ... NORM OF GRADIENT IS LESS THAN GTOL
C  5 ... # OF FUNCTION EVALUATIONS IS OVER 100*(N+1)
C
C  SUBPROGRAMS CALLED
C  -----
C  INIT ..... INITIALIZE COUNTERS & GET INITIAL POINT
C  PRINT .... PRINT OUT THE INITIAL POINT
C  DIRECT ... CALCULATE THE SEARCH DIRECTION
C  LNSRCH ... PERFORM LINE SEARCH ALONG SEARCH DIRECTION
C  PROUT ... PRINT THE OUTPUT OF THIS CALL
C
C*****
      IMPLICIT REAL*8(A-H, O-Z)

C --- FORMAL PARAMETER ...
      REAL*8 X(N)

C --- CONSTANTS ...
      PARAMETER(MM=65,NN=40)

C --- LOCAL ARRAYS: RESIDUAL, JACOBIAN, SEARCH DIRECTION ...
      REAL*8 FVEC(MM), FJAC(MM,NN), P(NN)

C --- FUNCTIONS CALLED: DO LINE SEARCH & COMPUTE SEARCH DIRECTION ...
      INTEGER LNSRCH, DIRECT

C --- INITIALIZE COUNTERS & GET/PRINT INITIAL POINT ...
      CALL INIT(M,N,X,K)

```

```

C --- BEGIN THE LOOP ...
      SOLVER=0
10   CONTINUE

C ----- CALCULATE THE SEARCH DIRECTION ...
      SOLVER=DIRECT(M,N,X,FVEC,FJAC,P)

C ----- PERFORM LINE SEARCH ALONG THE DIRECTION P ...
      IF(SOLVER.EQ.0) SOLVER=LNSRCH(M,N,X,P,FVEC,FJAC)

C --- END OF THE LOOP
      IF(SOLVER.EQ.0) GOTO 10

C --- PRINT THE OUTPUT OF THIS CALL ...
      CALL PROUT(M,N,X,SOLVER)
      END

      SUBROUTINE IO(N,X)
C*****
C
C  PURPOSE
C  -----
C  INPUT/OUTPUT
C
C  ENTRIES
C  -----
C  SETPAR ... SET/INPUT PARAMETERS
C  SETTAB ... STORE OUTPUTS INTO THE TABLE & INCREMENT THE SUMS
C  SETPRB ... SET/INPUT PROBLEM INFORMATION
C  PRTAB ... PRINT THE SUMMARY TABLE
C  PROUT ... PRINT THE OUTPUT OF A CALL
C  PRITER ... PRINT THE OUTPUT OF A ITERATION
C
C*****
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER(MAXCALL=54,NREAD=5,NWRITE=6)

C --- ARRAYS ...
      INTEGER ITABLE(7,MAXCALL)
      REAL*8 FTABLE(2,MAXCALL),X(N),MU

C --- COMMON VARIABLES:
      COMMON /REFNUM/NPROB, NFEV, NJEV, ITER
C          ... PROB. NO., #FUNC., #JACOBIAN EVALS., #ITERATIONS
      &      /TOLS/FTOL,GTOL /PAR/IPAR(5)
C          ... TOLERANCES, PARAMETERS
      &      /NORMS/FNORM,GNORM,FNORMO /MCHEPS/EPSMCH,DWARF
C          ... NORMS, MACHINE PARAMETERS
      &      /LSPAR/PAR,DF0,RATIO,ALPHA0,ALPHA,SIZE,IDIR,COS
C          ... INFORMATION TRANSFERRING BETWEEN DIRECT & LNSRCH
      &      /NOISE/YNOISE
C          ... NOISE OF DATA POINTS

C --- SAVING DATA ...
      SAVE /NORMS/, /TOLS/, /MCHEPS/, /NOISE/,
      &      ITABLE,FTABLE, NFEVS, NJEVS, ITERS, NCALLS, INFO
C          ... SUMMARY TABLES, CUMULATIVE COUNTERS, # OF CALLS, EXIT INFO

      DATA NCALLS, EPSMCH /0, 1.0D0/

C *** SET/INPUT PARAMETERS ***
      ENTRY SETPAR
C ----- COMPUTE THE MACHINE EPSILON ...
10   EPSMCH=EPSMCH/2.0D0
      IF(1.0D0+EPSMCH .GT. 1.0D0) GOTO 10
      EPSMCH=EPSMCH*2.0D0

```



```

C ----- COMPUTE THE SMALLEST POSITIVE NUMBER ...
      T=EPSMCH
20    DWARF=T
      T=T/2.0D0
      IF(T.GT.0.0D0) GOTO 20

C ----- PROMPT & READ USER INPUT FOR FLOATING PARAMETERS ...
      PRINT *,'=> USE DEFAULT PARAMETERS(0=USE/1=NOT USE)?'
      READ(NREAD,100)ISDFLT
100   FORMAT(I2)

      IF(ISDFLT.EQ.0) THEN
C ----- SET DEFAULT VALUES OF PARAMETERS ...
      FTOL=1.0D-10
C      ... TOLERANCE ON FUNCTION
      GTOL=FTOL
C      ... TOLERANCE ON GRADIENT
      IPAR(1)=1
C      ... =0/1: TURN OFF/ON RUHE'S ACCELERATION SWITCH
      IPAR(2)=1
C      ... =0: RHO=0.01, SIGMA=0.1
C      ... =1: DYNAMICALLY CONTROL LINE SEARCH ACCURACY
C      ... =-1: RHO=0.008, SIGMA=0.7
      IPAR(3)=2
C      ... AMOUNT OF INFORMATION TO BE PRINTED:
C      ... =0: SUMMARY TABLE ONLY
C      ... =1: SUMMARY TABLE & OUTPUT FOR EACH CALL
C      ... =2: PLUS DETAIL INFO IN EACH ITERATION
      IPAR(5)=0
C      ... =0: DO NOT CHECK POSSIBLE OVERFLOW
C      ... =1: CHECK POSSIBLE OVERFLOW
      YNOISE=0.0D0
C      ... NOISE OF DATA POINTS
      ELSE
C ----- PROMPT/READ USER INPUT FOR PARAMETERS ...
      PRINT *,'=> FTOL, GTOL, NOISE'
      READ(NREAD,200) FTOL, GTOL, YNOISE
      PRINT *,
      &      '=> ACCEL.(0/1), DYN.(-1/0/1), PRINT(0/1/2), CHKOVF(0/1)?'
      READ(NREAD,300) IPAR(1),IPAR(2),IPAR(3),IPAR(5)
200   FORMAT(3D9.5)
300   FORMAT(4I3)
      ENDIF
      PRINT *,'=> PROBLEM #, N, M, #TRIES'
      RETURN

C *** SET/INPUT PROBLEM INFORMATION ***
      ENTRY SETPRB(M,N,NTRIES)
C ----- INPUT PROBLEM NO., DIMENSIONS AND #TRIES ...
      READ(NREAD,400)NPROB,N,M,NTRIES
400   FORMAT(4I5)

      IF(NPROB.LE.0) NTRIES=0
C      ... NO MORE CALL IF NPROB=0
      IPAR(4)=100*(N+1)
C      ... SET MAXIMUM # OF FUNCTION EVALUATIONS
      RETURN

C *** STORE OUTPUTS INTO THE TABLE & INCREMENT THE SUMS ***
      ENTRY SETTAB(M,N,INFO)
C ----- INCREMENT CALL COUNTER ...
      NCALLS=NCALLS+1

C ----- STORE THE OUTPUT OF THIS CALL INTO THE TABLE ...
      ITABLE(1,NCALLS)=NPROB
      ITABLE(2,NCALLS)=N
      ITABLE(3,NCALLS)=M

```

```

        ITABLE(4,NCALLS)=NFEV
        ITABLE(5,NCALLS)=NJEV
        ITABLE(6,NCALLS)=ITER
        ITABLE(7,NCALLS)=INFO
        FTABLE(1,NCALLS)=FNORM
        FTABLE(2,NCALLS)=GNORM

C ----- INCREMENT THE SUMS ...
        NFEVS=NFEVS+NFEV
        NJEVS=NJEVS+NJEV
        ITERS=ITERS+ITER
        RETURN

C *** PRINT THE SUMMARY OF ALL CALLS TO THE SOLVER ***
        ENTRY PRTAB
C ----- PRINT THE TITLE OF THE SUMMARY TABLE ...
        WRITE(NWRITE,500) NCALLS,
        & 'NPROB  N  M  NFEV  NJEV  NITR  INFO  FINAL L2 NORM  GNORM'
500    FORMAT(/18X,10HSUMMARY OF,I3,14H CALLS TO ALML/1X,62A)

C ----- PRINT THE BODY OF THE SUMMARY TABLE ...
        WRITE(NWRITE,600)
        &      ((ITABLE(I,J),I=1,7),FTABLE(1,J),FTABLE(2,J),J=1,NCALLS)
600    FORMAT(I6,2I4,1X,4I6,D15.7,D8.1)

C ----- PRINT TOTAL ...
        WRITE(NWRITE,700) 'TOTAL:',NFEVS,NJEVS,ITERS
700    FORMAT(A15,3I6)

C ----- PRINT PARAMETERS ...
        WRITE(NWRITE,800)
        &      'FTOL=',FTOL, ',  GTOL=',GTOL, ',  NOISE=',YNOISE,
        &      ',  ACCE.=',IPAR(1), ',  DYN.=',IPAR(2),
        &      ',  PRINT=',IPAR(3), ',  CHKOVF=',IPAR(5)
800    FORMAT(/3(A8,D6.1),4(A8,I1))
        RETURN

C *** PRINT THE INITIAL INFORMATION FOR THIS CALL ***
        ENTRY PRINT(M,N,X)
        IF(IPAR(3).EQ.0) RETURN
        PRINT 900, '  PROBLEM', NPROB, 'N=', N, 'M=', M,
        &      '  INITIAL POINT', (X(I),I=1,N)
900    FORMAT(/ A9,I3,2(4X,A2,I3),/ 5X,A13/(5X,5D15.7))
        RETURN

C *** PRINT THE OUTPUT OF A CALL ***
        ENTRY PROUT(M,N,X,INFO)
        IF(IPAR(3).EQ.0) RETURN
C
        ... NO PRINTOUT IF THE SWITCH IS OFF

        PRINT 1000, FNORMO, FNORM, GNORM, NFEV, NJEV, ITER, INFO, (X(I), I=1, N)
1000   FORMAT(5X,33H INITIAL L2 NORM OF THE RESIDUALS, D19.7/
        &      5X,33H FINAL L2 NORM OF THE RESIDUALS  , D19.7/
        &      5X,33H FINAL L2 NORM OF THE GRADIENT    , D19.7/
        &      5X,33H NUMBER OF FUNCTION EVALUATIONS   , I10 /
        &      5X,33H NUMBER OF JACOBIAN EVALUATIONS    , I10 /
        &      5X,33H NUMBER OF ITERATIONS              , I10 /
        &      5X,33H EXIT INFO                          , I10 /
        &      5X,27H FINAL APPROXIMATE SOLUTION /(5X,5D15.7))
        RETURN

C *** PRINT THE OUTPUT OF A ITERATION ***
        ENTRY PRITER(MU)
        IF(IPAR(3).NE.2) RETURN
C
        ... NO PRINTOUT IF THE SWITCH IS OFF

C ----- PRINT TITLE ...
        IF(MOD(ITER,23).EQ.1) PRINT 1100,

```

```

      &      'IT NF NJ D RATIO      FNORM GNORM ALPHAO ALPHA  PAR
      & SIZE  COS  MU'
1100  FORMAT(/1X,77A)

C ---- PRINT INFORMATION FOR A ITERATION ...
      PRINT 1200, ITER, NFEV, NJEV, IDIR, RATIO, FNORM, GNORM,
      &      ALPHAO, ALPHA, PAR, SIZE, COS, MU
1200  FORMAT(I3, 2I4, I2, D7.1, D11.5, 7D7.1)
      RETURN
      END

```

```

      INTEGER FUNCTION DIRECT(M,N,X,FVEC,FJAC,P)
C*****
C
C  PURPOSE
C  -----
C  FIND A LEVENBERG-MARQUARDT SEARCH DIRECTION USING C-G
C  ACCELERATION WHEN NECESSARY
C
C  FORMAL PARAMETERS
C  -----
C  M ..... # OF RESIDUALS
C  N ..... # OF VARIABLES
C  X ..... VARIABLE VECTOR
C  FVEC ... VECTOR OF RESIDUALS
C  FJAC ... JACOBIAN MATRIX
C  P ..... VECTOR OF SEARCH DIRECTION
C
C  MAJOR SUBPROGRAMS CALLED
C  -----
C  FCN ..... FUNCTION EVALUATION
C  JAC ..... JACOBIAN EVALUATION
C  QRFAC .... QR FACTORIZATION
C  NQTF .... STORE THE FIRST N ELEMENTS OF QT*F IN QTF
C  LMPAR .... DETERMINE THE LEVENBERG-MARQUARDT PARAMETER
C
C  OTHER SUBPROGRAMS CALLED
C  -----
C  VADD,VMAX,VMUL,VNCPY,VSCALE,VCOPY,INNER,ENORM ... VECTOR UTILITIES
C  MCOPY,MVMUL ..... A MATRIX UTILITY
C  (SEE FILE 'UTIL.FOR' FOR MORE DETAILS)
C
C  MAJOR REFERENCES
C  -----
C  1. MORE, J. J., THE LEVENBERG-MARQUARDT ALGORITHM: IMPLEMENTATION
C     AND THEORY, IN LECTURE NOTES IN MATHEMATICS, NO. 630-NUMERICAL
C     ANALYSIS, EDITED BY G. WATSON, SPRINGER-VERLAG, NEW YORK,
C     PP. 105-116, 1978.
C  2. RUHE A., ACCELERATED GAUSS-NEWTON ALGORITHMS FOR NONLINEAR
C     LEAST SQUARES PROBLEMS, BIT, VOL.19, PP.356-367, 1979.
C
C*****
      IMPLICIT REAL*8 (A-H,O-Z)

C --- FORMAL PARAMETERS ...
      REAL*8 FVEC(M),FJAC(M,N),X(N),P(N)

C --- CONSTANT ...
      PARAMETER(MM=65,NN=40)

C --- LOCAL ARRAYS, FUNCTION CALLED ...
      INTEGER IPVT(NN)
C     ... PERMUTATION VECTOR
      REAL*8 G(NN),QTF(NN),ACNORM(NN),RDIAG(NN),W(NN),FJAC1(MM,NN),
      &      JPNORM,INNER
C     ... GRADIENT, 1ST N ELEMS OF QT*F, COLUMN NORMS OF FJAC, ETC

```

```

C --- COMMON VARIABLES: SEE SUBROUTINE IO ...
COMMON /REFNUM/NPROB,NFEV,NJEV,ITER
&      /TOLS/FTOL,GTOL
&      /NORMS/FNORM,GNORM,FNORMO
&      /PAR/IPAR(5)
&      /LSPAR/PAR,DFO,RATIO,ALPHA0,ALPHA,SIZE,IDIR,COS
&      /JP/FJACP(MM),JPNORM

C --- SAVING DATA ...
REAL*8 OLDP(NN),D(NN),XI(6),ETA(3)
SAVE FACTOR,XI,ETA,OLDP,ODELTA,D,IC,ORATIO,DPNORM, /LSPAR/,/JP/

C --- PARAMETERS FOR ADJUSTING SIZE OF TRUST REGION ...
DATA FACTOR,XI/1.0D2,5.0D-2,0.2D0,1.0D3,0.1D0,5.0D0,1.0D3/

C --- PARAMETERS FOR CONTROLLING USE OF RUHE'S STEP ...
DATA ETA/0.5D0, 2*0.2D0/

C --- INITIALIZE RETURN INFO ...
DIRECT=0

C --- INCREMENT ITERATION COUNTER ...
ITER=ITER+1

      IF(ITER.EQ.1) THEN
C ----- EVALUATE THE FUNCTION AT THE STARTING POINT...
      CALL FCN(M,N,X,FVEC,FNORMO)
      FNORM=FNORMO
      IF(IPAR(3).EQ.2) PRINT '(A8,D19.8)', 'FNORMO=',FNORMO

C ----- CALCULATE THE JACOBIAN MATRIX AT THE STARTING POINT...
      CALL JAC(M,N,X,FJAC)

C ----- INITIALIZATION ...
      IDIR=0
C          ... INDICATOR OF SEARCH DIRECTION
      PAR=0.0D0
C          ... LM PARAMETER
      IC=1
C          ... COUNTER OF THE RUHE STEPS
      ENDIF

C --- COPY FJAC TO FJAC1...
      CALL MCOPY(M,N,FJAC1,FJAC)

C --- CALCULATE THE QR FACTORIZATION OF THE JACOBIAN MATRIX ...
      CALL QRFAC(M,N,FJAC,IPVT,RDIAG,ACNORM)
C          ... FJAC CHANGED HERE!

C --- STORE THE FIRST N ELEMENTS OF Q^T*F IN QTF ...
      CALL NQTF(M,N,FVEC,FJAC,QTF,RDIAG)

C --- CALCULATE THE DIAGONAL MATRIX & THE SIZE OF THE TRUST REGION ...
      IF(ITER.EQ.1) THEN
C ----- D=ACNORM ...
      DO 10 J=1,N
          D(J)=ACNORM(J)
          IF(ACNORM(J).EQ.0.0D0) D(J)=1.0D0
10      CONTINUE

C ----- CALCULATE INITIAL SIZE OF THE TRUST REGION: SIZE=||D*X|| ...
      CALL VMUL(N,D,X,W)
      DXNORM=ENORM(N,W)
      SIZE=FACTOR
      IF(DXNORM.NE.0.0D0) SIZE=SIZE*DXNORM
      ELSE

C ----- UPDATE THE SIZE OF THE TRUST REGION ...
      IF(RATIO.LT.XI(1)) THEN

```

```

C          ... CONVERGENCE RATE NOT SATISFACTORY
      SIZE=SIZE*DMAX1(XI(2),DMIN1(ALPHA,XI(3)*COS))
      PAR=PAR/ALPHA
      ELSE IF(PAR.EQ.0.ODO .OR. RATIO.GT.XI(4)) THEN
C          ... CONVERGENCE RATE SATISFACTORY
      SIZE=DMIN1(XI(5),XI(6)*ALPHA)*DPNORM
      PAR=PAR/XI(5)
      ENDIF
      CALL VMAX(N,D,ACNORM,D)
C          ... D=MAX(D,ACNORM)
      ENDIF

C --- DETERMINE THE LM PARAMETER & LM DIRECTION ...
      CALL LMPAR(M,N,FJAC,IPVT,D,QTF,SIZE,PAR,P,G)

C --- CALCULATE THE GRADIENT NORM AND TEST FOR CONVERGENCE ...
      GNORM=ENORM(N,G)
      IF(GNORM.LE.GTOL) THEN
        DIRECT=4
        RETURN
      ENDIF

C --- DIRECTIONAL DERIVATIVE F'(0)=G^T*P ...
      CALL VNCOPY(N,P,P)
      DFO=INNER(N,G,P)

C --- CALCULATE NORMS OF P, D*P & COS OF ANGLE BETWEEN P AND G ...
      PNORM=ENORM(N,P)
      COS=-DFO/PNORM/GNORM
      CALL VMUL(N,D,P,W)
      DPNORM=ENORM(N,W)

C --- CHECK THE C-G ACCELERATION SWITCH ...
      IF(N.EQ.1.OR.IPAR(1).EQ.0) GOTO 20

C --- DON'T USE C-G ACCELERATION, IF CURRENT STEP IS LM'S, OR
C --- THE LAST RUHE'S DIRECTION IS NOT GOOD ENOUGH ...
      IF(PAR.GT.0.ODO .OR. IDIR.EQ.1.AND.RATIO.LT.ETA(1)*ORATIO) THEN
        IC=1
        IDIR=0
        ORATIO=RATIO
        GOTO 20
      ENDIF

C --- IF PREVIOUS GN DIR. IS NOT CLOSE TO -G OR GN DIR IS GOOD ENOUGH,
C --- DON'T USE RUHE'S DIRECTION ...
      IF(IDIR.EQ.0 .AND. (COS.LT.ETA(2).OR.RATIO.GT.ETA(3))) IC=1

      IDIR=0
      DELTA=ENORM(N,QTF)**2
      IF(MOD(IC,N).NE.1) THEN
C ----- RUHE'S ACCELERATION: P=P+(DELTA/ODELTA)*OLDP ...
        CALL VSCALE(N,DELTA/ODELTA,OLDP,W)
        CALL VADD(N,W,P,P)

C ----- DIRECTIONAL DERIVATIVE F'(0)=G^T*P ...
        DFO=INNER(N,G,P)

C ----- CALCULATE NORMS OF P, D*P & COS OF ANGLE BETWEEN P AND G ...
        PNORM=ENORM(N,P)
        COS=-DFO/PNORM/GNORM
        CALL VMUL(N,D,P,W)
        DPNORM=ENORM(N,W)
        IDIR=1
      ENDIF

C --- UPDATE ORATIO, ODELTA, OLDP AND IC ...
      ORATIO=RATIO

```

```

ODELTA=DELTA
CALL VCOPY(N,OLDP,P)
IC=IC+1

C --- FJACP = FJAC*P ...
20 CALL MVMUL(M,N,FJAC1,P,FJACP)
   JPNORM=ENORM(M,FJACP)
   END

      INTEGER FUNCTION LNSRCH(M,N,X,P,FVEC,FJAC)
C*****
C
C PURPOSE
C -----
C PERFORM LINE SEARCH ALONG THE SEARCH DIRECTION
C
C FORMAL PARAMETERS
C -----
C M ..... # OF RESIDUALS
C N ..... # OF VARIABLES
C X ..... VARIABLE VECTOR
C P ..... VECTOR OF SEARCH DIRECTION
C FVEC ..... VECTOR OF RESIDUALS
C FJAC ..... JACOBIAN MATRIX
C
C MAJOR SUBPROGRAMS CALLED
C -----
C FCN ..... FUNCTION EVALUATION
C JAC ..... JACOBIAN EVALUATION
C T ..... FIND A POINT IN INTERVAL T WHEN INTERPOLATING
C E ..... FIND A POINT IN INTERVAL E WHEN EXTRAPOLATING
C Q ..... FIND A MIN. OF Q(ALPHA) USING NEWTON METHOD
C SETACC ... SET THE ACCURACY PARAMETERS FOR LINE SEARCH
C PRITER ... PRINT INFO FOR EACH ITERATION
C
C OTHER SUBPROGRAMS CALLED
C -----
C VADD,VSCALE,VCOPY,INNER,ENORM ... VECTOR UTILITIES
C MVMUL ..... MATRIX TIMES VECTOR
C (SEE FILE 'UTIL.FOR' FOR MORE DETAILS)
C
C MAJOR REFERENCE
C -----
C AL-BAALI, M. AND FLETCHER, R., AN EFFICIENT LINE SEARCH FOR
C NONLINEAR LEAST SQUARES, JOURNAL OF OPTIMIZATION THEORY AND
C APPLICATIONS, VOL.48, PP.359-377, 1986.
C
C*****
      IMPLICIT REAL*8 (A-H, O-Z)

C --- FORMAL ARGUMENTS ...
      REAL*8 X(N),FVEC(M),P(N),FJAC(M,N)

C --- CONSTANTS ...
      PARAMETER(MM=65,NN=40)

C --- LOCAL VARIABLES, ARRAYS, AND FUNCTION CALLED ...
      REAL*8 X1(NN),FX1(MM),AX(NN),FAX(MM),W(NN),MU,JPNORM,INNER
      EQUIVALENCE(X1,W)

C --- COMMON VARIABLES ...
      COMMON /REFNUM/NPROB,NFEV,NJEV,ITER /PAR/IPAR(4)
      & /NORMS/FNORM,GNORM,FNORMO /TOLS/FTOL,GTOL
      & /LSPAR/PAR,DFO,RATIO,ALPHA,ALPHA0,ALPHA,SIZE,IDIR,COS
      & /JP/DFAVEC(MM),JPNORM
      & /STEPSZ/ISLONG

```

```

C --- SAVING DATA ...
      SAVE REDUCE, OALPHA, ISAME, THETA, GAMMA, MAXITR, SATIS, /STEPSZ/

C --- PARAMETERS CONTROLLING LINE SEARCH ...
      DATA THETA, GAMMA, SATIS, MAXITR/3.0D1, 0.5D-1, 0.8D0, 10/

C --- STATEMENT FUNCTION ...
      DF(F1,F2)=0.5D0*(F1-F2)*(F1+F2)

C --- SET THE ACCURACY PARAMETERS FOR LINE SEARCH ...
      CALL SETACC(RHO,SIGMA)

C --- INITIALIZE VARIABLES...
      LNSRCH=0
      ISEND=0
      NFEVO=NFEV
      FV=0.5D0*FNORM**2
      EPS=FTOL*DMAX1(1.0D0,FV)
      MU=-FV/RHO/DF0
      IF(ITER.GT.1) MU=MU*DMIN1(1.0D0,THETA*RATIO)
      DFA=DF0
      A=0.0D0
      B=MU
      FANORM=FNORM
      CALL VCOPY(N,AX,X)
      CALL VCOPY(M,FAX,FVEC)

C --- CHOOSE THE INITIAL STEPLENGTH ALPHA ...
      IF(ITER.EQ.1) THEN
        OALPHA=1.0D0
        REDUCE=0.5D0*FV
      ELSE IF(ISAME.EQ.1) THEN
        ALPHA=OALPHA
        GOTO 5
C
      ELSE IF(PAR.EQ.0.0D0.AND.IDIR.EQ.0) THEN
        ALPHA=1.0D0
        GOTO 5
C
      ELSE
        ... GN STEP
      ENDIF
      ALPHAF=-DMAX1(REDUCE,3.0D1*EPS)/DF0
      ALPHAL=-DF0/JPENORM**2
      ALPHA=DMAX1(ALPHAF,ALPHAL)
      IF(ALPHA.GE.1.0D0) ALPHA=DMIN1(DMIN1(ALPHAF,ALPHAL),1.0D0)
      ALPHA=DMAX1(GAMMA,DMIN1(ALPHA,MU))

5   ALPHA=ALPHA

C --- EVALUATE F(X1) AND ITS NORM, WHERE X1=X+ALPHA*P ...
10  CALL VSCALE(N,ALPHA,P,X1)
      CALL VADD(N,X,X1,X1)
      CALL FCN(M,N,X1,FX1,FNORM1)

C --- IF THE SWITCH IS ON AND STEPLENGTH IS TOO LONG ...
      IF(IPAR(5).EQ.1.AND.ISLONG.EQ.1) THEN
        ALPHA=0.5D0*ALPHA
C
        ... REDUCE STEPLENGTH
        IF(IPAR(3).EQ.2)
          & PRINT *, 'POSSIBLE OVERFLOW, STEPLENGTH REDUCED!'
          GOTO 10
      ENDIF

C --- TEST FOR TERMINATION ...
      ISEND=1
      IF(NFEV.GE.IPAR(4)) THEN
        LNSRCH=5
C
        ... TOO MANY FUNCTION EVALUATIONS
        GOTO 15

```

```

ENDIF
IF(NFEV-NFEVO.GT.MAXITR .OR. (FNORM1/FNORM)**2.LE.SATIS) GOTO 15
C      ... TOO MANY ITERATIONS OR REDUCE IS GOOD ENOUGH
IF(DABS((A-ALPHA)*DFA).LE.1.0D1*FTOL) GOTO 15
C      ... TOO SMALL POSSIBLE STEPLENGTH
ISEND=0

C --- CHOOSE ALPHA FROM INTERVAL T(A,ALPHA,B) ...
IF(DF(FNORM,FNORM1).LT.-RHO*ALPHA*DFO.OR.FNORM1.GE.FANORM) THEN
  REDUCE=DF(FANORM,FNORM1)
  TMP=A+0.5DO*(ALPHA-A)/(1.0DO+REDUCE/(ALPHA-A)/DFA)
  CALL Q(M,TMP,A,ALPHA,FAX,FX1,DFAVEC)
  B=ALPHA
  ALPHA=T(A,B,TMP)
  GOTO 10
ENDIF

C --- UPDATE AX,FAX,FANORM, ETC ...
15  IF(FNORM1.LT.FANORM) THEN
  CALL VCOPY(N,AX,X1)
  REDUCE=DF(FANORM,FNORM1)
  FANORM=FNORM1
ENDIF

C --- EVALUATE DFA=F'(ALPHA) ...
CALL JAC(M,N,X1,FJAC)
CALL MVMUL(M,N,FJAC,P,DFAVEC)
DFA=INNER(M,DFAVEC,FX1)

C --- CHOOSE ALPHA FROM INTERVAL E(A,ALPHA,B) ...
IF(ISEND.NE.1 .AND. DABS(DFA).GT.-SIGMA*DFO) THEN
  TMP=ALPHA+0.5DO*(A-ALPHA)/(1.0DO-REDUCE/(A-ALPHA)/DFA)
  CALL Q(M,TMP,ALPHA,A,FX1,FAX,DFAVEC)
  IF((B-A)*DFA.LT.0.0DO) THEN
    A=ALPHA
    ALPHA=E(A,B,TMP)
  ELSE
    B=A
    A=ALPHA
    ALPHA=T(B,ALPHA,TMP)
  ENDIF
  CALL VCOPY(M,FAX,FX1)
  GOTO 10
ENDIF
CALL VCOPY(M,FAX,FX1)

C --- TESTS FOR TERMINATION ...
REDUCE=DF(FNORM,FANORM)
IF(LNSRCH.EQ.0 .AND. REDUCE.LE.EPS) LNSRCH=1

C --- COMPUTE GNORM AT THE FINAL SOLUTION FOR SUMMARY TABLE ...
IF(LNSRCH.NE.0) THEN
  DO 30 J=1,N
    W(J)=0.0DO
  DO 20 I=1,M
    W(J)=W(J)+FJAC(I,J)*FAX(I)
20  CONTINUE
30  CONTINUE
  GNORM=ENORM(N,W)
  NJEV=NJEV-1
ENDIF

C --- UPDATE X,FVEC AND FNORM, ETC ...
CALL VCOPY(N,X,AX)
CALL VCOPY(M,FVEC,FAX)
RATIO=1.0DO-(FANORM/FNORM)**2
FNORM=FANORM
ISAME=0

```



```

IF(ITER.GT.1 .AND. ALPHA.EQ.OALPHA) ISAME=1
OALPHA=ALPHA

C --- PRINT INFO FOR EACH ITERATION IF THE SWITCH IS ON...
CALL PRITER(MU)
END

REAL*8 FUNCTION T(A,B,C)
C*****
C
C   CHOOSE C IN THE INTERVAL T(A,B)
C
C*****
REAL*8 A,B,C,AA,BB,T1,T2
SAVE T1,T2
DATA T1,T2/0.5D-1,0.5D0/

AA=DMIN1(A+T1*(B-A),B-T2*(B-A))
BB=DMAX1(A+T1*(B-A),B-T2*(B-A))
T=DMIN1(BB,DMAX1(AA,C))
END

REAL*8 FUNCTION E(ALPHA,B,C)
C*****
C
C   RETURN A NUMBER IN THE INTERVAL E(ALPHA,B,C)
C
C*****
REAL*8 ALPHA,B,C,AA,BB,T3,T4
SAVE T3,T4
DATA T3,T4 /0.5D-1,0.5D0/

AA=DMIN1(ALPHA+T3*(B-ALPHA),B-T4*(B-ALPHA))
BB=DMAX1(ALPHA+T3*(B-ALPHA),B-T4*(B-ALPHA))
E=DMIN1(DMAX1(AA,C),BB)
END

SUBROUTINE Q(M,ALPHA,A,B,FA,FB,DFA)
C*****
C
C   MINIMIZE Q(ALPHA) BY USING THE NEWTON-RAPHSON METHOD
C
C*****
IMPLICIT REAL*8 (A-H,O-Z)
REAL*8 FA(M),FB(M),DFA(M)
SAVE MAXITR, SIGMA, DALPHA
DATA MAXITR, SIGMA /15,0.1D0/
C   ... MAX #ITERS, ACCURACY PARA

K=0
10 K=K+1
DQ=0.0D0
D2Q=0.0D0
DO 20 I=1,M
TMP=ALPHA-A
D2QI=2*((FB(I)-FA(I))/(B-A)-DFA(I))/(B-A)
DQI=DFA(I)+D2QI*TMP
QI=FA(I)+(DFA(I)+0.5D0*D2QI*TMP)*TMP
DQ=DQ+QI*DQI
D2Q=D2Q+QI*D2QI+DQI*DQI
20 CONTINUE
DALPHA=-DQ/D2Q
IF(DABS(DALPHA).GT.0.1*SIGMA*DABS(A-B) .AND. K.LE.MAXITR)THEN
ALPHA=ALPHA+DALPHA
GOTO 10

```

```

      ENDIF
      END

      SUBROUTINE SETACC(RHO, SIGMA)
C*****
C
C   SET ACCURACY PARAMETERS FOR LINE SEARCH
C
C*****
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON /NORMS/FNORM,GNORM,FNORMO /PAR/IPAR(4)
      SAVE RHO1,RHO2,SIGMA1,SIGMA2
      DATA RHO1,RHO2,SIGMA1,SIGMA2 /0.1D-3, 0.5D-1, 0.6D0, 0.8D0/

      IF(IPAR(2).EQ.1) THEN
C ----- DYNAMICALLY CONTROL LINE SEARCH ACCURACY ...
          OMEGA=GNORM/DMAX1(1.0D0, FNORM)
          RHO=DMAX1(RHO1, RHO2/(1.0D0+OMEGA))
          SIGMA=DMIN1(SIGMA1+OMEGA, SIGMA2)
      ELSE IF(IPAR(2).EQ.-1) THEN
          RHO=0.008
          SIGMA=0.7
      ELSE
C ----- AS SUGGESTED BY AL-BAALI AND FLETCHER ...
          RHO=0.01
          SIGMA=0.1
      ENDIF
      END

      SUBROUTINE LMPAR(M,N,R,IPVT,DIAG,QTB,BOUND,PAR,P,G)
C*****
C
C   * THIS IS A MODIFIED VERSION OF THE MINPACK ROUTINE LMPAR
C
C   * GIVEN AN M BY N MATRIX A, AN N BY N NONSINGULAR DIAGONAL
C   MATRIX D, AN M-VECTOR B, AND A POSITIVE NUMBER DELTA,
C   THE PROBLEM IS TO DETERMINE A VALUE FOR THE PARAMETER
C   PAR SUCH THAT IF P SOLVES THE SYSTEM
C
C           A*P = B ,      SQRT(PAR)*D*P = 0 ,
C
C   IN THE LEAST SQUARES SENSE, AND DPNORM IS THE EUCLIDEAN
C   NORM OF D*P, THEN EITHER PAR IS ZERO AND
C
C           (DPNORM-DELTA) .LE. ERR*DELTA ,
C
C   OR PAR IS POSITIVE AND
C
C           ABS(DPNORM-DELTA) .LE. ERR*DELTA .
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 LMPAR EXPECTS
C   THE FULL UPPER TRIANGLE OF R, THE PERMUTATION MATRIX P,
C   AND THE FIRST N COMPONENTS OF (Q TRANSPOSE)*B. ON OUTPUT
C   LMPAR ALSO PROVIDES AN UPPER TRIANGULAR MATRIX S SUCH THAT
C
C           T   T           T
C           P *(A *A + PAR*D*D)*P = S *S .
C
C   S IS EMPLOYED WITHIN LMPAR AND MAY BE OF SEPARATE INTEREST.
C
C   ONLY A FEW ITERATIONS ARE GENERALLY NEEDED FOR CONVERGENCE

```

```

C   OF THE ALGORITHM. IF, HOWEVER, THE LIMIT OF 10 ITERATIONS
C   IS REACHED, THEN THE OUTPUT PAR WILL CONTAIN THE BEST
C   VALUE OBTAINED SO FAR.
C
C   * PARAMETERS:
C
C   N -- A POSITIVE INTEGER INPUT VARIABLE SET TO THE ORDER OF R.
C
C   R -- AN N BY N ARRAY. ON INPUT THE FULL UPPER TRIANGLE
C       MUST CONTAIN THE FULL UPPER TRIANGLE OF THE MATRIX R.
C       ON OUTPUT THE FULL UPPER TRIANGLE IS UNALTERED, AND THE
C       STRICT LOWER TRIANGLE CONTAINS THE STRICT UPPER TRIANGLE
C       (TRANSPOSED) OF THE UPPER TRIANGULAR MATRIX S.
C
C   LDR -- A POSITIVE INTEGER INPUT VARIABLE NOT LESS THAN N
C         WHICH SPECIFIES THE LEADING DIMENSION OF THE ARRAY R.
C
C   IPVT -- AN INTEGER INPUT ARRAY OF LENGTH N WHICH DEFINES THE
C          PERMUTATION MATRIX P SUCH THAT  $A * P = Q * R$ . COLUMN J OF P
C          IS COLUMN IPVT(J) OF THE IDENTITY MATRIX.
C
C   DIAG -- AN INPUT ARRAY OF LENGTH N WHICH MUST CONTAIN THE
C          DIAGONAL ELEMENTS OF THE MATRIX D.
C
C   QTB -- AN INPUT ARRAY OF LENGTH N WHICH MUST CONTAIN THE FIRST
C          N ELEMENTS OF THE VECTOR  $(Q \text{ TRANSPOSE}) * B$ .
C
C   BOUND -- A POSITIVE INPUT VARIABLE WHICH SPECIFIES AN UPPER
C           BOUND ON THE EUCLIDEAN NORM OF  $D * P$ .
C
C   PAR -- A NONNEGATIVE VARIABLE. ON INPUT PAR CONTAINS AN
C         INITIAL ESTIMATE OF THE LEVENBERG-MARQUARDT PARAMETER.
C         ON OUTPUT PAR CONTAINS THE FINAL ESTIMATE.
C
C   P -- AN OUTPUT ARRAY OF LENGTH N WHICH CONTAINS THE LEAST
C        SQUARES SOLUTION OF THE SYSTEM  $A * P = B$ ,  $\text{SQRT}(\text{PAR}) * D * P = 0$ ,
C        FOR THE OUTPUT PAR.
C
C   G -- THE GRADIENT VECTOR
C
C   SDIAG -- AN OUTPUT ARRAY OF LENGTH N WHICH CONTAINS THE
C           DIAGONAL ELEMENTS OF THE UPPER TRIANGULAR MATRIX S.
C
C   SUBPROGRAMS CALLED ... ENORM, QRSOLV
C
C*****
C   IMPLICIT REAL*8 (A-H,O-Z)
C   INTEGER IPVT(N)
C   REAL*8 BOUND,PAR,R(M,N),DIAG(N),QTB(N),P(N),G(N)
C
C   PARAMETER(NN=40)
C
C   LOCAL VARIABLES ...
C   REAL*8 SDIAG(NN),WA1(NN),WA2(NN)
C
C   COMMON /MCHEPS/EPSMCH,DWARF
C
C   --- RELATIVE ERROR ALLOWED FOR  $\|DIAG * P\| \dots$ 
C   ERR=0.1D0
C
C -- COMPUTE AND STORE IN P THE GAUSS-NEWTON DIRECTION. IF THE
C -- JACOBIAN IS RANK-DEFICIENT, OBTAIN A LEAST SQUARES SOLUTION.
C   NSING = N
C   DO 10 J=1,N
C     WA1(J)=QTB(J)
C     IF(R(J,J).EQ.0.0D0 .AND. NSING.EQ.N) NSING=J-1
C     IF(NSING.LT.N) WA1(J)=0.0D0
10  CONTINUE

```

```

DO 30 K = 1, NSING
  J=NSING-K+1
  WA1(J) = WA1(J)/R(J,J)
  TEMP = WA1(J)
  DO 20 I=1,J-1
    WA1(I)=WA1(I)-R(I,J)*TEMP
20  CONTINUE
30  CONTINUE
    CALL VPVT(N,P,WA1,IPVT)

C -- EVALUATE THE FUNCTION AT THE ORIGIN, AND TEST FOR ACCEPTANCE
C -- OF THE GN DIRECTION.
    CALL VMUL(N,DIAG,P,WA2)
    DPNORM=ENORM(N,WA2)
    FP=DPNORM-BOUND

C --- COMPUTE THE GRADIENT: G=IPVT*R^T*QTF ...
C --- CALCULATE AN UPPER BOUND, PARU, FOR THE ZERO OF THE FUNCTION.
    DO 80 J = 1, N
      SUM = 0.0DO
      DO 70 I = 1, J
        SUM = SUM + R(I,J)*QTB(I)
70  CONTINUE
      L = IPVT(J)
      G(L) = SUM
      WA1(J) = SUM/DIAG(L)
80  CONTINUE
    SGNORM = ENORM(N,WA1)

    IF(FP.LE.ERR*BOUND) THEN
      PAR=0.0DO
      RETURN
    ENDIF

C --- IF THE JACOBIAN IS NOT RANK DEFICIENT, THE NEWTON STEP PROVIDES
C --- A LOWER BOUND, PARL, FOR THE ZERO OF THE FUNCTION. OTHERWISE
C --- SET THIS BOUND TO ZERO.
    PARL = 0.0DO
    IF(NSING .GE. N) THEN
      DO 40 J = 1, N
        L = IPVT(J)
        WA1(J) = DIAG(L)*(WA2(L)/DPNORM)
40  CONTINUE
      DO 60 J = 1, N
        SUM = 0.0DO
        DO 50 I = 1, J-1
          SUM = SUM + R(I,J)*WA1(I)
50  CONTINUE
        WA1(J) = (WA1(J) - SUM)/R(J,J)
60  CONTINUE
      TEMP = ENORM(N,WA1)
      PARL = FP/BOUND/TEMP/TEMP
    ENDIF

    PARU = SGNORM/BOUND
    IF (PARU .EQ. 0.0DO) PARU = DWARF/DMIN1(BOUND,ERR)

C --- IF PAR LIES OUT OF (PARL,PARU), SET PAR TO THE CLOSER ENDPOINT
    PAR = DMIN1(DMAX1(PAR,PARL),PARU)
    IF (PAR .EQ. 0.0DO) PAR = SGNORM/DPNORM

C --- BEGINNING OF AN ITERATION.
    ITER=0
90  CONTINUE
    ITER = ITER + 1

C ----- EVALUATE THE FUNCTION AT THE CURRENT VALUE OF PAR.
    IF (PAR .EQ. 0.0DO) PAR = DMAX1(DWARF,1.0D-3*PARU)

```

```

TEMP = DSQRT(PAR)
CALL VSCALE(N,TEMP,DIAG,WA1)
CALL QRSOLV(N,R,M,IPVT,WA1,QT,P,SDIAG,WA2)
CALL VMUL(N,DIAG,P,WA2)
DPNORM = ENORM(N,WA2)
TEMP = FP
FP = DPNORM - BOUND

C ----- IF THE FUNCTION IS SMALL ENOUGH, ACCEPT THE CURRENT VALUE OF
C ----- PAR. ALSO TEST FOR THE EXCEPTIONAL CASES WHERE PARL IS ZERO OR
C ----- THE # OF ITERATIONS HAS REACHED 10.
IF(DABS(FP).LE.ERR*BOUND .OR. PARL.EQ.0.ODO .AND. FP.LE.TEMP
& .AND. TEMP.LT.0.ODO .OR. ITER.EQ.10) RETURN

C ----- COMPUTE THE NEWTON CORRECTION.
DO 100 J = 1, N
  L = IPVT(J)
  WA1(J) = DIAG(L)*(WA2(L)/DPNORM)
100 CONTINUE
DO 120 J = 1, N
  WA1(J) = WA1(J)/SDIAG(J)
  TEMP = WA1(J)
  DO 110 I = J+1, N
    WA1(I) = WA1(I) - R(I,J)*TEMP
110 CONTINUE
120 CONTINUE
TEMP = ENORM(N,WA1)
PARC = FP/BOUND/TEMP/TEMP

C ----- DEPENDING ON THE SIGN OF THE FUNCTION, UPDATE PARL OR PARU.
IF(FP .GT. 0.ODO) PARL = DMAX1(PARL,PAR)
IF(FP .LT. 0.ODO) PARU = DMIN1(PARU,PAR)

PAR = DMAX1(PARL,PAR+PARC)
C ... COMPUTE AN IMPROVED ESTIMATE FOR PAR
GO TO 90
C ... END OF AN ITERATION.
END

SUBROUTINE QRSOLV(N,R,M,IPVT,DIAG,QT,P,SDIAG,WA)
C*****
C
C * THIS IS A MODIFIED VERSION OF THE MINPACK ROUTINE QRSOLV
C
C * GIVEN AN M BY N MATRIX A, AN N BY N DIAGONAL MATRIX D,
C AND AN M-VECTOR B, THE PROBLEM IS TO DETERMINE AN X WHICH
C SOLVES THE SYSTEM
C
C  $A * X = B$  ,  $D * X = 0$  ,
C
C IN THE LEAST SQUARES SENSE.
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 QRSOLV EXPECTS
C THE FULL UPPER TRIANGLE OF R, THE PERMUTATION MATRIX P,
C AND THE FIRST N COMPONENTS OF (Q TRANSPOSE)*B. THE SYSTEM
C  $A * X = B$ ,  $D * X = 0$ , IS THEN EQUIVALENT TO
C
C  $R * Z = Q * B$  ,  $P * D * P * Z = 0$  ,
C
C WHERE  $X = P * Z$ . IF THIS SYSTEM DOES NOT HAVE FULL RANK,
C THEN A LEAST SQUARES SOLUTION IS OBTAINED. ON OUTPUT QRSOLV

```

```

C     ALSO PROVIDES AN UPPER TRIANGULAR MATRIX S SUCH THAT
C
C           T   T           T
C           P *(A *A + D*D)*P = S *S .
C
C     S IS COMPUTED WITHIN QRSOLV AND MAY BE OF SEPARATE INTEREST.
C
C * PARAMETERS:
C
C     R -- AN N BY N ARRAY. ON INPUT THE FULL UPPER TRIANGLE
C           MUST CONTAIN THE FULL UPPER TRIANGLE OF THE MATRIX R.
C           ON OUTPUT THE FULL UPPER TRIANGLE IS UNALTERED, AND THE
C           STRICT LOWER TRIANGLE CONTAINS THE STRICT UPPER TRIANGLE
C           (TRANSPOSED) OF THE UPPER TRIANGULAR MATRIX S.
C
C     M -- A POSITIVE INTEGER INPUT VARIABLE NOT LESS THAN N
C           WHICH SPECIFIES THE LEADING DIMENSION OF THE ARRAY R.
C
C     IPVT -- AN INTEGER INPUT ARRAY OF LENGTH N WHICH DEFINES THE
C            PERMUTATION MATRIX P SUCH THAT A*P = Q*R. COLUMN J OF P
C            IS COLUMN IPVT(J) OF THE IDENTITY MATRIX.
C
C     DIAG -- AN INPUT ARRAY OF LENGTH N WHICH MUST CONTAIN THE
C            DIAGONAL ELEMENTS OF THE MATRIX D.
C
C     QTB -- AN INPUT ARRAY OF LENGTH N WHICH MUST CONTAIN THE FIRST
C            N ELEMENTS OF THE VECTOR (Q TRANSPOSE)*B.
C
C     X -- AN OUTPUT ARRAY OF LENGTH N WHICH CONTAINS THE LEAST
C           SQUARES SOLUTION OF THE SYSTEM A*X = B, D*X = 0.
C
C     SDIAG -- AN OUTPUT ARRAY OF LENGTH N WHICH CONTAINS THE
C            DIAGONAL ELEMENTS OF THE UPPER TRIANGULAR MATRIX S.
C
C     WA -- A WORK ARRAY OF LENGTH N
C
C*****
C     INTEGER N, M, IPVT(N)
C     REAL*8 R(M,N),DIAG(N),QTB(N),P(N),SDIAG(N),WA(N),
C     &      COS,COTAN,QTBPJ,SIN,SUM,TAN,TEMP
C
C --- COPY R AND (Q TRANSPOSE)*B TO PRESERVE INPUT AND INITIALIZE S.
C --- IN PARTICULAR, SAVE THE DIAGONAL ELEMENTS OF R IN P.
C     DO 20 J = 1, N
C       DO 10 I = J, N
C         R(I,J) = R(J,I)
10    CONTINUE
C       P(J) = R(J,J)
C       WA(J) = QTB(J)
20    CONTINUE
C
C --- ELIMINATE THE DIAGONAL MATRIX D USING A GIVENS ROTATION.
C     DO 100 J = 1, N
C
C ----- PREPARE THE ROW OF D TO BE ELIMINATED, LOCATING THE
C ----- DIAGONAL ELEMENT USING P FROM THE QR FACTORIZATION.
C       L = IPVT(J)
C       IF (DIAG(L) .EQ. 0.0D0) GO TO 90
C       DO 30 K = J, N
C         SDIAG(K) = 0.0D0
30    CONTINUE
C       SDIAG(J) = DIAG(L)
C
C ----- THE TRANSFORMATIONS TO ELIMINATE THE ROW OF D MODIFY ONLY A
C ----- SINGLE ELEMENT OF (Q^T)*B BEYOND 1ST N, WHICH IS INITIALLY 0.
C       QTBPJ = 0.0D0
C       DO 80 K = J, N

```

```

C ----- DETERMINE A GIVENS ROTATION WHICH ELIMINATES THE
C ----- APPROPRIATE ELEMENT IN THE CURRENT ROW OF D.
      IF(SDIAG(K) .EQ. 0.0DO) GO TO 80
      IF(DABS(R(K,K)) .LT. DABS(SDIAG(K))) THEN
        COTAN = R(K,K)/SDIAG(K)
        SIN = 0.5DO/DSQRT(0.25DO+0.25DO*COTAN**2)
        COS = SIN*COTAN
      ELSE
        TAN = SDIAG(K)/R(K,K)
        COS = 0.5DO/DSQRT(0.25DO+0.25DO*TAN**2)
        SIN = COS*TAN
      ENDIF

C ----- COMPUTE THE MODIFIED DIAGONAL ELEMENT OF R AND
C ----- THE MODIFIED ELEMENT OF ((Q TRANSPOSE)*B,O).
      R(K,K) = COS*R(K,K) + SIN*SDIAG(K)
      TEMP = COS*WA(K) + SIN*QTBPJ
      QTBPJ = -SIN*WA(K) + COS*QTBPJ
      WA(K) = TEMP

C ----- ACCUMULATE THE TRANSFORMATION IN THE ROW OF S.
      DO 60 I = K+1, N
        TEMP = COS*R(I,K) + SIN*SDIAG(I)
        SDIAG(I) = -SIN*R(I,K) + COS*SDIAG(I)
        R(I,K) = TEMP
60      CONTINUE
80      CONTINUE
90      CONTINUE

C --- STORE THE DIAGONAL ELEMENT OF S AND RESTORE
C --- THE CORRESPONDING DIAGONAL ELEMENT OF R.
      SDIAG(J) = R(J,J)
      R(J,J) = P(J)
100     CONTINUE

C --- SOLVE THE TRIANGULAR SYSTEM FOR Z. IF THE SYSTEM IS
C --- SINGULAR, THEN OBTAIN A LEAST SQUARES SOLUTION.
      NSING = N
      DO 110 J = 1, N
        IF(SDIAG(J) .EQ. 0.0DO .AND. NSING .EQ. N) NSING = J - 1
        IF(NSING .LT. N) WA(J) = 0.0DO
110     CONTINUE
      DO 130 K=1,NSING
        J=NSING-K+1
        SUM=0.0DO
        DO 120 I=J+1,NSING
          SUM=SUM+R(I,J)*WA(I)
120     CONTINUE
        WA(J)=(WA(J)-SUM)/SDIAG(J)
130     CONTINUE

      CALL VPVT(N,P,WA,IPVT)
C      ... PERMUTE THE COMPONENTS OF Z BACK TO COMPONENTS OF P
      END

      SUBROUTINE QRFAC(M,N,A,IPVT,RDIAG,ACNORM)
C*****
C
C * THIS IS A MODIFIED VERSION OF THE MINPACK ROUTINE QRFAC
C
C * THAT USES HOUSEHOLDER TRANSFORMATIONS WITH COLUMN PIVOTING
C (OPTIONAL) TO COMPUTE A QR FACTORIZATION OF THE M BY N MATRIX A.
C THAT IS, QRFAC DETERMINES AN ORTHOGONAL MATRIX Q, A PERMUTATION
C MATRIX P, AND AN UPPER TRAPEZOIDAL MATRIX R WITH DIAGONAL
C ELEMENTS OF NONINCREASING MAGNITUDE, SUCH THAT A*P = Q*R. THE
C HOUSEHOLDER TRANSFORMATION FOR COLUMN K, K = 1,2,...,MIN(M,N),
C IS OF THE FORM

```

```

C          T
C          I - (1/U(K))*U*U
C
C          WHERE U HAS ZEROS IN THE FIRST K-1 POSITIONS.
C
C          A -- AN M BY N ARRAY. ON INPUT A CONTAINS THE MATRIX FOR
C          WHICH THE QR FACTORIZATION IS TO BE COMPUTED. ON OUTPUT
C          THE STRICT UPPER TRAPEZOIDAL PART OF A CONTAINS THE STRICT
C          UPPER TRAPEZOIDAL PART OF R, AND THE LOWER TRAPEZOIDAL
C          PART OF A CONTAINS A FACTORED FORM OF Q (THE NON-TRIVIAL
C          ELEMENTS OF THE U VECTORS DESCRIBED ABOVE).
C
C          IPVT -- AN INTEGER OUTPUT ARRAY WHICH DEFINES THE PERMUTATION
C          MATRIX P SUCH THAT A*P = Q*R. COLUMN J OF P IS COLUMN
C          IPVT(J) OF THE IDENTITY MATRIX.
C
C          RDIAG -- AN OUTPUT ARRAY WHICH CONTAINS THE DIAGONAL ELEMENTS OF R.
C
C          ACNORM -- AN OUTPUT ARRAY OF LENGTH N WHICH CONTAINS THE NORMS OF
C          THE CORRESPONDING COLUMNS OF THE INPUT MATRIX A.
C
C          SUBPROGRAMS CALLED ... ENORM
C
C*****
C          INTEGER M,N, IPVT(N)
C          REAL*8 A(M,N),RDIAG(N),ACNORM(N)
C          INTEGER I,J,K,KMAX
C          PARAMETER(NN=40)
C          REAL*8 AJNORM,EPSMCH,DWARF,SUM,TEMP,WA(NN),ENORM
C          COMMON /MCHEPS/ EPSMCH,DWARF
C          ... EPSMCH IS THE MACHINE PRECISION
C
C --- COMPUTE THE INITIAL COLUMN NORMS AND INITIALIZE SEVERAL ARRAYS.
C          DO 10 J = 1, N
C              ACNORM(J) = ENORM(M,A(1,J))
C              RDIAG(J) = ACNORM(J)
C              WA(J) = RDIAG(J)
C              IPVT(J) = J
10          CONTINUE
C
C --- REDUCE A TO R WITH HOUSEHOLDER TRANSFORMATIONS.
C          DO 110 J = 1, MINO(M,N)
C
C ----- BRING THE COLUMN OF LARGEST NORM INTO THE PIVOT POSITION.
C          KMAX = J
C          DO 20 K = J, N
C              IF (RDIAG(K) .GT. RDIAG(KMAX)) KMAX = K
20          CONTINUE
C          IF(KMAX.NE.J) THEN
C              DO 30 I = 1, M
C                  TEMP = A(I,J)
C                  A(I,J) = A(I,KMAX)
C                  A(I,KMAX) = TEMP
30          CONTINUE
C          RDIAG(KMAX) = RDIAG(J)
C          WA(KMAX) = WA(J)
C          K = IPVT(J)
C          IPVT(J) = IPVT(KMAX)
C          IPVT(KMAX) = K
C          ENDIF
C
C ----- COMPUTE THE HOUSEHOLDER TRANSFORMATION TO REDUCE THE
C ----- J-TH COLUMN OF A TO A MULTIPLE OF THE J-TH UNIT VECTOR.
C          AJNORM = ENORM(M-J+1,A(J,J))
C          IF(AJNORM .EQ. 0.0D0) GOTO 110
C          IF(A(J,J) .LT. 0.0D0) AJNORM = -AJNORM
C          DO 50 I = J, M
C              A(I,J) = A(I,J)/AJNORM

```



```

50     CONTINUE
      A(J,J) = A(J,J) + 1.0DO

C ----- APPLY TRANSFORMATION TO REMAINING COLUMNS & UPDATE THE NORMS
      DO 90 K = J+1, N
        SUM = 0.0DO
        DO 60 I = J, M
          SUM = SUM + A(I,J)*A(I,K)
60     CONTINUE
        TEMP = SUM/A(J,J)
        DO 70 I = J, M
          A(I,K) = A(I,K) - TEMP*A(I,J)
70     CONTINUE
        IF(RDIAG(K) .NE. 0.0DO) THEN
          TEMP = A(J,K)/RDIAG(K)
          RDIAG(K) = RDIAG(K)*DSQRT(DMAX1(0.0DO,1.0DO-TEMP**2))
          IF(0.05DO*(RDIAG(K)/WA(K))**2 .LE. EPSMCH) THEN
            RDIAG(K) = ENORM(M-J,A(J+1,K))
            WA(K) = RDIAG(K)
          ENDIF
        ENDIF
      ENDIF
90     CONTINUE
      RDIAG(J) = -AJNORM
110    CONTINUE
      END

```

```

      SUBROUTINE VCOPY(N,A,B)
C*****
C
C   VECTOR COPY: A=B
C
C*****
      REAL*8 A(N),B(N)
      DO 10 I=1,N
        A(I)=B(I)
10     CONTINUE
      END

```

```

      SUBROUTINE VNCOPY(N,A,B)
C*****
C
C   VECTOR COPY: A=-B
C
C*****
      REAL*8 A(N),B(N)
      DO 10 I=1,N
        A(I)=-B(I)
10     CONTINUE
      END

```

```

      SUBROUTINE MCOPY(M,N,A,B)
C*****
C
C   MATRIX COPY: A=B
C
C*****
      REAL*8 A(M,N),B(M,N)
      DO 10 J=1,N
        DO 10 I=1,M
          A(I,J)=B(I,J)
10     CONTINUE
      END

```

```

      SUBROUTINE VADD(N,A,B,C)

```

```

C*****
C
C   VECTOR ADDITION: C=A+B
C
C*****
      REAL*8 A(N),B(N),C(N)
      DO 10 I=1,N
        C(I)=A(I)+B(I)
10    CONTINUE
      END

      SUBROUTINE VMUL(N,A,B,C)
C*****
C
C   VECTOR MULTIPLICATION(COMPONENT-BY-COMPONENT): C =A *B
C
C
C
C*****
      REAL*8 A(N),B(N),C(N)
      DO 10 I=1,N
        C(I)=A(I)*B(I)
10    CONTINUE
      END

      SUBROUTINE VPVT(N,A,B,IPVT)
C*****
C
C   VECTOR PIVOT: A=IPVT*B
C
C
C*****
      REAL*8 A(N),B(N)
      INTEGER IPVT(N)
      DO 10 J=1,N
        L=IPVT(J)
        A(L)=B(J)
10    CONTINUE
      END

      SUBROUTINE MVMUL(M,N,A,B,C)
C*****
C
C   MATRIX TIMES VECTOR: C=A*B
C
C
C*****
      REAL*8 A(M,N),B(N),C(M)
      DO 10 I=1,M
        C(I)=0.000
        DO 10 J=1,N
          C(I)=C(I)+A(I,J)*B(J)
10    CONTINUE
      END

      SUBROUTINE VMAX(N,A,B,C)
C*****
C
C   MAX OF 2 VECTORS: C=MAX(A,B)
C
C
C*****
      REAL*8 A(N),B(N),C(N)
      DO 10 I=1,N
        C(I)=DMAX1(A(I),B(I))
10    CONTINUE
      END

```

```

      SUBROUTINE VSCALE(N,SCALE,A,B)
C*****
C
C   VECTOR*SCALE: B=SCALE*A
C
C*****
      REAL*8 SCALE,A(N),B(N)
      DO 10 I=1,N
         B(I)=SCALE*A(I)
10    CONTINUE
      END

```

```

      SUBROUTINE VSET(N,A,SCALE)
C*****
C
C   VECTOR SET: A =SCALE
C           I
C
C*****
      REAL*8 A(N),SCALE
      DO 10 I=1,N
         A(I)=SCALE
10    CONTINUE
      END

```

```

      SUBROUTINE MSET(M,N,A,SCALE)
C*****
C
C   MATRIX SET: A =SCALE
C           IJ
C
C*****
      REAL*8 A(M,N),SCALE
      DO 20 J=1,N
         DO 10 I=1,M
            A(I,J)=SCALE
10        CONTINUE
20    CONTINUE
      END

```

```

      REAL*8 FUNCTION INNER(N,A,B)
C*****
C
C   VECTOR INNER PRODUCT: INNER = A*B
C
C*****
      REAL*8 A(N),B(N)
      INNER=0.000
      DO 10 I=1,N
         INNER=INNER+A(I)*B(I)
10    CONTINUE
      END

```

```

      REAL*8 FUNCTION VSUM(N,X)
C*****
C
C   VECTOR ELEMENTS SUMMATION: VSUM = X(1)+...+X(N)
C
C*****
      REAL*8 X(N)
      VSUM = 0.000
      DO 10 J = 1, N
         VSUM = VSUM + X(J)

```

```
10 CONTINUE
END
```

```
      REAL*8 FUNCTION ENORM(N,X)
C*****
C
C * THIS IS A MODIFIED VERSION OF THE MINPACK ROUTINE ENORM
C   THAT CALCULATES THE EUCLIDEAN NORM OF AN N-VECTOR X
C
C * THE EUCLIDEAN NORM IS COMPUTED BY ACCUMULATING THE SUM OF
C   SQUARES IN THREE DIFFERENT SUMS. THE SUMS OF SQUARES FOR THE
C   SMALL AND LARGE COMPONENTS ARE SCALED SO THAT NO OVERFLOWS
C   OCCUR. NON-DESTRUCTIVE UNDERFLOWS ARE PERMITTED. UNDERFLOWS
C   AND OVERFLOWS DO NOT OCCUR IN THE COMPUTATION OF THE UNSCALED
C   SUM OF SQUARES FOR THE INTERMEDIATE COMPONENTS.
C
C * THE DEFINITIONS OF SMALL, INTERMEDIATE AND LARGE COMPONENTS
C   DEPEND ON TWO CONSTANTS, RDWARF AND RGIANT. THE MAIN RESTRICTIONS
C   ON THESE CONSTANTS ARE THAT RDWARF**2 NOT UNDERFLOW AND RGIANT**2
C   NOT OVERFLOW. THE CONSTANTS GIVEN HERE ARE SUITABLE FOR EVERY
C   KNOWN COMPUTER.
C
C*****
      REAL*8 X(N),
      &      AGIANT,FLOATN,RDWARF,RGIANT,S1,S2,S3,XABS,X1MAX,X3MAX
      DATA RDWARF,RGIANT/3.834D-20,1.304D19/

      S1 = 0.0D0
      S2 = 0.0D0
      S3 = 0.0D0
      X1MAX = 0.0D0
      X3MAX = 0.0D0
      FLOATN = N
      AGIANT = RGIANT/FLOATN
      DO 90 I = 1, N
        XABS = DABS(X(I))
        IF (XABS.GE.AGIANT) THEN
C ----- SUM FOR LARGE COMPONENTS ...
          IF(XABS .GT. X1MAX) THEN
            S1 = 1.0D0 + S1*(X1MAX/XABS)**2
            X1MAX = XABS
          ELSE
            S1 = S1 + (XABS/X1MAX)**2
          ENDIF
          ELSE IF(XABS.LE.RDWARF) THEN
C ----- SUM FOR SMALL COMPONENTS ...
            IF(XABS .GT. X3MAX) THEN
              S3 = 1.0D0 + S3*(X3MAX/XABS)**2
              X3MAX = XABS
            ELSE IF (XABS .NE. 0.0D0) THEN
              S3 = S3 + (XABS/X3MAX)**2
            ENDIF
          ELSE
C ----- SUM FOR INTERMEDIATE COMPONENTS ...
            S2 = S2 + XABS**2
          ENDIF
        END DO
      90 CONTINUE

C --- CALCULATION OF NORM.
      IF (S1.NE.0.0D0) THEN
        ENORM = X1MAX*DSQRT(S1+(S2/X1MAX)/X1MAX)
      ELSE IF (S2.NE.0.0D0) THEN
        IF (S2 .GE. X3MAX) THEN
          ENORM=DSQRT(S2*(1.0D0+(X3MAX/S2)*(X3MAX*S3)))
        ELSE
          ENORM=DSQRT(X3MAX*((S2/X3MAX)+(X3MAX*S3)))
        ENDIF
      ENDIF
```

```

ELSE
  ENORM=X3MAX*DSQRT(S3)
ENDIF
END

SUBROUTINE NQTF(M,N,FVEC,FJAC,QTF,RDIAG)
C*****
C
C          T
C  COMPUTE Q * F AND G
C
C*****
  REAL*8 FVEC(M),FJAC(M,N),QTF(N),RDIAG(N)

  PARAMETER(MM=65)
  INTEGER I,J
  REAL*8 SUM,TEMP,W(MM)

C --- COMPUTE Q^T*F ...
  CALL VCOPY(M,W,FVEC)
  DO 30 J = 1, N
    IF(FJAC(J,J).NE.0.0DO) THEN
      SUM = 0.0DO
      DO 10 I = J, M
        SUM = SUM + FJAC(I,J)*W(I)
10      CONTINUE
      TEMP = -SUM/FJAC(J,J)
      DO 20 I = J, M
        W(I) = W(I) + FJAC(I,J)*TEMP
20      CONTINUE
      ENDIF
      FJAC(J,J) = RDIAG(J)
      QTF(J) = W(J)
30    CONTINUE
  END

SUBROUTINE NOISEY(M,Y)
C*****
C
C  MAKE NOISE ON DATA POINTS
C
C*****
  REAL*8 Y(M),YNOISE
  SAVE ISWHO
  COMMON /NOISE/YNOISE /REFNUM/NPROB,NFEV,NJEV,ITER

  IF(YNOISE.EQ.0.0DO .OR. ISWHO.EQ.NPROB) RETURN
  ISWHO=NPROB
  DO 10 I=1,M
    Y(I)=Y(I)+YNOISE*(2*MOD(I,2)-1)
10  CONTINUE
  END

SUBROUTINE INIT(M,N,X,K)
C*****
C
C  GET AN INITIAL POINT FOR A GIVEN TEST PROBLEM
C
C*****
  REAL*8 X(N)
  COMMON /REFNUM/NPROB,NFEV,NJEV,ITER

C --- INITIALIZE COUNTERS ...
  NFEV=0
  NJEV=0

```

```

ITER=0

GOTO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18)NPROB

1  CALL INIT1(N,X)
   GOTO 99
2  CALL INIT2(N,X)
   GOTO 99
3  CALL INIT3(N,X)
   GOTO 99
4  CALL INIT4(N,X)
   GOTO 99
5  CALL INIT5(N,X)
   GOTO 99
6  CALL INIT6(N,X)
   GOTO 99
7  CALL INIT7(N,X)
   GOTO 99
8  CALL INIT8(N,X)
   GOTO 99
9  CALL INIT9(N,X)
   GOTO 99
10 CALL INIT10(N,X)
   GOTO 99
11 CALL INIT11(N,X)
   GOTO 99
12 CALL INIT12(N,X)
   GOTO 99
13 CALL INIT13(N,X)
   GOTO 99
14 CALL INIT14(N,X)
   GOTO 99
15 CALL INIT15(N,X)
   GOTO 99
16 CALL INIT16(N,X)
   GOTO 99
17 CALL INIT17(N,X)
   GOTO 99
18 CALL INIT18(N,X)

C --- COMPUTE MULTIPLE OF INITIAL POINT.
99 IF(K.GT.1) THEN
   IF(NPROB.EQ.11) THEN
     CALL VSET(N,X,1.0D1**(K-1))
     ... XO IS A ZERO VECTOR FOR PROBLEM 11
   ELSE
     CALL VSCALE(N,1.0D1**(K-1),X,X)
     ... INITIAL PT IS 10(K-1)*XO ON THE K-TH TRY
   ENDIF
ENDIF
CALL PRINT(M,N,X)
END

SUBROUTINE FCN(M,N,X,FVEC,FNORM)
C*****
C
C  EVALUATE THE OBJECTIVE AT A GIVEN POINT
C
C*****
REAL*8 X(N),FVEC(M),FNORM,ENORM
COMMON /REFNUM/NPROB,NFEV,NJEV,ITER /STEPSZ/ISLONG

GOTO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18)NPROB

1  CALL FCN1(M,N,X,FVEC)
   GOTO 99
2  CALL FCN2(M,N,X,FVEC)

```

```

      GOTO 99
3    CALL FCN3(M,N,X,FVEC)
      GOTO 99
4    CALL FCN4(M,N,X,FVEC)
      GOTO 99
5    CALL FCN5(M,N,X,FVEC)
      GOTO 99
6    CALL FCN6(M,N,X,FVEC)
      GOTO 99
7    CALL FCN7(M,N,X,FVEC)
      GOTO 99
8    CALL FCN8(M,N,X,FVEC)
      GOTO 99
9    CALL FCN9(M,N,X,FVEC)
      GOTO 99
10   CALL FCN10(M,N,X,FVEC)
      GOTO 99
11   CALL FCN11(M,N,X,FVEC)
      GOTO 99
12   CALL FCN12(M,N,X,FVEC)
      GOTO 99
13   CALL FCN13(M,N,X,FVEC)
      GOTO 99
14   CALL FCN14(M,N,X,FVEC)
      GOTO 99
15   CALL FCN15(M,N,X,FVEC)
      GOTO 99
16   CALL FCN16(M,N,X,FVEC)
      GOTO 99
17   CALL FCN17(M,N,X,FVEC)
      GOTO 99
18   CALL FCN18(M,N,X,FVEC)

99   IF(ISLONG.EQ.1) RETURN
      FNORM=ENORM(M,FVEC)
      NFEV=NFEV+1
      END

```

```

SUBROUTINE JAC(M,N,X,FJAC)

```

```

C*****
C
C   EVALUATE THE JACOBIAN MATRIX AT A GIVEN POINT
C
C*****
      REAL*8 X(N),FJAC(M,N)
      COMMON /REFNUM/NPROB,NFEV,NJEV,ITER

      GOTO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18)NPROB

1    CALL JAC1(M,N,X,FJAC)
      GOTO 99
2    CALL JAC2(M,N,X,FJAC)
      GOTO 99
3    CALL JAC3(M,N,X,FJAC)
      GOTO 99
4    CALL JAC4(M,N,X,FJAC)
      GOTO 99
5    CALL JAC5(M,N,X,FJAC)
      GOTO 99
6    CALL JAC6(M,N,X,FJAC)
      GOTO 99
7    CALL JAC7(M,N,X,FJAC)
      GOTO 99
8    CALL JAC8(M,N,X,FJAC)
      GOTO 99
9    CALL JAC9(M,N,X,FJAC)
      GOTO 99

```

```

10 CALL JAC10(M,N,X,FJAC)
   GOTO 99
11 CALL JAC11(M,N,X,FJAC)
   GOTO 99
12 CALL JAC12(M,N,X,FJAC)
   GOTO 99
13 CALL JAC13(M,N,X,FJAC)
   GOTO 99
14 CALL JAC14(M,N,X,FJAC)
   GOTO 99
15 CALL JAC15(M,N,X,FJAC)
   GOTO 99
16 CALL JAC16(M,N,X,FJAC)
   GOTO 99
17 CALL JAC17(M,N,X,FJAC)
   GOTO 99
18 CALL JAC18(M,N,X,FJAC)

99  NJEV=NJEV+1
   END

```

```

C*****
C
C  PROBLEM LIBRARY
C  -----
C      IN THIS LIBRARY THERE ARE CURRENTLY 18 NONLINEAR LEAST SQUARES
C  TEST PROBLEMS, EACH OF WHICH CONTAINS THREE ENTRIES. THE FIRST ONE
C  IS FOR INITIAL POINT, THE SECOND ONE IS FOR FUNCTION EVALUATION
C  AND THE LAST ONE FOR JACOBIAN MATRIX EVALUATION.
C
C  MAJOR REFERENCE
C  -----
C  MORE, J. J., GARBOW, B. S. AND HILLSTROM, K.E, TESTING
C  UNCONSTRAINED OPTIMIZATION SOFTWARE, ACM TRANS. ON MATH. SOFTWARE,
C  VOL.7, NO.1, 1981, PP.17-41.
C
C*****

```

```

      SUBROUTINE PROB1(M,N,X,FVEC,FJAC)
C*****
C
C  PROBLEM 1:  LINEAR FUNCTION - FULL RANK
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N),SUM,TEMP,VSUM

      ENTRY INIT1(N,X)
        CALL VSET(N,X,1.0D0)
        RETURN

      ENTRY FCN1(M,N,X,FVEC)
        SUM=VSUM(N,X)
        TEMP=2.0D0*SUM/M+1.0D0
        DO 10 I=1,M
          FVEC(I)=-TEMP
          IF(I.LE.N)FVEC(I)=FVEC(I)+X(I)
10      CONTINUE
        RETURN

      ENTRY JAC1(M,N,X,FJAC)
        TEMP=2.0D0/M
        DO 30 J=1,N
          DO 20 I=1,M
            FJAC(I,J)=-TEMP
20      CONTINUE
          FJAC(J,J)=FJAC(J,J)+1.0D0
30      CONTINUE

```



```

RETURN
END

```

```

SUBROUTINE PROB2(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 2:  LINEAR FUNCTION - RANK 1
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),TEMP

ENTRY INIT2(N,X)
  CALL VSET(N,X,1.0D0)
  RETURN

ENTRY FCN2(M,N,X,FVEC)
  TEMP=0.0D0
  DO 10 J=1,N
    TEMP=TEMP+J*X(J)
10  CONTINUE
  DO 20 I=1,M
    FVEC(I)=I*TEMP-1.0D0
20  CONTINUE
  RETURN

ENTRY JAC2(M,N,X,FJAC)
  DO 40 J=1,N
    DO 30 I=1,M
      FJAC(I,J)=I*J
30  CONTINUE
40  CONTINUE
  RETURN
END

```

```

SUBROUTINE PROB3(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 3:  LINEAR FUNCTION - RANK 1 WITH ZERO COLUMNS AND ROWS
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),SUM

ENTRY INIT3(N,X)
  CALL VSET(N,X,1.0D0)
  RETURN

ENTRY FCN3(M,N,X,FVEC)
  SUM = 0.0D0
  DO 10 J=2,N-1
    SUM=SUM+J*X(J)
10  CONTINUE
  DO 20 I=1,M
    FVEC(I)=(I-1)*SUM-1.0D0
20  CONTINUE
  FVEC(M)=-1.0D0
  RETURN

ENTRY JAC3(M,N,X,FJAC)
  CALL MSET(M,N,FJAC,0.0D0)
  DO 40 J=2,N-1
    DO 30 I=2,M-1
      FJAC(I,J)=I*J-J
30  CONTINUE
40  CONTINUE
  RETURN
END

```

```

      SUBROUTINE PROB4(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 4:  ROSENBROCK FUNCTION
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N)

      ENTRY INIT4(N,X)
        X(1) = -1.2D0
        X(2) = 1.0D0
      RETURN

      ENTRY FCN4(M,N,X,FVEC)
        FVEC(1) = 1.0D1*(X(2)-X(1)**2)
        FVEC(2) = 1.0D0-X(1)
      RETURN

      ENTRY JAC4(M,N,X,FJAC)
        FJAC(1,1) = -2.0D1*X(1)
        FJAC(1,2) = 1.0D1
        FJAC(2,1) = -1.0D0
        FJAC(2,2) = 0.0D0
      RETURN
      END

      SUBROUTINE PROB5(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 5:  HELICAL VALLEY FUNCTION
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N),TMP1,TMP2,PI
      SAVE PI,TMP2

      ENTRY INIT5(N,X)
        X(1) = -1.0D0
        X(2) = 0.0D0
        X(3) = 0.0D0
        PI = 8.0D0*DATAN(1.0D0)
      RETURN

      ENTRY FCN5(M,N,X,FVEC)
        IF(X(1).EQ.0.0D0) THEN
          TMP1 = DSIGN(2.5D-1,X(2))
        ELSE IF (X(1) .GT. 0.0D0) THEN
          TMP1 = DATAN(X(2)/X(1))/PI
        ELSE
          TMP1 = DATAN(X(2)/X(1))/PI + 0.5D0
        ENDIF
        TMP2 = DSQRT(X(1)**2+X(2)**2)
        FVEC(1) = 1.0D1*(X(3) - 1.0D1*TMP1)
        FVEC(2) = 1.0D1*(TMP2 - 1.0D0)
        FVEC(3) = X(3)
      RETURN

      ENTRY JAC5(M,N,X,FJAC)
        TMP1 = PI*(X(1)**2+X(2)**2)
        FJAC(1,1) = 1.0D2*X(2)/TMP1
        FJAC(1,2) = -1.0D2*X(1)/TMP1
        FJAC(1,3) = 1.0D1
        FJAC(2,1) = 1.0D1*X(1)/TMP2
        FJAC(2,2) = 1.0D1*X(2)/TMP2
        FJAC(2,3) = 0.0D0
        FJAC(3,1) = 0.0D0

```

```

      FJAC(3,2) = 0.0D0
      FJAC(3,3) = 1.0D0
      RETURN
      END

```

```

      SUBROUTINE PROB6(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 6: POWELL SINGULAR FUNCTION
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N)

      ENTRY INIT6(N,X)
        X(1)=3.0D0
        X(2)=-1.0D0
        X(3)=0.0D0
        X(4)=1.0D0
      RETURN

      ENTRY FCN6(M,N,X,FVEC)
        FVEC(1)=X(1)+1.0D1*X(2)
        FVEC(2)=DSQRT(5.0D0)*(X(3)-X(4))
        FVEC(3)=(X(2)-2.0D0*X(3))**2
        FVEC(4)=DSQRT(1.0D1)*(X(1)-X(4))**2
      RETURN

      ENTRY JAC6(M,N,X,FJAC)
        CALL MSET(M,N,FJAC,0.0D0)
        FJAC(1,1)=1.0D0
        FJAC(1,2)=1.0D1
        FJAC(2,3)=DSQRT(5.0D0)
        FJAC(2,4)=-FJAC(2,3)
        FJAC(3,2)=2.0D0*(X(2)-2.0D0*X(3))
        FJAC(3,3)=-2.0D0*FJAC(3,2)
        FJAC(4,1)=2.0D0*DSQRT(1.0D1)*(X(1)-X(4))
        FJAC(4,4)=-FJAC(4,1)
      RETURN
      END

```

```

      SUBROUTINE PROB7(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 7: FREUDENSTEIN AND ROTH FUNCTION
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N)

      ENTRY INIT7(N,X)
        X(1)=0.5D0
        X(2)=-2.0D0
      RETURN

      ENTRY FCN7(M,N,X,FVEC)
        FVEC(1)=-1.3D1+X(1)+((5.0D0-X(2))*X(2)-2.0D0)*X(2)
        FVEC(2)=-2.9D1+X(1)+((1.0D0+X(2))*X(2)-1.4D1)*X(2)
      RETURN

      ENTRY JAC7(M,N,X,FJAC)
        FJAC(1,1) = 1.0D0
        FJAC(1,2) = X(2)*(1.0D1 - 3.0D0*X(2)) - 2.0D0
        FJAC(2,1) = 1.0D0
        FJAC(2,2) = X(2)*(2.0D0 + 3.0D0*X(2)) - 1.4D1
      RETURN
      END

```

```

      SUBROUTINE PROB8(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 8:  BARD FUNCTION
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N),Y(15),TMP1,TMP2,TMP3,TMP4
      SAVE Y
      DATA Y/1.4D-1,1.8D-1,2.2D-1,2.5D-1,2.9D-1,3.2D-1,3.5D-1,3.9D-1,
      *      3.7D-1,5.8D-1,7.3D-1,9.6D-1,1.34D0,2.1D0,4.39D0/

      ENTRY INIT8(N,X)
      CALL VSET(N,X,1.0D0)

C ----- MAKE NOISE ON DATA POINTS ...
      CALL NOISEY(15,Y)
      RETURN

      ENTRY FCN8(M,N,X,FVEC)
      DO 10 I = 1, M
          TMP1=I
          TMP2=16-I
          TMP3=TMP1
          IF(I.GT.8) TMP3=TMP2
          FVEC(I)=Y(I)-(X(1)+TMP1/(X(2)*TMP2+X(3)*TMP3))
10      CONTINUE
      RETURN

      ENTRY JAC8(M,N,X,FJAC)
      DO 20 I=1,M
          TMP1=I
          TMP2=16-I
          TMP3=TMP1
          IF(I.GT.8) TMP3=TMP2
          TMP4=(X(2)*TMP2+X(3)*TMP3)**2
          FJAC(I,1)=-1.0D0
          FJAC(I,2)=TMP1*TMP2/TMP4
          FJAC(I,3)=TMP1*TMP3/TMP4
20      CONTINUE
      RETURN
      END

      SUBROUTINE PROB9(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 9:  KOWALIK AND OSBORNE FUNCTION
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N),V(11),Y(11),TMP1(11),TMP2(11)
      SAVE TMP1, TMP2, Y
      DATA V/4.0D0,2.0D0,1.0D0,5.0D-1,2.5D-1,1.67D-1,1.25D-1,
      &      1.0D-1,8.33D-2,7.14D-2,6.25D-2/
      &      Y/1.957D-1,1.947D-1,1.735D-1,1.6D-1,8.44D-2,6.27D-2,
      &      4.56D-2, 3.42D-2,3.23D-2,2.35D-2,2.46D-2/

      ENTRY INIT9(N,X)
          X(1) = 2.5D-1
          X(2) = 3.9D-1
          X(3) = 4.15D-1
          X(4) = 3.9D-1

C ----- MAKE NOISE ON DATA POINTS ...
      CALL NOISEY(11,Y)
      RETURN

      ENTRY FCN9(M,N,X,FVEC)

```

```

      DO 10 I = 1, 11
        TMP1(I)=V(I)*(V(I)+X(2))
        TMP2(I)=V(I)*(V(I)+X(3))+X(4)
        FVEC(I)=Y(I)-X(1)*TMP1(I)/TMP2(I)
10     CONTINUE
      RETURN

      ENTRY JAC9(M,N,X,FJAC)
      DO 20 I=1,11
        FJAC(I,1)=-TMP1(I)/TMP2(I)
        FJAC(I,2)=-V(I)*X(1)/TMP2(I)
        FJAC(I,3)=FJAC(I,1)*FJAC(I,2)
        FJAC(I,4)=FJAC(I,3)/V(I)
20     CONTINUE
      RETURN
      END

      SUBROUTINE PROB10(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 10: MEYER FUNCTION(M=16, N=3)
C
C*****
      REAL*8 X(N),FVEC(M),FJAC(M,N),Y(16),TMP1(16),TMP2(16)
      SAVE TMP1,TMP2,Y
      COMMON /STEPSZ/ISLONG /PAR/IPAR(5)
      DATA Y/3.478D4,2.861D4,2.365D4,1.963D4,1.637D4,1.372D4,1.154D4,
&          9.744D3,8.261D3,7.03D3, 6.005D3,5.147D3,4.427D3, 3.82D3,
&          3.307D3,2.872D3/

      ENTRY INIT10(N,X)
        X(1) = 2.0D-2
        X(2) = 4.0D3
        X(3) = 2.5D2

C ----- MAKE NOISE ON DATA POINTS ...
      CALL NOISEY(16,Y)
      RETURN

      ENTRY FCN10(M,N,X,FVEC)
        ISLONG=1
        DO 10 I = 1, 16
          TMP1(I)=5.0D0*I+4.5D1+X(3)
          IF(IPAR(5).EQ.1.AND.X(2)/TMP1(I).GT.30) RETURN
          TMP2(I)=DEXP(X(2)/TMP1(I))
          FVEC(I)=X(1)*TMP2(I)-Y(I)
10     CONTINUE
        ISLONG=0
      RETURN

      ENTRY JAC10(M,N,X,FJAC)
        DO 20 I = 1, 16
          TMP1(I)=5.0D0*I+4.5D1+X(3)
          FJAC(I,1)=TMP2(I)
          FJAC(I,2)=X(1)*TMP2(I)/TMP1(I)
          FJAC(I,3)=-X(2)/TMP1(I)*FJAC(I,2)
20     CONTINUE
      RETURN
      END

      SUBROUTINE PROB11(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 11: WATSON FUNCTION
C
C*****

```

```

REAL*8 X(N),FVEC(M),FJAC(M,N),DX,DIV,S1,S2,TEMP

ENTRY INIT11(N,X)
  CALL VSET(N,X,0.0D0)
  RETURN

ENTRY FCN11(M,N,X,FVEC)
  DO 30 I=1,29
    DIV=I/2.9D1
    S1=0.0D0
    DX=1.0D0
    DO 10 J=2, N
      S1=S1+(J-1)*DX*X(J)
      DX=DIV*DX
10    CONTINUE
    S2 = 0.0D0
    DX = 1.0D0
    DO 20 J = 1, N
      S2 = S2 + DX*X(J)
      DX = DIV*DX
20    CONTINUE
    FVEC(I) = S1 - S2**2 - 1.0D0
30    CONTINUE
    FVEC(30) = X(1)
    FVEC(31) = X(2) - X(1)**2 - 1.0D0
  RETURN

ENTRY JAC11(M,N,X,FJAC)
  DO 60 I=1,29
    DIV=I/2.9D1
    S2=0.0D0
    DX=1.0D0
    DO 40 J = 1, N
      S2 = S2 + DX*X(J)
      DX = DIV*DX
40    CONTINUE
    TEMP=2.0D0*DIV*S2
    DX=1.0D0/DIV
    DO 50 J=1,N
      FJAC(I,J)=DX*(J-1-TEMP)
      DX=DIV*DX
50    CONTINUE
60    CONTINUE
    DO 70 J = 3, N
      FJAC(30,J) = 0.0D0
      FJAC(31,J) = 0.0D0
70    CONTINUE
    FJAC(30,1) = 1.0D0
    FJAC(31,1) = -2.0D0*X(1)
    FJAC(30,2) = 0.0D0
    FJAC(31,2) = 1.0D0
  RETURN
END

```

```

SUBROUTINE PROB12(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 12: BOX 3-DIMENSIONAL FUNCTION
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),TEMP,TMP1

ENTRY INIT12(N,X)
  X(1) = 0.0D0
  X(2) = 1.0D1
  X(3) = 2.0D1
  RETURN

```

```

ENTRY FCN12(M,N,X,FVEC)
  DO 10 I = 1, M
    TEMP=I
    TMP1=TEMP/1.0D1
    FVEC(I)=DEXP(-TMP1*X(1))-DEXP(-TMP1*X(2))
&      +(DEXP(-TEMP)-DEXP(-TMP1))*X(3)
10  CONTINUE
RETURN

ENTRY JAC12(M,N,X,FJAC)
  DO 20 I = 1, M
    TEMP=I
    TMP1=TEMP/1.0D1
    FJAC(I,1) = -TMP1*DEXP(-TMP1*X(1))
    FJAC(I,2) = TMP1*DEXP(-TMP1*X(2))
    FJAC(I,3) = DEXP(-TEMP) - DEXP(-TMP1)
20  CONTINUE
RETURN
END

SUBROUTINE PROB13(M,N,X,FVEC,FJAC)
C*****
C
C  PROBLEM 13: JENNRICH AND SAMPSON FUNCTION
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),TEMP
COMMON /STEPSZ/ISLONG /PAR/IPAR(5)

ENTRY INIT13(N,X)
  X(1) = 0.3D0
  X(2) = 0.4D0
RETURN

ENTRY FCN13(M,N,X,FVEC)
C ----- TEST FOR POSSIBLE OVERFLOW ...
  ISLONG=1
  IF(IPAR(5).EQ.1.AND.(X(1).GT.3.0D0 .OR. X(2).GT.3.0D0))
&    RETURN
  ISLONG=0

  DO 10 I = 1, M
    TEMP=I
    FVEC(I)=2.0D0+2.0D0*TEMP-DEXP(TEMP*X(1))-DEXP(TEMP*X(2))
10  CONTINUE
RETURN

ENTRY JAC13(M,N,X,FJAC)
  DO 20 I=1,M
    TEMP=I
    FJAC(I,1)=-TEMP*DEXP(TEMP*X(1))
    FJAC(I,2)=-TEMP*DEXP(TEMP*X(2))
20  CONTINUE
RETURN
END

SUBROUTINE PROB14(M,N,X,FVEC,FJAC)
C*****
C
C  PROBLEM 14: BROWN AND DENNIS FUNCTION
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),TMP1(20),TMP2(20),TEMP,TI
SAVE TMP1,TMP2

```

```

ENTRY INIT14(N,X)
  X(1) = 2.5D1
  X(2) = 5.0D0
  X(3) = -5.0D0
  X(4) = -1.0D0
RETURN

ENTRY FCN14(M,N,X,FVEC)
  DO 10 I=1,M
    TEMP=I/5.0D0
    TMP1(I)=X(1)+TEMP*X(2)-DEXP(TEMP)
    TMP2(I)=X(3)+DSIN(TEMP)*X(4)-DCOS(TEMP)
    FVEC(I)=TMP1(I)**2+TMP2(I)**2
10  CONTINUE
RETURN

ENTRY JAC14(M,N,X,FJAC)
  DO 20 I=1,M
    TEMP=I/5.0D0
    TI=DSIN(TEMP)
    FJAC(I,1) = 2.0D0*TMP1(I)
    FJAC(I,2) = TEMP*FJAC(I,1)
    FJAC(I,3) = 2.0D0*TMP2(I)
    FJAC(I,4) = TI*FJAC(I,3)
20  CONTINUE
RETURN
END

SUBROUTINE PROB15(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 15: CHEBYQUAD FUNCTION
C
C*****
  REAL*8 X(N),FVEC(M),FJAC(M,N),TEMP,TMP1,TMP2,IEV,TI,DX

  ENTRY INIT15(N,X)
    TEMP=1.0D0/(N+1)
    DO 10 J=1,N
      X(J)=J*TEMP
10  CONTINUE
RETURN

  ENTRY FCN15(M,N,X,FVEC)
    CALL VSET(M,FVEC,0.0D0)
    DO 30 J = 1, N
      TMP1=1.0D0
      TMP2=2.0D0*X(J)-1.0D0
      TEMP=2.0D0*TMP2
      DO 20 I=1,M
        FVEC(I)=FVEC(I)+TMP2
        TI=TEMP*TMP2-TMP1
        TMP1=TMP2
        TMP2=TI
20  CONTINUE
30  CONTINUE
    DX=1.0D0/N
    IEV=-1
    DO 40 I=1,M
      FVEC(I)=DX*FVEC(I)
      IF(IEV.GT.0) FVEC(I)=FVEC(I)+1.0D0/(I**2-1.0D0)
      IEV = -IEV
40  CONTINUE
RETURN

  ENTRY JAC15(M,N,X,FJAC)
    DX = 1.0D0/N

```



```

DO 60 J = 1, N
  TMP1 = 1.0D0
  TMP2 = 2.0D0*X(J) - 1.0D0
  TEMP = 2.0D0*TMP2
  TMP3 = 0.0D0
  TMP4 = 2.0D0
  DO 50 I = 1, M
    FJAC(I,J) = DX*TMP4
    TI = 4.0D0*TMP2 + TEMP*TMP4 - TMP3
    TMP3 = TMP4
    TMP4 = TI
    TI = TEMP*TMP2 - TMP1
    TMP1 = TMP2
    TMP2 = TI
50  CONTINUE
60  CONTINUE
RETURN
END

```

```

SUBROUTINE PROB16(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 16: BROWN ALMOST-LINEAR FUNCTION
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),PROD,SUM
SAVE PROD

ENTRY INIT16(N,X)
  CALL VSET(N,X,0.5D0)
RETURN

ENTRY FCN16(M,N,X,FVEC)
  SUM=-(N+1)
  PROD=1.0D0
  DO 10 J = 1, N
    SUM=SUM+X(J)
    PROD=X(J)*PROD
10  CONTINUE
  DO 20 I=1,N-1
    FVEC(I)=X(I)+SUM
20  CONTINUE
  FVEC(N)=PROD-1.0D0
RETURN

ENTRY JAC16(M,N,X,FJAC)
  DO 60 J=1,N
    DO 30 I=1,N-1
      FJAC(I,J)=1.0D0
30  CONTINUE
      FJAC(J,J)=2.0D0
      IF(X(J).EQ.0.0D0) THEN
        PROD=1.0D0
        DO 50 K=1,N
          IF(K.NE.J) PROD=X(K)*PROD
50  CONTINUE
          FJAC(N,J)=PROD
        ELSE
          FJAC(N,J)=PROD/X(J)
        ENDIF
60  CONTINUE
RETURN
END

```

```

SUBROUTINE PROB17(M,N,X,FVEC,FJAC)
C*****

```

```

C
C   PROBLEM 17: OSBORNE 1 FUNCTION
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),Y(33),TMP1(33),TMP2(33),TEMP
SAVE TMP1,TMP2,Y
COMMON /STEPSZ/ISLONG /PAR/IPAR(5)
DATA Y/8.44D-1,9.08D-1,9.32D-1,9.36D-1,9.25D-1,9.08D-1,8.81D-1,
&      8.5D-1, 8.18D-1,7.84D-1,7.51D-1,7.18D-1,6.85D-1,6.58D-1,
&      6.28D-1,6.03D-1,5.8D-1, 5.58D-1,5.38D-1,5.22D-1,5.06D-1,
&      4.9D-1, 4.78D-1,4.67D-1,4.57D-1,4.48D-1,4.38D-1,4.31D-1,
&      4.24D-1,4.2D-1, 4.14D-1,4.11D-1,4.06D-1/

ENTRY INIT17(N,X)
  X(1) = 0.5D0
  X(2) = 1.5D0
  X(3) = -1.0D0
  X(4) = 1.0D-2
  X(5) = 2.0D-2

C ----- MAKE NOISE ON DATA POINTS ...
  CALL NOISEY(33,Y)
  RETURN

ENTRY FCN17(M,N,X,FVEC)
C ----- TEST FOR POSSIBLE OVERFLOW ...
  ISLONG=1
  IF(IPAR(5).EQ.1.AND.(X(4).LT.-0.1D0 .OR. X(5).LT.-0.1D0))
&    RETURN
  ISLONG=0

  DO 10 I = 1, 33
    TEMP = 10*(I-1)
    TMP1(I)=DEXP(-X(4)*TEMP)
    TMP2(I)=DEXP(-X(5)*TEMP)
    FVEC(I)=Y(I)-X(1)-X(2)*TMP1(I)-X(3)*TMP2(I)
10  CONTINUE
  RETURN

ENTRY JAC17(M,N,X,FJAC)
  DO 20 I = 1, 33
    TEMP=10*(I-1)
    FJAC(I,1)=-1.0D0
    FJAC(I,2)=-TMP1(I)
    FJAC(I,3)=-TMP2(I)
    FJAC(I,4)=TEMP*X(2)*TMP1(I)
    FJAC(I,5)=TEMP*X(3)*TMP2(I)
20  CONTINUE
  RETURN
END

SUBROUTINE PROB18(M,N,X,FVEC,FJAC)
C*****
C
C   PROBLEM 18: OSBORNE 2 FUNCTION
C
C*****
REAL*8 X(N),FVEC(M),FJAC(M,N),Y(65),TEMP,T1,T2,T3,
&      TMP1(65),TMP2(65),TMP3(65),TMP4(65)
SAVE TMP1,TMP2,TMP3,TMP4,Y
COMMON /STEPSZ/ISLONG /PAR/IPAR(5)
DATA Y/1.366D0,1.191D0,1.112D0,1.013D0,9.91D-1,8.85D-1,8.31D-1,
&      8.47D-1,7.86D-1,7.25D-1,7.46D-1,6.79D-1,6.08D-1,6.55D-1,
&      6.16D-1,6.06D-1,6.02D-1,6.26D-1,6.51D-1,7.24D-1,6.49D-1,
&      6.49D-1,6.94D-1,6.44D-1,6.24D-1,6.61D-1,6.12D-1,5.58D-1,
&      5.33D-1,4.95D-1,5.0D-1, 4.23D-1,3.95D-1,3.75D-1,3.72D-1,
&      3.91D-1,3.96D-1,4.05D-1,4.28D-1,4.29D-1,5.23D-1,5.62D-1,

```

```

&      6.07D-1,6.53D-1,6.72D-1,7.08D-1,6.33D-1,6.68D-1,6.45D-1,
&      6.32D-1,5.91D-1,5.59D-1,5.97D-1,6.25D-1,7.39D-1, 7.1D-1,
&      7.29D-1,7.2D-1, 6.36D-1,5.81D-1,4.28D-1,2.92D-1,1.62D-1,
&      9.8D-2,5.4D-2/

ENTRY INIT18(N,X)
  X(1) = 1.3D0
  X(2) = 6.5D-1
  X(3) = 6.5D-1
  X(4) = 0.7D0
  X(5) = 0.6D0
  X(6) = 3.0D0
  X(7) = 5.0D0
  X(8) = 7.0D0
  X(9) = 2.0D0
  X(10) = 4.5D0
  X(11) = 5.5D0

C ----- MAKE NOISE ON DATA POINTS ...
  CALL NOISEY(65,Y)
  RETURN

  ENTRY FCN18(M,N,X,FVEC)
C ----- TEST FOR POSSIBLE OVERFLOW ...
  ISLONG=1
  IF(IPAR(5).EQ.1.AND.-X(5)*6.4D0.GT.3.0D1) RETURN

  DO 10 I=1,65
    TEMP=(I-1)/1.0D1
    T1=-X(6)*(TEMP-X(9))**2
    T2=-X(7)*(TEMP-X(10))**2
    T3=-X(8)*(TEMP-X(11))**2

C ----- TEST FOR POSSIBLE OVERFLOW ...
    IF(IPAR(5).EQ.1.AND.(T1.GT.3.0D1 .OR. T2.GT.3.0D1 .OR.
&      T3.GT.3.0D1)) RETURN

    TMP1(I) = DEXP(-X(5)*TEMP)
    TMP2(I) = DEXP(T1)
    TMP3(I) = DEXP(T2)
    TMP4(I) = DEXP(T3)
    FVEC(I)=Y(I)-
&      (X(1)*TMP1(I)+X(2)*TMP2(I)+X(3)*TMP3(I)+X(4)*TMP4(I))
10  CONTINUE
    ISLONG=0
  RETURN

ENTRY JAC18(M,N,X,FJAC)
  DO 20 I=1,65
    TEMP=(I-1)/1.0D1
    FJAC(I,1)=-TMP1(I)
    FJAC(I,2)=-TMP2(I)
    FJAC(I,3)=-TMP3(I)
    FJAC(I,4)=-TMP4(I)
    FJAC(I,5)=TEMP*X(1)*TMP1(I)
    FJAC(I,6)=X(2)*(TEMP - X(9))**2*TMP2(I)
    FJAC(I,7)=X(3)*(TEMP - X(10))**2*TMP3(I)
    FJAC(I,8)=X(4)*(TEMP - X(11))**2*TMP4(I)
    FJAC(I,9)=-2.0D0*X(2)*X(6)*(TEMP - X(9))*TMP2(I)
    FJAC(I,10)=-2.0D0*X(3)*X(7)*(TEMP - X(10))*TMP3(I)
    FJAC(I,11)=-2.0D0*X(4)*X(8)*(TEMP - X(11))*TMP4(I)
20  CONTINUE
  RETURN
END

```

VITA^γ

Wei Yuan

Candidate for the Degree of

Master of Science

Thesis: AN ACCELERATED LEVENBERG-MARQUARDT ALGORITHM FOR
NONLINEAR LEAST SQUARES PROBLEMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Changsha, Hunan, People's Republic of China, May 3, 1959, the son of Liantang Yuan and Fangdu Zhu.

Education: Received the Bachelor of Science degree in Mathematics from Hunan Normal University, Changsha, Hunan, People's Republic of China, in December, 1981; received the Master of Science degree in Applied Mathematics from Huazhong University of Science and Technology, Wuhan, Hubei, People's Republic of China, in July, 1984; completed requirements for the Master of Science degree at Oklahoma State University in July, 1992.

Professional Experience: Lecturer and scientific programmer, Department of Applied Mathematics, Huazhong University of Science and Technology, Wuhan, Hubei, People's Republic of China, from September, 1984, to March, 1990.