AN AUGMENTED JACOBIAN MATRIX ALGORITHM FOR TRACKING HOMOTOPY ZERO CURVES

by

Stephen Clyde Billups

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

APPROVED:

L. T. Watson, Chairman

J. P. Bixler

R. B. Minton

September, 1985

Blacksburg, Virginia

AN AUGMENTED JACOBIAN MATRIX ALGORITHM FOR TRACKING HOMOTOPY ZERO CURVES

by

Stephen Clyde Billups

Committee Chairman: Layne T. Watson
Computer Science

(ABSTRACT)

There are algorithms for finding zeros or fixed points of nonlinear systems of (algebraic) equations that are globally convergent for almost all starting points, i.e., with probability one. The essence of all such algorithms is the construction of an appropriate homotopy map and then tracking some smooth curve in the zero set of this homotopy map. The augmented Jacobian matrix algorithm is part of the software package HOMPACK, and is based on an algorithm developed by W.C. Rheinboldt. The algorithm exists in two forms–one for dense Jacobian matrices, and the other for sparse Jacobian matrices.

Acknowledgements

I want to thank Dr. Watson for all he has taught me, the many hours of help he has given me, and for all of his patience. I also want to thank my parents who have supported me throughout. But I especially want to thank my father whose excellent example continually motivates me to achieve. I hope I can make him as proud of me as I am of him.

1. Introduction. Homotopy algorithms are the most powerful methods for solving nonlinear systems of equations currently available. While they are computationally more expensive than other methods, these algorithms can often present a savings in human effort by eliminating considerable work in finding a good starting point. What's more, homotopy algorithms can handle problems which simply cannot be solved effectively by other means [22]. Thus, while still considered mainly methods of last resort, homotopy algorithms are an important addition to the field of numerical computation. With that in mind, it makes sense to continually try out new approaches in hopes of producing even more powerful and efficient homotopy algorithms. This is the motivation for developing the augmented Jacobian matrix algorithm.

Homotopy algorithms are related to some long established techniques of numerical analysis called continuation methods. Some related ideas are parameter continuation, incremental loading, displacement incrementation, and invariant imbedding. The idea behind continuation methods is to solve a series of problems as some parameter λ is slowly varied monotonically. Thus, a curve is traced out, producing a different point for each λ. Homotopy algorithms are based on this same curve tracking philosophy, however they are concerned not with the curve itself, but rather with where the curve ends up.

Another difference is that homotopy algorithms are not disturbed if the curve reverses direction in the λ component. The continuation methods require that λ change in only one direction. If the curve turns around, the continuation method quits. In contrast, the homotopy method will follow the curve no matter how much it turns. Thus, homotopy methods, although similar to continuation methods, are a distinct field in themselves.

The first homotopy algorithms were the globally convergent simplicial fixed point algorithms of Scarf, Kuhn, Merrill, Eaves, Saigal, and Todd (See references 1,12,43 in [22]). These algorithms had a theoretical base in topology, were potentially extremely powerful, but were horribly inefficient in their early forms. They did however represent a fundamental breakthrough, separating homotopy methods from the related idea of continuation.

Another significant advance was the development of differential equation based algorithms (See references 5,15,25,27,28,30-32,38-41,68 in [22]). These algorithms were quite successful on many practical problems, however, they had a major problem. A Jacobian matrix somewhere could become singular, and the computer implementation would either experience great difficulty or the method would fail completely.

This problem was solved by the development of probability one homotopy methods by S.N. Chow, J. Mallet-Paret, and J.A. Yorke [2]. These methods were constructed so that for almost all (i.e., with probability one) choices of some parameter vector involved in the homotopy map, there are no singular points, and the method is globally convergent. Thus, the problem of singular Jacobian matrices was eliminated.

Advances continue to be made. So far, homotopy algorithms have been created which are globally convergent for Brouwer fixed point problems [13], certain classes of zero finding problems [16], nonlinear programming problems [14], and two-point boundary value problems [17]. In addition, Morgan [9,10] obtained some elegant results for polynomial systems.

The algorithm presented here is one of three qualitatively different algorithms included in the software package HOMPACK, which is currently being developed at Sandia National Laboratories, General Motors Research Laboratories, and Virginia Polytechnic Institute

and State University. The algorithms in HOMPACK are known as probability one globally convergent homotopy algorithms. These algorithms are also referred to as "continuous" methods, to distinguish them from the simplicial homotopy methods.

The augmented Jacobian matrix algorithm is based on an algorithm developed by W.C. Rheinboldt and J.V. Burkardt [11], but with some significant differences: (1) a Hermite cubic rather than a linear predictor is used; (2) a tangent vector rather than a standard basis vector is used to augment the Jacobian matrix of the homotopy map; (3) updated QR factorizations and quasi-Newton updates are used rather than Newton's method; (4) different step size control, necessitated by the use of quasi-Newton iterations, is used; (5) a different scheme for locating the target point at $\lambda = 1$ is used which allows the Jacobian matrix of F to be singular at the solution.

Two versions of the algorithm exist–one for dense Jacobian matrices, and the other for sparse Jacobian matrices. The following section describes the theory behind the algorithm. Sections 3–6 describe the four phases of the dense algorithm, and section 7 discusses the sparse matrix version. Finally, test results and conclusions are presented in sections 8 and 9.

2. Theoretical Background. HOMPACK is designed to solve two types of nonlinear problems: zero finding problems, and fixed point problems. The frameworks for these problems are slightly different, so they will be discussed separately.

The fixed point problem will be considered first. Let B be the closed unit ball in n-dimensional real Euclidean space E^n, and let $f : B \rightarrow B$ be a C^2 map. Define the homotopy

map $\rho_a : [0,1) \times B \to E^n$ by

$$\rho_a(\lambda, x) = \lambda(x - f(x)) + (1 - \lambda)(x - a). \tag{1}$$

The fundamental result [2] is that for almost all a (in the sense of Lebesgue measure) in the interior of B, there is a zero curve $\gamma \subset [0,1] \times B$ of ρ_a, along which the Jacobian matrix $D\rho_a(\lambda, x)$ has rank n, emanating from $(0, a)$ and reaching a point $(1, \bar{x})$, where $\bar{x}$ is a fixed point of f. Thus with probability one, picking a starting point $a \in \text{int } B$ and following γ leads to a fixed point $\bar{x}$ of f. This justifies the phrase "globally convergent with probability one".

The zero finding problem

$$F(x) = 0, \tag{2}$$

where $F : E^n \to E^n$ is a C^2 map, is more complicated. Suppose there exists a C^2 map

$$\rho : E^m \times [0,1) \times E^n \to E^n$$

such that

1) the $n \times (m + 1 + n)$ Jacobian matrix $D\rho(a, \lambda, x)$ has rank n on the set

$$\rho^{-1}(0) = \{(a, \lambda, x) \mid a \in E^m, 0 \leq \lambda < 1, x \in E^n, \rho(a, \lambda, x) = 0\},$$

and for any fixed $a \in E^m$,

2) $\rho_a(0, x) = \rho(a, 0, x) = 0$ has a unique solution x_0,

3) $\rho_a(1, x) = F(x)$,

4) $\rho_a^{-1}(0)$ is bounded.

Then the supporting theory [16,17,18] says that for almost all $a \in E^m$ there exists a zero curve γ of ρ_a, along which the Jacobian matrix $D\rho_a$ has rank n, emanating from $(0, x_0)$ and reaching a zero $\bar{x}$ of F at $\lambda = 1$. γ does not intersect itself and is disjoint from any other zeros of ρ_a. An obvious choice for ρ_a is

$$\rho_a(\lambda, x) = \lambda\, F(x) + (1 - \lambda)\,(x - a). \tag{3}$$

This satisfies properties 1)-3), but not necessarily 4). There are fairly general sufficient conditions on $F(x)$ so that (3) will satisfy property 4), but for some practical problems of interest the homotopy map (3) will not suffice. This is why HOMPACK is designed to handle arbitrary homotopy maps $\rho_a(\lambda, x)$ satisfying properties 1)-4).

The basic idea behind the homotopy algorithms is simple: just follow the zero curve γ from $(0, a)$ until a point $(1, \bar{x})$ is found. $\bar{x}$ will then be the desired zero or fixed point. Depending on the problem, the homotopy map $\rho_a(\lambda, x)$ may be given by (1), (3), or something else that is even nonlinear in λ. Once the homotopy map has been determined, the problem is simply to track the zero curve. This curve tracking is aided by the following parameterization. Assuming that $F(x)$ is C^2, a is such that the Jacobian matrix $D\rho_a(\lambda, x)$ has full rank along γ , and γ is bounded, the zero curve γ is C^1 and can be parameterized by arclength s. Thus, $\lambda = \lambda(s)$, $x = x(s)$ along γ , and

$$\rho_a(\lambda(s), x(s)) = 0 \tag{4}$$

identically in s. Therefore

$$\frac{d}{ds}\rho_a(\lambda(s), x(s)) = D\rho_a(\lambda(s), x(s)) \begin{pmatrix} \dfrac{d\lambda}{ds} \\ \dfrac{dx}{ds} \end{pmatrix} = 0, \tag{5}$$

$$\left\| \left(\frac{d\lambda}{ds}, \frac{dx}{ds} \right) \right\|_2 = 1. \tag{6}$$

With the initial conditions

$$\lambda(0) = 0, \quad x(0) = a, \tag{7}$$

the zero curve γ is the trajectory of the initial value problem (5-7). When $\lambda(\bar{s}) = 1$, the corresponding $x(\bar{s})$ is a zero (or fixed point) of $F(x)$.

The curve tracking algorithm consists of four phases: prediction, correction, step-size estimation, and computation of the solution at $\lambda = 1$ (the end game). Each of the phases will be discussed in turn.

3. Prediction. The prediction phase involves finding a point $Z^{(0)}$ close to γ somewhere farther along the zero curve than the current point $P^{(2)}$. This is achieved by creating a local model of γ and taking a step of some size h along this local model. Two types of polynomials are used as local models. For the first step along the curve, a linear model is used. For the remaining steps, the model chosen is a Hermite cubic interpolating polynomial.

The linear model is used only for the first step. It is constructed by computing the unit tangent vector $T^{(2)}$ of γ at the point $(0, a)$. With this information, the predictor point is calculated by

$$Z^{(0)} = (0, a) + hT^{(2)}. \tag{8}$$

The cubic model is used for all of the remaining steps along the zero curve. Having the points $P^{(1)} = (\lambda(s_1), x(s_1)), P^{(2)} = (\lambda(s_2), x(s_2))$ on γ with corresponding tangent vectors

$$T^{(1)} = \begin{pmatrix} \frac{d\lambda}{ds}(s_1) \\ \frac{dx}{ds}(s_1) \end{pmatrix}, \qquad T^{(2)} = \begin{pmatrix} \frac{d\lambda}{ds}(s_2) \\ \frac{dx}{ds}(s_2) \end{pmatrix},$$

the prediction $Z^{(0)}$ of the next point on γ is given by

$$Z^{(0)} = p(s_2 + h), \tag{9}$$

where $p(s)$ is the Hermite cubic interpolating $(\lambda(s), x(s))$ at s_1 and s_2. Precisely,

$$p(s_1) = P^{(1)}, \quad p'(s_1) = T^{(1)},$$

$$p(s_2) = P^{(2)}, \quad p'(s_2) = T^{(2)},$$

and each component of $p(s)$ is a polynomial in s of degree less than or equal to 3.

In order to implement either prediction method, a means of calculating the tangent vector $T^{(2)}$ at a point $P^{(2)}$ is required. The method used here is to solve the system

$$\begin{bmatrix} D\rho_a\left(P^{(2)}\right) \\ T^{(1)\,t} \end{bmatrix} z = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{10}$$

for z, where $D\rho_a$ is the $n \times (n+1)$ Jacobian of ρ_a, and $T^{(1)}$ is a vector chosen to insure that z has the correct sign. For the linear predictor, $T^{(1)} = (1, 0, \ldots, 0)^t$, and for the cubic predictor, $T^{(1)}$ is chosen as the tangent vector at the previous point $P^{(1)}$. Normalizing z gives

$$T^{(2)} = \frac{z}{\|z\|}. \tag{11}$$

There are many ways other than equation (10) to compute a tangent vector. In fact, all that is necessary is that the tangent vector satisfies the equation $\left[D\rho_a\left(P^{(2)}\right)\right] z = 0$, which insures that the change in each component of ρ along the tangent vector is zero at $P^{(2)}$. However, this equation is underdetermined and thus has infinitely many solutions. In

order to get a unique solution, the system is augmented by the equation $T^{(1)\,t} z = 1$, giving equation (10). It is the augmentation of the Jacobian matrix with this additional row which motivates the name "augmented Jacobian matrix algorithm."

The last row of (10) is chosen for two reasons. First, it causes the solution z to be roughly of unit magnitude, thus reducing the risk of losing precision in the normalization phase. Second, this additional row forces the computed tangent to make an acute angle with $T^{(1)}$. When $T^{(1)}$ is the previous tangent vector, $T^{(2)}$ should have the correct sign based on the fact that the direction of the zero curve is not likely to change by more than 90° over a small step. When $T^{(1)}$ is the basis vector e_1 used for the first step, the acute angle criteria forces the computed tangent to have a positive λ component. This is the correct direction for the first step.

The augmentation scheme chosen here differs from Rheinboldt's algorithm [11]. His scheme is to augment the Jacobian matrix by a standard basis vector, and then to compute a determinant in order to determine the correct sign. This scheme was rejected for three reasons. First, the computation of a determinant is expensive. Second, determinant calculations are prone to computational difficulties; thus, there is no reason to believe that the determinant calculation would give better results than the acute angle test. Finally, the chosen augmentation scheme lends itself better to the quasi-Newton correction process.

Since the computation of the tangent vectors involves the solution of equation (10), a means of solving a linear system $Ax = b$ is required. The method chosen is QR decomposition. This technique involves factoring the matrix A into an orthogonal matrix Q, and an upper triangular matrix R, so that $A = QR$. x can then be found by solving the system

$Rx = Q^t b$, which is simply a matter of backward substitution.

The reasons for choosing QR factorization are two-fold. First, QR decomposition is very stable. Since Q is orthogonal, $\|R\|_2 = \|QR\|_2 = \|A\|_2$. Thus, the decomposition does not magnify size differences of elements of A when forming R. Second, the QR decomposition allows for cheap rank-one updates, which is of tremendous value in the quasi-Newton correction step.

The QR factorization is obtained by applying a series of Householder transformations. A is transformed to R by premultiplying A by a series of $n-1$ Householder transformations Q_i. Each Q_i zeros out the elements of the ith column of $Q_{i-1}\cdots Q_1 A$ below the main diagonal, while leaving the first $i-1$ columns unchanged. The orthogonal matrix Q is simply the product $Q_1 \cdots Q_{n-1}$. Each Q_i has the form

$$Q_i = I - u_i u_i^t,$$

where $(u_i)_j = 0$, for $j < i$, and the remaining elements of u_i are selected so that Q_i is orthogonal and induces the desired zeroes in column i.

Having solved equation (10) and calculated the unit tangent vector, the predictor point $Z^{(0)}$ is easily obtained from (8) or (9). The next step is the correction process.

4. Correction. The correction phase is concerned with returning to the zero curve γ after computing the predictor point $Z^{(0)}$. This job is performed by a quasi-Newton algorithm. The basis of the algorithm is to produce successive iterates $Z^{(k)}$, $k = 1, 2, \ldots$, which come closer and closer to the zero curve. The iterates converge to the desired point $Z^{(*)}$ on the zero curve. A restriction that the iterates remain in a hyperplane perpendicular to the

previous tangent vector is employed so that the correction process will return to the zero curve as directly as possible.

In order to discuss the quasi-Newton algorithm, Newton's method will first be described. Newton's method is an iterative process which finds the root $x^{(*)}$ of a nonlinear function $F(x)$. Each iteration of the algorithm starts with a point $x^{(k)}$, and produces a new point $x^{(k+1)}$ which is closer to the root. The computation of $x^{(k+1)}$ is performed by creating a linear model M of the function at the point $x^{(k)}$. Finding the root of this model produces the next iterate $x^{(k+1)}$.

The linear model M is defined by

$$M(x^{(k)} + p) = F(x^{(k)}) + DF(x^{(k)})p, \tag{12}$$

where $DF(x^{(k)})$ is the Jacobian matrix of F evaluated at $x^{(k)}$. Finding the root of M involves finding $\Delta x^{(k)}$ such that $M(x^{(k)} + \Delta x^{(k)}) = 0$. This is achieved by solving

$$DF(x^{(k)})\Delta x^{(k)} = -F(x^{(k)}). \tag{13}$$

The next iterate is then computed by $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$. The iteration stops when the correction $\Delta x^{(k)}$ is smaller than some tolerance.

The quasi-Newton algorithm works exactly like the Newton algorithm except that instead of computing the exact Jacobian matrix at each iterate $x^{(k)}$, an approximation $A^{(k)}$ to the Jacobian matrix is used. After each corrector step is made, a Broyden rank-one update is applied to the matrix in order to produce the next approximation $A^{(k+1)}$. The Broyden update is defined by the equation

$$A^{(k+1)} = A^{(k)} + \frac{\left(y^{(k)} - A^{(k)}s^{(k)}\right)s^{(k)t}}{s^{(k)t}s^{(k)}}, \tag{14}$$

where $y^{(k)} = F(x^{(k+1)}) - F(x^{(k)})$, and $s^{(k)} = x^{(k+1)} - x^{(k)}$.

Considerable computational effort can be saved by performing the updates in QR-factored form. It is possible to obtain the factorization $Q^{(k+1)}R^{(k+1)}$ of $A^{(k+1)}$ simply by updating the factors $Q^{(k)}$ and $R^{(k)}$ of $A^{(k)}$. This is much cheaper than explicitly calculating and factoring $A^{(k+1)}$. The savings amounts to a reduction from $O(n^3)$ to $O(n^2)$ operations.

The idea behind updating in factored form is not difficult. For simplicity, the update formula (14) can be written

$$A^{(k+1)} = A^{(k)} + uv^t, \tag{15}$$

where $u, v \in E^n$. Assuming the factors of $A^{(k)}$ are known, equation (15) can be rewritten

$$A^{(k+1)} = Q^{(k)}R^{(k)} + uv^t = Q^{(k)}\left(R^{(k)} + wv^t\right), \tag{16}$$

where $w = Q^{(k)\,t}u$. Thus, the first step is to calculate the QR decomposition

$$R^{(k)} + wv^t = \tilde{Q}\tilde{R}. \tag{17}$$

Then, $R^{(k+1)} = \tilde{R}$ and $Q^{(k+1)} = Q^{(k)}\tilde{Q}$.

The fact that the matrix $R^{(k)} + wv^t$ is a rank-one update of an upper-triangular matrix allows the QR decomposition to be performed in $O(n^2)$ operations rather than the $O(n^3)$ operations usually required to perform a factorization. The decomposition is performed by premultiplying $R^{(k)} + wv^t$ by $2n-2$ Jacobi rotations. These Jacobi rotations are orthogonal matrices constructed in such a way as to zero out a particular element of a matrix while only changing two rows of that matrix.

The use of Jacobi rotations to perform the QR factorization of $R^{(k)} + wv^t$ is a two step process. First, $n-1$ Jacobi rotations are applied to zero out in succession rows $n, n-1, \ldots, 2$

of wv^t, by combining rows i and $i-1$ to zero out row i. This is possible because each row of wv^t is simply a scalar multiple of any other row. The effect of each rotation on $R^{(k)}$ is to alter some existing elements and to introduce one new element directly below the diagonal in the $(i, i-1)$ position. Thus, the first $n-1$ rotations transform $R^{(k)} + wv^t$ into an upper Hessenberg matrix. The second step of the process is to apply $n-1$ additional Jacobi rotations to successively zero out the $(i, i-1)$ element, $i = 2, \ldots, n$, by combining rows $i-1$ and i. The resulting upper triangular matrix is $\tilde{R}$, and $\tilde{Q}^t$ is the product of the $2n-2$ Jacobi rotations.

The reason for using the quasi-Newton approach instead of the Newton approach is to improve computational efficiency. The evaluation of Jacobian matrices can be very expensive for some problems. Thus, the fact that the quasi-Newton algorithm avoids evaluating the Jacobian represents a considerable savings. In addition, applying the rank-one updates in factored form further reduces the cost per iteration by saving in matrix factorizations.

The disadvantages of the quasi-Newton algorithm are three-fold. First, convergence is slower. Newton's method converges quadratically whereas the quasi-Newton method converges only super-linearly. Second, the radius of convergence is generally smaller. Thus, the starting point $x^{(0)}$ may need to be closer to the root for the iteration to converge. Finally, the overhead for implementing the quasi-Newton algorithm is greater. Thus, small dimensional problems with inexpensive Jacobians are better solved with Newton's method, but large-dimensional problems, or problems with expensive Jacobians are solved more efficiently by using the quasi-Newton approach.

Applying the quasi-Newton algorithm to the corrector step involves finding the solution

$y^{(*)}$ of the augmented nonlinear system

$$\begin{pmatrix} \rho_a(y) \\ T^{(2)\,t}\left(y - Z^{(0)}\right) \end{pmatrix} = 0. \tag{18}$$

This equation insures that the solution $y^{(*)}$ lies on the zero curve, and also, the last row of this system insures that the solution $y^{(*)}$ and the predictor point $Z^{(0)}$ lie in a hyperplane perpendicular to the tangent vector $T^{(2)}$.

The quasi-Newton iteration to solve (18) is defined by

$$Z^{(k+1)} = Z^{(k)} + \Delta Z^{(k)} \qquad k = 0, 1, \ldots \tag{19}$$

where the corrector step $\Delta Z^{(k)}$ is computed by

$$\begin{bmatrix} A^{(k)} \\ T^{(2)\,t} \end{bmatrix} \Delta Z^{(k)} = \begin{pmatrix} -\rho_a\left(Z^{(k)}\right) \\ 0 \end{pmatrix}. \tag{20}$$

Here $A^{(k)}$ is the approximation to the Jacobian matrix $D\rho_a(Z^{(k)})$ obtained by successive rank-one updates.

It should be noted that the quasi-Newton iteration must start with an exact Jacobian matrix in order for the convergence to be super-linear. The Jacobian used to start the process is the one used to compute the tangent vector $T^{(2)}$ at the previous point $P^{(2)}$. Precisely, letting $Z^{(-1)} = P^{(2)}$, $A^{(-1)} = D\rho_a(P^{(2)})$, $\Delta Z^{(-1)} = Z^{(0)} - P^{(2)}$, and

$$M^{(k)} = \begin{bmatrix} A^{(k)} \\ T^{(2)\,t} \end{bmatrix},$$

the update formulas are

$$M^{(-1)} = \begin{bmatrix} A^{(-1)} \\ T^{(2)\,t} \end{bmatrix} = \begin{bmatrix} D\rho_a\left(P^{(2)}\right) \\ T^{(1)\,t} \end{bmatrix} + e_{n+1}\left(T^{(2)} - T^{(1)}\right)^t, \tag{21}$$

and

$$M^{(k+1)} = M^{(k)} + \frac{\left(\tilde{\Delta}\rho_a - M^{(k)}\Delta Z^{(k)}\right)\Delta Z^{(k)t}}{\Delta Z^{(k)t}\,\Delta Z^{(k)}}, \qquad k = -1, 0, \ldots \tag{22}$$

where

$$\tilde{\Delta}\rho_a = \begin{pmatrix} \rho_a\left(Z^{(k+1)}\right) - \rho_a\left(Z^{(k)}\right) \\ 0 \end{pmatrix}.$$

Note that these updates are actually performed in QR factored form since they are all of rank one.

The goal in the corrector process is to return to the zero curve as directly as possible. An ideal method would be to iterate in a direction perpendicular to the zero curve; however the information needed to do this is unavailable. The next best choice is to iterate perpendicular to the latest tangent vector, as this is the best information available about the direction of the curve.

Two other choices for restricting the iterates were considered and rejected. The first choice was to iterate perpendicular to the previous tangent $T^{(1)}$ instead of the current tangent $T^{(2)}$. The motivation for doing this was to save the rank-one update (21). The strategy turned out to be poor in practice because smaller steps had to be taken in order to stay with the zero curve. The extra work in having to take more steps far outweighed any savings from avoiding the rank-one update.

The other method considered was to iterate perpendicular to a standard basis vector. This method was successfully implemented by Rheinboldt [11]. However, the basis vector seemed to be an even poorer approximation to the direction of the curve than the previous tangent. Thus, this method was also likely to require smaller steps in order to stay with

the curve. In addition, unlike the first alternative, this method represented no savings in matrix updating. Thus, it was rejected.

It is entirely possible that the quasi-Newton process will not converge. This can happen whenever the prediction $Z^{(0)}$ is too far away from the zero curve. To handle this event, a limit is placed on the number of iterations allowed by the quasi-Newton algorithm. If this limit is exceeded, the process is considered to have failed. Therefore, the prediction is abandoned, the step size h is cut in half, and a new, more conservative, predictor is computed.

Because of the failure, the augmented Jacobian matrix used for the quasi-Newton method is probably meaningless. Thus, a new Jacobian matrix is computed at the predictor point. From here, the corrector process continues as before.

When handling these failures, it is possible that the step-size may have to be cut in half several times in order to get convergence. This is perfectly acceptable unless the step-size becomes unreasonably small (i.e., close to the unit roundoff). In this case, the entire homotopy algorithm is abandoned, and an error flag is returned.

The limit on the number of iterations is computed as a function of the error tolerance for tracking the curve. It is reasonable to expect that the quasi-Newton process will provide at least half a digit of extra precision for each iteration. Thus, if n digits of precision are required, the limit on the number of iterations is $2n$. Precisely, the limit is computed by

$$limit = 2\lfloor -\log_{10} arctol + 1 \rfloor, \tag{23}$$

where *arctol* is the tolerance for tracking the curve.

Another possible problem is that the correction process may converge to the wrong curve. As there may be other components of the zero set of ρ_a other than γ, it is entirely possible that the correction process may converge to a point on one of these other components. This component could be a closed loop, or may shoot off to infinity without ever coming near $\lambda = 1$. Thus, tracking this other component may not lead to a solution.

The only way to detect this problem is to place a limit on the number of steps which can be taken in tracking the zero curve. Thus, when this limit is exceeded, it may mean that γ has been lost, and some other curve in the zero set is being tracked. At this point, the algorithm returns an appropriate flag, and the user has the option of continuing on or quitting.

Another problem is that the tangent vector at a point may be computed with the wrong sign. This can happen if the curve is turning so quickly as to change direction by more than 90° over the step taken. If this happens, the correct tangent vector makes an obtuse angle with the previous tangent vector. However, the sign of the computed tangent vector is chosen to make an acute angle with the previous tangent vector. Thus, the algorithm thinks the curve is going in exactly the wrong direction. If no more bad steps are taken, the algorithm effectively retraces its steps, and eventually ends up back at $\lambda = 0$.

A partial solution to this problem is to require that each new tangent forms an angle no greater than 60° with the previous tangent. With this idea, the step-length is chosen so as to keep the angle at a maximum of 60°. If an error is made in computing this step-length, resulting in an angle greater than 60°, the algorithm distrusts the new tangent; thus, the new point is discarded, the step-size is halved, and a more conservative step is made. The

only way the algorithm can get turned around is for the step-length calculation to make an error of at least 100%, resulting in an angle greater than 120°. In this case the sign reversal gives an angle less than 60°, and the error is not detected.

The implementation of this strategy greatly improves the robustness of the curve tracking algorithm. With this strategy, it is possible to track a tightly turning curve much more loosely without losing it. This increased robustness translates into greater efficiency also, as larger steps can be made in tracking the curve without fear of failure.

In summary, the correction phase works by applying a quasi-Newton iteration to the predictor point $Z^{(0)}$ in order to return to the zero curve. The matrices used in computing each corrector step are derived from rank-one updates of the matrix used for computing the tangent in the predictor phase. These updates are performed in QR-factored form. If the algorithm fails to converge or if the correction produces a point whose tangent forms more than a 60° angle with the previous tangent, the step is considered to be bad. In this case, the step is discarded, and a more conservative step of half the size is made.

5. Step-size Estimation. The step-size estimation algorithm operates by trying to keep the number of corrector iterations at each step relatively constant. This is achieved by estimating the curvature of γ for the step, and by computing the ideal predictor error for the next step. These two pieces of information together with various restrictions aimed at stabilizing the algorithm are used to compute the step-size h to be used for the next predictor step.

The goal in computing the step-size is to allow the curve to be tracked as efficiently as possible. Small steps are inefficient because many of them are needed in order to track the

entire curve. On the other hand, large steps can be inefficient because they may cause a large number of correction iterations. Worse yet, they may cause failure of the correction process altogether, resulting in a considerable amount of wasted effort as the attempted step is abandoned, and a smaller step is taken.

Achieving this goal can be accomplished to some degree by a simple heuristic–keep the number of corrector iterations constant. The rule cannot be followed exactly due to imperfect information; however, a scheme developed by Rheinboldt is reasonably effective [11].

The first step of Rheinboldt's scheme is to estimate the curvature of γ for the step to be taken. In order to do this, an approximation to the curvature is calculated at each step by the formula

$$\left\| w^{(k)} \right\| = \frac{2}{\Delta s_k} \left| \sin\left(\alpha_k/2\right) \right|, \tag{24}$$

where

$$w^{(k)} = \frac{T^{(k)} - T^{(k-1)}}{\Delta s_k}, \qquad \alpha_k = \arccos\left(T^{(k)\,t} T^{(k-1)}\right), \qquad \Delta s_k = \left\| P^{(k)} - P^{(k-1)} \right\|.$$

Intuitively, α_k represents the angle between the last two tangent vectors, and the curvature is approximated by the Euclidean norm of the difference between these two tangents divided by Δs_k.

This curvature data can then be extrapolated to give an estimate of the curvature for the next step by the formula

$$\hat{\xi}_k = \left\| w^{(k)} \right\| + \frac{\Delta s_k}{\Delta s_k + \Delta s_{k-1}} \left(\left\| w^{(k)} \right\| - \left\| w^{(k-1)} \right\| \right). \tag{25}$$

It should be noted that Rheinboldt's formula for the curvature (24) is cheaper to evaluate than the equivalent formula $\|(T^{(2)} - T^{(1)})/\Delta s\|$. The sine formula requires only one vector operation and two implicit function evaluations, whereas the explicit equation requires two vector operations (which are more expensive).

The extrapolation formula (25) can result in a negative estimate for the curvature. This is clearly inappropriate. Thus, the estimate is revised so as to produce a positive curvature by the formula

$$\xi_k = \max(\xi_{min}, \hat{\xi}_k) \quad \text{for some small} \quad \xi_{min} > 0. \tag{26}$$

The next step in Rheinboldt's step-length algorithm is to calculate what the ideal error δ_k for the predictor should be. In other words, this step is designed to determine how far the predictor point should be from γ in order for the corrector process to take the desired number of iterations. Rheinboldt accomplishes this by using information about the convergence rate of Newton's method. Since the quasi-Newton algorithm is being employed here, Rheinboldt's scheme cannot be used. Instead a much simpler idea is implemented.

The ideal error is computed as a function of the tolerance for tracking the zero curve. If the tolerance becomes stricter, more iterations are necessary to achieve convergence. Thus, the initial error is made smaller so that the number of iterations stays the same. The ideal error is computed by

$$\tilde{\delta}_k = (arcae + arcre\|y\|)^{1/4}, \tag{27}$$

where *arcae* and *arcre* are the absolute and relative errors used for tracking the curve.

In order to prevent unreasonably large ideal error estimates, the ideal error is restricted to be no larger than half of the previous step-length. Thus, the final estimate for the ideal

error is

$$\delta_k = \min(h/2, \tilde{\delta}_k). \tag{28}$$

The choice of $h/2$ is somewhat arbitrary. Rheinboldt's algorithm uses h instead; however the $h/2$ scheme worked better on the test problems, based on the criteria of how loosely the curves could be tracked without getting lost.

The final step in Rheinboldt's algorithm is to compute the step-size based on the ideal error and the curvature calculations. The formula for doing this is

$$\hat{h} = \sqrt{\frac{2\delta_k}{\xi_k}}, \tag{29}$$

The above formula is derived from the idea of taking a linear predictor step. Given the approximate curvature ξ_k, the distance from the curve of a linear step of size h is approximated by $\frac{1}{2}h^2\xi_k$. Equation (29) follows from setting this distance equal to the ideal error δ_k.

The question arises, "why use a formula based on a linear predictor when a cubic predictor is actually being employed?" The answer to this is that Equation (29) works well for a linear predictor, and using a cubic predictor should only improve the performance of the curve tracking algorithm. Perhaps a better scheme could be derived to take advantage of the cubic prediction, but the point is, a scheme which works well for a linear predictor should also work for a cubic predictor.

It is entirely possible that the step-length calculations above could produce bizarre step-lengths. To prevent this from ruining the algorithm, some additional restrictions are

placed on the step-length. First, a minimum and maximum step-size h_{min} and h_{max} are chosen as bounds on the step-size. Also, the rate of growth is restricted by the parameters B_{min} and B_{max}. Thus, the step-length is refined by the equation

$$\bar{h} = \min\left\{\max\{h_{\min}, B_{\min}h, \hat{h}\}, B_{\max}h, h_{\max}\right\}. \tag{30}$$

One final restriction is enforced whenever the last step was achieved after a failure of the corrector iteration. In this case, the step-size is not allowed to exceed the smallest step-size h_{fail} known to cause failure in the previous step. Thus, the final step-length is computed by

$$\bar{h} := \min\{h_{\text{fail}}, \bar{h}\}. \tag{31}$$

The step-size estimation algorithm works well for some problems but not as well for others. It works well when the zero curve is relatively straight because the curvature estimates are fairly accurate, and not much of the "special case" logic is invoked. On the other hand, the algorithm is poor for very crooked homotopy zero curves. For these problems, the curvature estimate is terrible, and the algorithm has to fall back on the extra step-size restrictions in order to produce a reasonable step-size.

6. End Game. The end game phase is concerned with finding the particular point on γ where $\lambda = 1$. Intuitively, the idea is to iterate by finding points on γ which are closer and closer to the hyperplane $\lambda = 1$. This phase is entered when the curve tracking algorithm produces a point with λ component greater than or equal to one. This point, together with the previous point on the zero curve, are the starting points for an iterative process which converges to the desired solution.

The implementation of this iterative scheme involves an alternating process of linear prediction and quasi-Newton correcton. First, the prediction phase produces a point on the hyperplane $\lambda = 1$. At this point, a single quasi-Newton correction step is taken to return close to the zero curve, but not necessarily on $\lambda = 1$.

The linear prediction generally works by a secant method. Having two points $P^{(k-1)}$ and $P^{(k)}$ near γ , the secant prediction is calculated by

$$Z^{(k-2)} = P^{(k)} + \left(P^{(k-1)} - P^{(k)}\right) \frac{\left(1 - P_1^{(k)}\right)}{\left(P_1^{(k-1)} - P_1^{(k)}\right)}. \tag{32}$$

The ensuing quasi-Newton correction step produces a point $P^{(k+1)}$. $P^{(k)}$ and $P^{(k+1)}$ are then used in equation (32) to calculate the next predictor point $Z^{(k-1)}$.

While the secant method generally works quite well, it runs the risk of producing a disastrous prediction. To see this, consider what happens when two points with nearly equal λ components are used in equation (32). The resulting predictor point is very far from the zero curve.

To handle this difficulty, the linear prediction is computed by a chord method whenever the results of the secant method are suspect. This chord method requires two points, $P^{(k)}$ and $P^{(opp)}$ on either side of $\lambda = 1$. $P^{(opp)}$ is defined as the most recent iterate opposite of $\lambda = 1$ from $P^{(k)}$. The formula for the prediction is

$$Z^{(k-2)} = P^{(k)} + \left(P^{(opp)} - P^{(k)}\right) \frac{\left(1 - P_1^{(k)}\right)}{\left(P_1^{(opp)} - P_1^{(k)}\right)}. \tag{33}$$

Since the two points "bracket" $\lambda = 1$, the method will always produce a reasonable predictor. However, the chord method converges only linearly as compared to the 1.618 rate of convergence for the secant method. Thus, the secant method is generally preferred.

The decision mechanism governing which method to use is based on the closeness of the secant predictor relative to the iterates $P^{(k)}$ and $P^{(opp)}$. If the secant method produces a point farther from $P^{(k)}$ than $P^{(opp)}$ is, the point is considered unreliable, and the chord method is used instead. Precisely, the chord method is used whenever the following inequality is true.

$$\left\| Z^{(k-2)} - P^{(k)} \right\| > \left\| P^{(k)} - P^{(opp)} \right\| \tag{34}$$

An exception to using these linear prediction schemes occurs with the first step. Since the tangents $T^{(1)}$ and $T^{(2)}$ at $P^{(1)}$ and $P^{(2)}$ are available, a Hermite cubic polynomial $p(s)$ is constructed in order to compute the first prediction point $Z^{(0)}$. By finding the root $\bar{s}$ of the equation $p_1(s) = 1$, the predictor is computed by

$$Z^{(0)} = p(\bar{s}). \tag{35}$$

After the predictor $Z^{(k-2)}$ has been determined, a single quasi-Newton step is taken to get the point $P^{(k+1)}$. It makes little sense to iterate until convergence after each predictor because this extra accuracy does not improve the next predictor dramatically. Thus, considerable work is saved by taking only one quasi-Newton step. An exception to this occurs if the corrector iterate remains on the hyperplane $\lambda = 1$. In this case, the predictor step is skipped, and the corrector process continues until the iterates leave the hyperplane. The corrector step is defined by

$$P^{(k+1)} = Z^{(k-2)} + \Delta Z^{(k-2)}, \tag{36}$$

where $\Delta Z^{(k-2)}$ is the solution to (20). As with the stepping algorithm, the matrix in (20) is produced by the rank one updates (21) and (22) in QR-factored form.

The end game algorithm is designed to work even if the Jacobian of F is singular at the solution. Consider the effect of augmenting the matrix in (20) by $(1,0,\ldots,0)$ instead of by $T^{(2)\,t}$. This scheme, which is used in Rheinboldt's algorithm, is justified by forcing the iterates to remain in the hyperplane $\lambda = 1$. This removes the need to keep computing predictor points; however, a problem comes from the fact that if the Jacobian of F is singular at the solution, then the augmented Jacobian is also singular. In this case, the algorithm fails because equation (20) cannot be solved. To prevent this problem, the augmented Jacobian matrix algorithm uses a scheme which allows the iterates to leave the hyperplane $\lambda = 1$.

One final consideration is that the end game algorithm may never converge. Thus, as in the curve tracking phase, a limit is placed on the number of iterations. If this limit is exceeded, the algorithm quits, returning an appropriate error flag. This limit is computed exactly the same way as in the curve tracking algorithm except that the desired tolerance is much smaller, resulting in a larger limit for the end game.

In summary, the augmented Jacobian matrix algorithm is:

1. $s := 0$, $y := (0,a)$, $ypold := (1,0)$, $h := 0.1$, $failed :=$ false, $firststep :=$ true, $arcae, arcre :=$ absolute, relative error tolerances for tracking γ, $ansae, ansre :=$ absolute, relative error tolerances for the answer.

2. Compute the tangent yp at y, using (10) and (11), and update the augmented Jacobian matrix using (21).

3. If *firstep* = false then

    4. Compute the predicted point $Z^{(0)}$ with the cubic predictor (9) based on *yold*, *ypold*, *y*, *yp*.

    else

    5. Compute the predicted point $Z^{(0)}$ using the linear predictor (8) based on *y* and *yp*.

6. If *failed* = true then

    7. Compute the augmented Jacobian matrix at $Z^{(0)}$.

    8. Compute the next iterate $Z^{(1)}$ using (19).

9. *limit* : = $2\Big(\lfloor -\log_{10}(arcae + arcre\|y\|)\rfloor + 1\Big)$. Repeat steps 10-11 until either

$$\left\|\Delta Z^{(k)}\right\| \leq arcae + arcre \left\|Z^{(k)}\right\|$$

or

limit iterations have been performed.

    10. Update the augmented Jacobian matrix using (22).

    11. Compute the next iterate using (19).

12. If the quasi-Newton iteration did not converge in *limit* steps, then

    13. h : = $h/2$; *failed* : = true.

    14. If h is unreasonably small, then return with an error flag.

15. Go to 3.

16. Compute the tangent at the accepted iterate $Z^{(*)}$ using (10) and (11), and update the augmented Jacobian matrix using (21).

17. Compute the angle α between the current and previous tangents by (24).

18. If $\alpha > \pi/3$, then

    19. $h := h/2$; $failed := \text{true}$.

    20. If h is unreasonably small, then return with an error flag.

    21. Go to 3.

22. $yold := y$, $ypold := yp$, $y := Z^{(*)}$, $yp :=$ tangent computed in step 16, $firststep :=$ false.

23. If $y_1 < 1$, then compute a new step size h by Equations (24–31) with $\xi_{min} = 0.01$,

$$\delta_k = \min\left\{(arcae + arcre\|y\|)^{1/4},\ \frac{1}{2}\|y - yold\|\right\},$$

    and go to 3.

24. Find $\bar{s}$ such that $p(\bar{s})_1 = 1$, using $yold$, $ypold$, y, and yp in (9). $yopp := yold$, $Z^{(0)} := p(\bar{s})$.

25. $limit := 2\Big(\lfloor -\log_{10}(ansae + ansre\|y\|)\rfloor + 1\Big)$. Do steps 26–33 for $k = 2, \ldots, limit + 2$.

    26. Update the augmented Jacobian matrix using (22).

    27. Take a quasi-Newton step with (36).

28. If

$$\left|P_1^{(k+1)} - 1\right| + \left\|\Delta Z^{(k-2)}\right\| \leq ansae + ansre \left\|Z^{(k-2)}\right\|,$$

then return (solution has been found).

29. If $\left|P_1^{(k+1)} - 1\right| \leq ansae + ansre$,

then

$$Z^{(k-1)} := P^{(k+1)}$$

else do steps 30-33.

30. $yold := y$, $y := P^{(k+1)}$.

31. If $yold_1$ and y_1 bracket $\lambda = 1$, then $yopp := yold$.

32. Compute $Z^{(k-1)}$ with the linear predictor (32) using y and $yold$.

33. If $\left\|Z^{(k-1)} - y\right\| > \|y - yopp\|$, then compute $Z^{(k-1)}$ with the linear predictor (33) using y and $yopp$.

34. Return with an error flag.

7. Augmented (sparse) Jacobian matrix algorithm. Large nonlinear systems of equations with sparse symmetric Jacobian matrices occur in many engineering disciplines, and each class of problems has special characteristics. The sparse algorithm here is designed to handle symmetric matrices with a "skyline" structure. This structure occurs frequently in structural mechanics and in many other engineering problems.

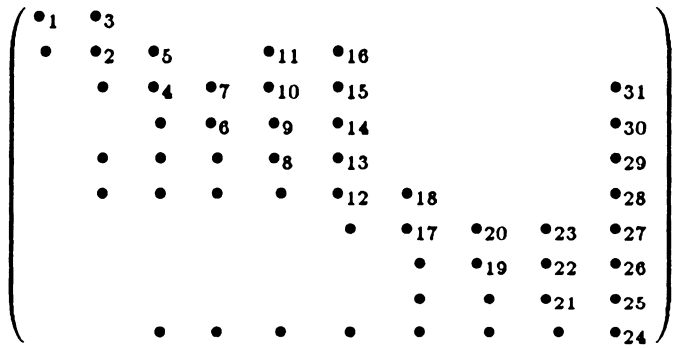
The sparse algorithm differs from the dense algorithm in three respects: (1) the low level numerical linear algebra is changed to take advantage of the sparsity of the problem; (2)

quasi-Newton iterations are abandoned in favor of pure Newton iterations; (3) Rheinboldt's step size control [11] is implemented more faithfully because of the use of Newton iterations. Except for these three changes, the logic for tracking the zero curve γ is exactly the same as for the dense algorithm.

For technical reasons it is preferable to write the homotopy map in (3) as

$$\rho_a(x, \lambda)$$

with the order of the arguments reversed (this is an internal matter to HOMPACK and causes no confusion at the user interface, since the user only specifies $F(x)$). The Jacobian matrix $D_x\rho_a(x, \lambda)$ is symmetric (because it is the Hessian matrix of some energy potential function) and sparse with a "skyline" structure, such as



Typically such matrices are stored in packed skyline format, in which the upper triangle is stored in a one-dimensional array indexed as shown above. The auxiliary array (1, 2, 4, 6, 8, 12, 17, 19, 21, 24, 32) of diagonal indices is also required. By convention the auxiliary integer array has length $n+1$ with the $(n+1)$st element containing the length of the packed real array plus one.

The general theory from the dense algorithm applies equally well here, so the zero curve

γ given by $(x(s), \lambda(s))$ is the trajectory of the initial value problem

$$\frac{d}{ds}\rho_a(x(s),\lambda(s)) = \left[D_x\rho_a(x(s),\lambda(s)), D_\lambda\rho_a(x(s),\lambda(s))\right] \begin{pmatrix} \frac{dx}{ds} \\ \frac{d\lambda}{ds} \end{pmatrix} = 0, \tag{37}$$

$$\left\| \left(\frac{dx}{ds}, \frac{d\lambda}{ds} \right) \right\|_2 = 1, \tag{38}$$

$$x(0) = a, \quad \lambda(0) = 0. \tag{39}$$

In solving any linear equation, care must now be taken in order to benefit from the sparse structure. For example, QR factorization is no longer appropriate because it causes fill-in of the matrix. To avoid this problem, a preconditioned conjugate gradient algorithm is used. The conjugate gradient algorithm will now be described.

Let $(\bar{x}, \bar{\lambda})$ be a point on the zero curve γ, and $\bar{y}$ the unit tangent vector to γ at $(\bar{x}, \bar{\lambda})$ in the direction of increasing arc length s. Then the matrix

$$A = \begin{bmatrix} D\rho_a(x,\lambda) \\ \bar{y} \end{bmatrix} \tag{40}$$

is invertible in a neighborhood of $(\bar{x}, \bar{\lambda})$ by continuity. Thus, the kernel of $D\rho_a$ can be found by solving the linear system of equations

$$Ay = e_{n+1} = b. \tag{41}$$

Given any nonsymmetric, nonsingular matrix A, the system of linear equations $Ay = b$ can be solved by considering the linear system

$$AA^t z = b.$$

Since the coefficient matrix for this system is both symmetric and positive definite, the system can be solved by a conjugate gradient algorithm. Once a solution vector z is obtained, the vector y from the original system can be computed as $y = A^t z$. An implementation of the conjugate gradient algorithm in which y is computed directly, without reference to z, any approximations of z, or AA^t, was originally proposed by Hestenes [7], and is commonly known as Craig's method [5]. Each iterate y^i minimizes the Euclidean error norm $\|y - y^i\|$ over the translated Krylov space

$$y^0 + span\{r^0, AA^t r^0, (AA^t)^2 r^0, \ldots, (AA^t)^{i-1} r^0\},$$

where $r^0 = b - Ay^0$. Below $\langle u, v \rangle$ denotes the inner product $u^t v$.

Craig's Method:

Choose y^0;

Compute $r^0 = b - Ay^0$;

Compute $p^0 = A^t r^0$;

For $i = 0$ step 1 until convergence do

BEGIN

$\alpha_i = \langle r^i, r^i \rangle / \langle p^i, p^i \rangle$

$y^{i+1} = y^i + \alpha_i p^i$

$r^{i+1} = r^i - \alpha_i A p^i$

$\beta_i = \langle r^{i+1}, r^{i+1} \rangle / \langle r^i, r^i \rangle$

$p^{i+1} = A^t r^{i+1} + \beta_i p^i$

END

Let Q be any nonsingular matrix. The solution to the system $Ay = b$ can be calculated by solving the system

$$By = (Q^{-1}A)y = Q^{-1}b = g. \tag{42}$$

The use of such a matrix is known as preconditioning. Since the goal of using preconditioning is to decrease the computational effort needed to solve the original system, Q should be some approximation to A. Then $Q^{-1}A$ would be close to the identity matrix, and the iterative method described above would converge more rapidly when applied to (42) than when applied to (41). In the following algorithm B and g are never explicitly formed. The algorithm given above can be obtained by substituting the identity matrix for Q.

Craig's method using a preconditioner:

Choose y^0, Q;

Compute $r^0 = b - Ay^0$;

Compute $\tilde{r}^0 = Q^{-1}r^0$;

Compute $p^0 = A^t Q^{-t}\tilde{r}^0$;

For $i = 0$ step 1 until convergence do

BEGIN

$\alpha_i = \langle \tilde{r}^i, \tilde{r}^i \rangle / \langle p^i, p^i \rangle$

$y^{i+1} = y^i + \alpha_i p^i$

$\tilde{r}^{i+1} = \tilde{r}^i - \alpha_i Q^{-1} A p^i$

$\beta_i = \langle \tilde{r}^{i+1}, \tilde{r}^{i+1} \rangle / \langle \tilde{r}^i, \tilde{r}^i \rangle$

$p^{i+1} = A^t Q^{-t} \tilde{r}^{i+1} + \beta_i p^i$

END

For this algorithm, a minimum of $5(n+1)$ storage locations is required (in addition to that for A). The vectors y, $\tilde{r}$, and p all require their own locations; $Q^{-t}\tilde{r}$ can share with Ap; $Q^{-1}Ap$ can share with $A^tQ^{-t}\tilde{r}$. The computational cost per iteration of this algorithm is:

1) two preconditioning solves ($Q^{-1}v$ and $Q^{-t}v$);

2) two matrix-vector products (Av and A^tv);

3) $5(n+1)$ multiplications (the inner products $\langle p, p \rangle$ and $\langle \tilde{r}, \tilde{r} \rangle$, αp, βp, and $\alpha Q^{-1} Ap$).

The coefficient matrix A in the linear system of equations (41) has a very special structure which can be exploited if (41) is attacked indirectly as follows. Note that the leading $n \times n$ submatrix of A is $D_x \rho_a$, which is symmetric and sparse, but possibly indefinite. Write

$$A = M + L \tag{43}$$

where

$$M = \begin{bmatrix} D_x \rho_a(\bar{x}, \bar{\lambda}) & c \\ c^t & d \end{bmatrix},$$

$$L = u e_{n+1}^t, \quad u = \begin{pmatrix} D_\lambda \rho_a(\bar{x}, \bar{\lambda}) - c \\ 0 \end{pmatrix}.$$

Using the Sherman-Morrison formula (L is rank one), the solution y to the original system $Ay = b$ can be obtained from

$$y = \left[I - \frac{M^{-1} u e_{n+1}^t}{(M^{-1}u)^t e_{n+1} + 1} \right] M^{-1} b. \tag{44}$$

which requires the solution of two linear systems with the sparse, symmetric, invertible matrix M. It is the systems $Mz = u$ and $Mz = b$ to which Craig's preconditioned conjugate gradient algorithm is actually applied.

The only remaining detail is the choice of the preconditioning matrix Q. Q is taken as the modified Cholesky decomposition of M, as described by Gill and Murray [6]. If M is positive definite and well conditioned, $Q = M$. Otherwise, Q is a well conditioned positive definite approximation to M. The use of a positive definite Q is reasonable since in the context of structural mechanics $DF(x)$ is positive definite or differs from a positive definite matrix by a low rank perturbation. The Gill-Murray factorization algorithm can exploit the symmetry and sparse skyline structure of M, and this entire scheme, Equations (41-44), is built around using the symmetry and sparse skyline structure of the Jacobian matrix $D_x \rho_a$.

In addition to linear algebra changes, the sparse algorithm differs from the dense algorithm by using Newton rather than quasi-Newton iterations. The use of Newton iterations is necessitated by the current lack of a good (comparable to Broyden or BFGS) sparse quasi-Newton update formula. The fill-in produced by a good (dense) update formula is unacceptable, and the efficacy of deferred updating [8] is questionable (the number of applications of the Sherman-Morrison formula grows exponentially with the number of deferred updates). Also there is some evidence that, at least in the context of structural mechanics

[19], a model trust region strategy with exact (expensive) Jacobian matrix evaluations is better than (cheap) quasi-Newton updating.

The final change for the sparse matrix algorithm is an enhancement to the step size control, allowed by the use of Newton iterations. The enhancement involves implementing a more sophisticated control over the ideal starting error δ_k used in Equation (28). Computing this error involves using the exact error $\tilde{\delta}_k^{(0)}$ of the last predicted point, the size of the last Newton step $\tilde{\delta}_k^{(*)}$, the ideal error δ_{k-1} from the previous step, and the number of iterations i_* required by the correction process.

$$\delta_k = \theta\, \delta_{k-1} \tag{45}$$

where θ is a function of $\tilde{\delta}_k^{(*)}$, $\tilde{\delta}_k^{(0)}$, and i_* as described by Rheinboldt [11].

The goal behind these calculations is to keep the number of corrector iterations fixed at four. Thus θ is computed to adjust the ideal error for the next step. If, for example, i_* was less than four, θ will be large so that the next step will take more iterations. The scheme is slightly different than the one used by Rheinboldt. His algorithm used the formula

$$\delta_k = \theta\, \tilde{\delta}_k^{(0)} \tag{46}$$

instead of (45). However, numerical experiments indicate that the desired behavior (convergence in four corrector iterations) is obtained by using (45).

8. Testing. The augmented Jacobian matrix algorithm (FIXPQF and FIXPQS) has been tested on several nonlinear problems. The performance of the algorithm was tested relative to the other two HOMPACK algorithms (FIXPDF and FIXPNF). In addition, the algorithm was used to solve several tough polynomial problems.

Table 1 shows some results for Brown's function, which has an ill conditioned Jacobian matrix, and an exponential function, whose zero curve γ has several sharp turns. Brown's function is

$$f_1(x) = \prod_{i=1}^{n} x_i - 1$$

$$f_k(x) = x_k + \sum_{i=1}^{n} x_i - (n+1), \quad k = 2, \ldots, n.$$

The exponential function is

$$f_k(x) = x_k - \exp\left(\cos\left(k \sum_{i=1}^{n} x_i\right)\right), \quad k = 1, \ldots, n.$$

The solutions were found from a starting point of 0 with a relative error of 10^{-10}, and the CPU times are for a VAX 11/785. NFE is the number of Jacobian evaluations. The runs were made with the largest tracking tolerance which could successfully track the zero curve. The magnitude of this tracking tolerance in powers of 10 is represented in parentheses in the NFE column.

FIXPQF was used to solve several tough polynomial problems. All of these problems were solved using a polynomial driver program POLSYS which is available in HOMPACK. This driver created the homotopy map and then called FIXPQF. The following three problems come from Alexander Morgan.

Table 1. Numerical results.

	FIXPDF		FIXPNF		FIXPQF		
n	NFE	TIME	NFE	TIME	NFE	TIME	arc length
			Brown's function				
5	87(-3)	.71	17(-2)	.19	9(-2)	.59	2.7
10	85(-2)	2.12	24(-2)	.61	8(-2)	1.54	3.7
15	102(-2)	5.64	23(-2)	1.39	11(-2)	4.46	4.4
20	98(-4)	10.44	22(-2)	2.44	9(-2)	5.56	5.1
25	123(-3)	22.83	29(-2)	5.39	11(-2)	11.27	5.7
30	96(-3)	27.99	23(-2)	6.62	11(-2)	15.46	6.2
35	110(-4)	48.54	28(-2)	11.72	12(-2)	25.41	6.6
40	110(-4)	68.54	26(-2)	15.85	11(-4)	30.99	7.1
45	128(-4)	105.32	30(-3)	24.73	13(-2)	48.01	7.5
50	113(-4)	125.48	29(-2)	32.99	11(-2)	45.18	7.8
			Exponential function				
2	70(-4)	.27	12(-2)	.07	5(-2)	.14	1.6
3	270(-5)	1.42	39(-2)	.31	26(-2)	1.18	5.1
4	280(-4)	2.03	75(-2)	.87	37(-3)	2.96	6.5
5	486(-4)	4.73	213(-6)	3.38	62(-3)	7.06	14.5
6	817(-5)	10.20	293(-8)	6.16	70(-3)	9.92	16.9
7	1517(-6)	24.98	433(-8)	11.73	105(-3)	18.49	24.0
8	2931(-7)	60.50	577(-8)	20.73	162(-4)	36.65	47.6
9	4511(-8)	109.82	824(-8)	37.44	206(-4)	54.11	61.8
10	5671(-8)	165.32	1001(-9)	53.80	268(-4)	79.45	85.8

Problem PB000402:

$$f_1(x) = (-.2292 * 10^{-3})x_1^2 + (.2393 * 10^{-14}) * x_2^2 + (-.2735 * 10^2)x_1x_2$$
$$+ (-.5537 * 10^4)x_1 + (.277 * 10^7)x_2 + (.1425 * 10^2)$$
$$f_2(x) = (-.7194 * 10^4)x_1^2 + (.2393 * 10^{-14})x_2^2 + (-.2735 * 10^2)x_1x_2$$
$$+ (.5537 * 10^4)x_1 + (-.277 * 10^7)x_2 + (.1418 * 10^2).$$

Problem PB000403:

$$f_1(x) = (-.98 * 10^{-3})x_1^2 + (.978 * 10^6)x_2^2 + (-9.8)x_1x_2$$
$$+ (-.235 * 10^3)x_1 + (.88900 * 10^5)x_2 + (-1.0)$$
$$f_2(x) = (-.1 * 10^{-1})x_1^2 + (-.984)x_2^2 + (-.297 * 10^2)x_1x_2$$
$$+ (.987 * 10^{-2})x_1 + (-.124)x_2 + (-.25).$$

Problem PB000601:

$$f_1(x) = (-.625 * 10^{14})x_1^2x_3 + (.53835 * 10^9)x_2^6 + (.503135 * 10^9)x_2^5$$
$$+ (.895258 * 10^8)x_2^4 + (.577586 * 10^7)x_2^3 + (.107358 * 10^6)x_2^2$$
$$+ (.617 * 10^3)x_2 + 1.0$$
$$f_2(x) = (.625 * 10^{14})x_1^2x_2 + (.1875 * 10^{15})x_1^2x_3 + (.2025 * 10^8)x_1x_2$$
$$+ (-.503135 * 10^9)x_2^5 - (.179052 * 10^9)x_2^4 + (-.173276)x_2^3$$
$$+ (-.429432 * 10^6)x_2^2 + (-.3085 * 10^4)x_2 - 6.0$$
$$f_3(x) = (-.555555 * 10^{16})x_1^2 + (.111111 * 10^{17})x_1x_3 + (.18 * 10^{10})x_2 + 1.0.$$

Morgan's three problems were solved with a tracking tolerance of 10^{-6}. Problems PB000402 and PB000403 both have 4 complex roots, all of which were found. Problem PB000601 has 60 complex roots. The algorithm was successful at finding 55 of these roots.

In addition to Morgan's problems, POLSYS was also used to solve a test problem presented by Dennis, Gay, and Vu [3]. This problem is

$$a + b = \Sigma M_x$$

$$c + d = \Sigma M_y$$

$$ta + ub - vc - wd = \Sigma A$$

$$va + wb + tc + ud = \Sigma B$$

$$a(t^2 - v^2) - 2ctv + b(u^2 - w^2) - 2duw = \Sigma C$$

$$c(t^2 - v^2) + 2atv + d(u^2 - w^2) - 2duw = \Sigma D$$

$$at(t^2 - 3v^2) + cv(v^2 - 3t^2) + bu(u^2 - 3w^2) + dw(w^2 - 3u^2) = \Sigma E$$

$$ct(t^2 - 3v^2) - av(v^2 - 3t^2) + du(u^2 - 3w^2) - bw(w^2 - 3u^2) = \Sigma F,$$

where the right-hand sides of the equations represent measured quantities. Thus, by using different right-hand sides, different problems are specified.

POLSYS was used to solve four different versions of the above problem, defined as follows:

$$\text{Let } \Sigma = (\Sigma M_x, \Sigma M_y, \Sigma A, \Sigma B, \Sigma C, \Sigma D, \Sigma E, \Sigma F).$$

Experiment 791129

$$\Sigma = (.485, -.0019, -.0581, .015, .105, .0406, .167, -.399)$$

Experiment 0121a

$$\Sigma = (-.816, -.017, -1.826, -.754, -4.839, -3.259, -14.023, 15.467)$$

Experiment 0121b

$$\Sigma = (-.809, -.021, -2.04, -.614, -6.903, -2.934, -26.328, 18.639)$$

Experiment 0121c

$$\Sigma = (-.807, -.021, -2.379, -.364, -10.541, -1.961, -51.551, 21.053).$$

Each of these problems has 576 complex roots, however only the real roots are significant. Table 2 shows the results for these experiments, indicating the number of real solutions found, and the number of Jacobian evaluations needed to find these roots.

The sparse algorithm (FIXPQS) was used to solve the following two problems:

$$f_1(x) = x_1^3 + 6x_2x_3 + x_1 - 1$$

$$f_2(x) = 6x_1x_3 + x_2^4x_5 + 3x_2 - 1$$

$$f_3(x) = 6x_1x_2 + x_3x_5 + 4x_3 - 1$$

$$f_4(x) = x_4^3x_8 + 2x_4 - 1$$

$$f_5(x) = .2x_2^5 + .5x_3 + x_8x_5 + 3x_5 - 1$$

$$f_6(x) = x_6x_8 + 4x_6 - 1$$

$$f_7(x) = x_7^2x_8^3 + 2x_7 - 1$$

$$f_8(x) = x_4^4 + .5x_5^2 + .5x_6^2 + x_7^3x_8^2 + 3x_8 - 1.$$

and

$$f_k(x) = k\cos(k(x_k - k))\exp(\sin(k(x_k - k))) \qquad k = 1 \ldots n.$$

The polynomial problem was solved with a tracking tolerance of 10^{-4}, and required 33 Jacobian evaluations. Table 3 shows the results for the exponential problem. Both problems were run with a starting point of 0.

Table 2. Dennis' problem.

experiment	solutions found	range of Jacobian evaluations
791129	28	26–86
0121a	18	32–98
0121b	6	32–59
0121c	23	30–101

Table 3. Sparse exponential problem.

n	NFE	TIME	arc length
1	18	.33	1.177
2	26	.59	1.738
3	28	.81	1.786
4	32	1.23	1.892
5	32	1.45	1.948

9. Conclusions.

The augmented Jacobian matrix algorithm represents a valuable addition to the HOMPACK library. Each of the HOMPACK algorithms is superior to the other two on certain problems. Thus, having three different algorithms enables the user to pick the one most equipped to solve his particular problem. The following is a description of the augmented Jacobian matrix algorithm's strengths and weaknesses. In addition to these strengths and weaknesses, a description of further research areas is presented.

FIXPQF is superior on problems with expensive Jacobian evaluations. The use of quasi-Newton iterations greatly decreases the amount of Jacobian evaluations. The test results demonstrate this fact. In contrast, when Jacobian evaluations are cheap, FIXPQF is less efficient than FIXPNF which uses Newton iterations. This makes sense because quasi-Newton iterations take longer to converge.

Another strength of the augmented Jacobian matrix algorithm is its ability to track tightly turning curves. The requirement that two consecutive tangents must make an angle no greater than 60° allows the algorithm to detect when questionable steps are made. This makes FIXPQF superior to the other algorithms in keeping with tightly turning curves.

The augmented Jacobian matrix algorithm is poor at tracking curves with ill-conditioned or badly scaled Jacobian matrices. The reason for this is that the radius of convergence of the quasi-Newton correction process is small compared to the Newton process. Thus, for ill-conditioned problems, smaller steps have to be made in order to keep with the curve.

The augmented Jacobian matrix algorithm is by no means optimal. In developing the

algorithm, several questions arose which represent areas for further research. The following is a list of the research problems.

1. The quasi-Newton algorithm converges most rapidly when the initial point is close to the solution. Because of this fact, it may be more efficient to start the quasi-Newton correction process with an exact Jacobian at the predictor point $Z^{(0)}$. The current implementation uses an approximate Jacobian matrix derived from a Broyden update of the matrix at the previous point $P^{(2)}$. The argument for this change is that $Z^{(0)}$ is much closer to the next point on the curve than $P^{(2)}$ is. Thus, the increased rate of convergence may justify the additional Jacobian evaluation.

2. What is the optimal limit for the number of quasi-Newton iterations allowed in the corrector process before admitting failure? This number should not be too small because it would cause many unnecessary failures. However, if the predictor step is so large that the process converges extremely slowly or not at all, this problem should be detected as soon as possible. Thus, the limit on the number of iterations should not be too large. It may be possible to derive a function which estimates the probability of converging on the next step given the number of steps already taken, the size of the last step, and the desired accuracy. By using this function, it is possible that a function for the optimal limit could be derived.

3. What is the maximum angle that two consecutive tangent vectors should make in order to safely track the zero curve? Currently a 60° angle is used as a maximum. Perhaps some other angle is better, or maybe the maximum angle should be computed dynamically.

4. The curvature estimate for computing the step-size is terrible for tightly turning curves. Other schemes should be explored.

5. The strategy for computing the ideal error in the step-size estimation phase could be improved. Two issues need to be considered. First, for the dense algorithm, a better relation between the ideal error and the number of iterations required needs to be devised. Second, the ideal error computation could take into account the cost of Jacobian evaluations. If Jacobian evaluations are extremely difficult, the ideal error should be increased so that larger steps are taken (at the cost of many more corrector iterations) so that fewer Jacobian evaluations are necessary.

6. The formula for computing the step-size from the curvature and the ideal error is based on a linear predictor. However, since the predictor step usually uses a cubic polynomial, the possibility of developing a formula based on a cubic predictor should be explored.

7. One difficulty homotopy algorithms have occurs when they are trying to find a multiple root. In this case, the zero curve does not cross $\lambda = 1$. This makes finding the exact solution very difficult, and the algorithms often fail. A way of handling this case needs to be developed.

8. The use of a quasi-Newton algorithm with deferred updating [8] may be a plausible alternative to Newton's method for the sparse algorithm. This alternative needs to be explored.

References

[1] P. BUSINGER AND G. G. GOLUB, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.

[2] S. N. CHOW, J. MALLET-PARET, AND J. A. YORKE, *Finding zeros of maps: Homotopy methods that are constructive with probability one*, Math. Comput., 32 (1978), pp. 887–899.

[3] J. E. DENNIS, D. M. GAY, AND P. A. VU, *A new nonlinear equations test problem*, Tech. Report 83-16, Dept. of Mathematical Sciences, Rice University, Houston, Texas, 1983.

[4] J. E. DENNIS AND R. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[5] D. K. FADEEV AND V. N. FADEEVA, *Computational Methods of Linear Algebra*, Freeman, London, 1963.

[6] P. E. GILL AND W. MURRAY, *Newton type methods for unconstrained and linearly constrained optimization*, Math. Programming, 7 (1974), pp. 311–350.

[7] M. R. HESTENES, *The conjugate gradient method for solving linear systems*, in Proc. Symp. Appl. Math., 6, Numer. Anal., AMS, New York, 83–102, 1956.

[8] H. MATTHIES AND G. STRANG, *The solution of nonlinear finite element equations*, Internat. J. Numer. Meth. Engrg., 14(1979).

[9] A. P. MORGAN, *A transformation to avoid solutions at infinity for polynomial systems*, Appl. Math. Comput., to appear.

[10] ——, *A homotopy for solving polynomial systems*, Appl. Math. Comput., to appear.

[11] W. C. RHEINBOLDT AND J. V. BURKARDT, *Algorithm 596: A program for a locally parameterized continuation process*, ACM Trans. Math. Software, 9 (1983), pp. 236-241.

[12] L. T. WATSON AND D. FENNER, *Chow-Yorke algorithm for fixed points or zeros of C^2 maps*, ACM Trans. Math. Software, 6 (1980), pp. 252–260.

[13] L. T. WATSON, *A globally convergent algorithm for computing fixed points of C^2 maps*, Appl. Math. Comput., 5 (1979), pp. 297–311.

[14] ——, *Computational experience with the Chow-Yorke algorithm*, Math. Programming, 19 (1980), pp. 92–101.

[15] ——, *Fixed points of C^2 maps*, J. Comput. Appl. Math., 5 (1979), pp. 131 140.

[16] ——, *Solving the nonlinear complementarity problem by a homotopy method*, SIAM J. Control Optim., 17 (1979), pp. 36–46.

[17] ——, *An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems*, SIAM J. Numer. Anal., 16 (1979), pp. 394-401.

[18] ——, *Solving finite difference approximations to nonlinear two-point boundary value problems by a homotopy method*, SIAM J. Sci. Stat. Comput., 1 (1980), pp. 467-480.

[19] ——, *Engineering applications of the Chow-Yorke algorithm*, Appl. Math. Comput., 9 (1981), pp. 111-133.

[20] L. T. WATSON, M. P. KAMAT, AND M. H. REASER, *A robust hybrid algorithm for computing multiple equilibrium solutions*, Engrg. Comput., to appear.

[21] L. T. WATSON AND M. R. SCOTT, *Solving spline collocation approximations to nonlinear two-point boundary value problems by a homotopy method*, Tech. Report TR-84-15, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1984.

[22] L. T. WATSON, *Numerical linear algebra aspects of globally convergent homotopy methods*, Tech. Report TR-85-14, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1985.

Appendix A Dense Algorithm Code Listing

```
      SUBROUTINE FIXPQF(N,NFE,IFLAG,ARCRE,ARCAE,ANSRE,ANSAE,ARCLEN,A,Y,
     &     YP,YOLD,YPOLD,QT,R,FO,F1,ZO,DZ,W,T,YSAV,SSPAR,PAR,IPAR)
C
C SUBROUTINE  FIXPQF  FINDS A FIXED POINT OR ZERO OF THE
C N-DIMENSIONAL VECTOR FUNCTION  F(X), OR TRACKS A ZERO CURVE OF A
C GENERAL HOMOTOPY MAP  RHO(A,LAMBDA,X).  FOR THE FIXED POINT PROBLEM
C F(X) IS ASSUMED TO BE A C2 MAP OF SOME BALL INTO ITSELF.  THE
C EQUATION  X=F(X)  IS SOLVED BY FOLLOWING THE ZERO CURVE OF THE
C HOMOTOPY MAP
C
C  LAMBDA*(X - F(X)) + (1 - LAMBDA)*(X - A),
C
C STARTING FROM  LAMBDA = 0, X = A.   THE CURVE IS PARAMETERIZED
C BY ARC LENGTH  S, AND IS FOLLOWED BY SOLVING THE ORDINARY
C DIFFERENTIAL EQUATION  D(HOMOTOPY MAP)/DS = 0  FOR
C Y(S) = (LAMBDA(S), X(S)).  THIS IS DONE BY USING A HERMITE CUBIC
C PREDICTOR AND A CORRECTOR WHICH RETURNS TO THE ZERO CURVE IN A
C HYPERPLANE PERPENDICULAR TO THE TANGENT TO THE ZERO CURVE AT THE
C MOST RECENT POINT.
C
C FOR THE ZERO FINDING PROBLEM  F(X)  IS ASSUMED TO BE A C2 MAP
C SUCH THAT FOR SOME  R > 0,  X*F(X) >= 0  WHENEVER  NORM(X) = R.
C THE EQUATION  F(X) = 0  IS SOLVED BY FOLLOWING THE ZERO CURVE OF
C THE HOMOTOPY MAP
C
C  LAMBDA*F(X) + (1 - LAMBDA)*(X - A)
C
C EMANATING FROM  LAMBDA = 0, X = A.
C
C A  MUST BE AN INTERIOR POINT OF THE ABOVE MENTIONED BALLS.
C
C FOR THE CURVE TRACKING PROBLEM RHO(A,LAMBDA,X) IS ASSUMED TO
C BE A C2 MAP FROM  E**M X [0,1) X E**N  INTO  E**N, WHICH FOR
C ALMOST ALL PARAMETER VECTORS  A  IN SOME NONEMPTY OPEN SUBSET
C OF E**M SATISFIES
C
C  RANK [D RHO(A,LAMBDA,X)/D LAMBDA, D RHO(A,LAMBDA,X)/DX] = N
C
C FOR ALL POINTS  (LAMBDA,X)  SUCH THAT  RHO(A,LAMBDA,X) = 0.  IT IS
C FURTHER ASSUMED THAT
C
C         RANK [ D RHO(A,0,XO)/DX ] = N.
C
C WITH  A  FIXED, THE ZERO CURVE OF  RHO(A,LAMBDA,X)  EMANATING FROM
C LAMBDA = 0, X = XO  IS TRACKED UNTIL  LAMBDA = 1  BY SOLVING THE
C ORDINARY DIFFERENTIAL EQUATION    D RHO(A,LAMBDA(S),X(S))/DS = 0
C FOR  Y(S) = (LAMBDA(S), X(S)), WHERE  S  IS ARC LENGTH ALONG THE
C ZERO CURVE.  ALSO THE HOMOTOPY MAP  RHO(A,LAMBDA,X)  IS ASSUMED TO
```

```
C BE CONSTRUCTED SUCH THAT
C
C         D LAMBDA(0)/DS > 0.
C
C FOR THE FIXED POINT AND ZERO FINDING PROBLEMS, THE USER MUST SUPPLY
C A SUBROUTINE  F(X,V)  WHICH EVALUATES  F(X)  AT  X  AND RETURNS THE
C VECTOR F(X) IN  V, AND A SUBROUTINE  FJAC(X,V,K)  WHICH RETURNS IN  V
C THE KTH COLUMN OF THE JACOBIAN MATRIX OF F(X) EVALUATED AT X.  FOR
C THE CURVE TRACKING PROBLEM, THE USER MUST SUPPLY A SUBROUTINE
C RHO(A,LAMBDA,X,V,PAR,IPAR)  WHICH EVALUATES THE HOMOTOPY MAP  RHO AT
C (A,LAMBDA,X)  AND RETURNS THE VECTOR  RHO(A,LAMBDA,X)  IN  V, AND
C A SUBROUTINE  RHOJAC(A,LAMBDA,X,V,K,PAR,IPAR)  WHICH RETURNS IN  V
C THE KTH COLUMN OF THE  N X (N+1)  JACOBIAN MATRIX
C [D RHO/D LAMBDA, D RHO/DX]  EVALUATED AT  (A,LAMBDA,X).  FIXPQF
C DIRECTLY OR INDIRECTLY USES THE SUBROUTINES  D1MACH, F (OR RHO),
C FJAC (OR RHOJAC), QRFAQF, QRSLQF, ROOT, ROOTQF, STEPQF, TANGQF,
C UPQRQF  AND THE BLAS ROUTINES  DAXPY, DCOPY, DDOT, DNRM2, AND  DSCAL.
C ONLY  D1MACH  CONTAINS MACHINE DEPENDENT CONSTANTS.  NO OTHER
C MODIFICATIONS BY THE USER ARE REQUIRED.
C
C
C ON INPUT:
C
C N  IS THE DIMENSION OF X, F(X), AND RHO(A,LAMBDA,X).
C
C IFLAG CAN BE -2, -1, 0, 2, OR 3.  IFLAG SHOULD BE 0 ON THE FIRST
C    CALL TO  FIXPQF  FOR THE PROBLEM  X=F(X), -1 FOR THE PROBLEM
C    F(X)=0, AND -2 FOR THE PROBLEM  RHO(A,LAMBDA,X)=0.    IN CERTAIN
C    SITUATIONS  IFLAG  IS SET TO 2 OR 3 BY  FIXPQF, AND  FIXPQF  CAN
C    BE CALLED AGAIN WITHOUT CHANGING  IFLAG.
C
C ARCRE, ARCAE  ARE THE RELATIVE AND ABSOLUTE ERRORS, RESPECTIVELY,
C    ALLOWED THE QUASI-NEWTON ITERATION ALONG THE ZERO CURVE.  IF
C    ARC?E .LE. 0.0  ON INPUT, IT IS RESET TO  .5*SQRT(ANS?E).
C    NORMALLY  ARC?E  SHOULD BE CONSIDERABLY LARGER THAN  ANS?E.
C
C ANSRE, ANSAE  ARE THE RELATIVE AND ABSOLUTE ERROR VALUES USED FOR
C    THE ANSWER AT  LAMBDA = 1.  THE ACCEPTED ANSWER  Y = (LAMBDA, X)
C    SATISFIES
C
C      |Y(1) - 1| .LE. ANSRE * ANSAE       .AND.
C
C      ||DZ|| .LE. ANSRE*||Y|| + ANSAE      WHERE
C
C      DZ IS THE QUASI-NEWTON STEP TO Y.
C
C A(1:*)  CONTAINS THE PARAMETER VECTOR  A.  FOR THE FIXED POINT
C    AND ZERO FINDING PROBLEMS,  A  NEED NOT BE INITIALIZED BY THE
C    USER, AND IS ASSUMED TO HAVE LENGTH  N.  FOR THE CURVE
C    TRACKING PROBLEM,  A  MUST BE INITIALIZED BY THE USER.
```

```
C
C Y(1:N+1)  CONTAINS THE STARTING POINT FOR TRACKING THE HOMOTOPY MAP.
C    (Y(2),...,Y(N+1)) = A  FOR THE FIXED POINT AND ZERO FINDING
C    PROBLEMS.  (Y(2),...,Y(N+1)) = XO  FOR THE CURVE TRACKING PROBLEM.
C    Y(1)  NEED NOT BE DEFINED BY THE USER.
C
C YP(1:N+1)  IS A WORK ARRAY CONTAINING THE TANGENT VECTOR TO THE
C    ZERO CURVE AT THE CURRENT POINT  Y.
C
C YOLD(1:N+1) IS A WORK ARRAY CONTAINING THE PREVIOUS POINT FOUND
C    ON THE ZERO CURVE.
C
C YPOLD(1:N+1) IS A WORK ARRAY CONTAINING THE TANGENT VECTOR TO
C    THE ZERO CURVE AT  YOLD.
C
C QT(1:N+1,1:N+1), R((N+1)*(N+2)/2), FO(1:N+1), F1(1:N+1), ZO(1:N+1),
C    DZ(1:N+1), W(1:N+1), T(1:N+1), YSAV(1:N+1)  ARE ALL WORK ARRAYS
C    USED BY  STEPQF, TANGQF AND ROOTQF TO CALCULATE THE TANGENT
C    VECTORS AND QUASI-NEWTON STEPS.
C
C SSPAR(1:4) =  (HMIN, HMAX, BMIN, BMAX)  IS A VECTOR OF PARAMETERS
C    USED FOR THE OPTIMAL STEP SIZE ESTIMATION.  A DEFAULT VALUE
C    CAN BE SPECIFIED FOR ANY OF THESE FOUR PARAMETERS BY SETTING IT
C    .LE. 0.0  ON INPUT.  SEE THE COMMENTS IN  STEPQF  FOR MORE
C    INFORMATION ABOUT THESE PARAMETERS.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C N, A  ARE UNCHANGED.
C
C NFE  IS THE NUMBER OF JACOBIAN EVALUATIONS.
C
C IFLAG =
C
C   1   NORMAL RETURN
C
C   2   SPECIFIED ERROR TOLERANCE CANNOT BE MET.  SOME OR ALL OF
C       ARCRE, ARCAE, ANSRE, ANSAE  HAVE BEEN INCREASED TO
C       SUITABLE VALUES.  TO CONTINUE, JUST CALL  FIXPQF  AGAIN
C       WITHOUT CHANGING ANY PARAMETERS.
C
C   3   STEPQF  HAS BEEN CALLED 1000 TIMES.  TO CONTINUE, CALL
C       FIXPQF  AGAIN WITHOUT CHANGING ANY PARAMETERS.
C
C   4   JACOBIAN MATRIX DOES NOT HAVE FULL RANK.  THE ALGORITHM
```

```
C       HAS FAILED (THE ZERO CURVE OF THE HOMOTOPY MAP CANNOT BE
C       FOLLOWED ANY FURTHER).
C
C   5   THE TRACKING ALGORITHM HAS LOST THE ZERO CURVE OF THE
C       HOMOTOPY MAP AND IS NOT MAKING PROGRESS.  THE ERROR
C       TOLERANCES  ARC?E  AND  ANS?E  WERE TOO LENIENT.  THE PROBLEM
C       SHOULD BE RESTRARTED BY CALLING  FIXPQF  WITH SMALLER ERROR
C       TOLERANCES AND  IFLAG = 0 (-1, -2).
C
C   6   THE QUASI-NEWTON ITERATION IN  STEPQF  OR  ROOTQF  FAILED TO
C       CONVERGE.  THE ERROR TOLERANCES  ANS?E  MAY BE TOO STRINGENT.
C
C   7   ILLEGAL INPUT PARAMETERS, A FATAL ERROR.
C
C ARCRE, ARCAE, ANSRE, ANSAE  ARE UNCHANGED AFTER A NORMAL RETURN
C    (IFLAG = 1).  THEY ARE INCREASED TO APPROPRIATE VALUES ON THE
C    RETURN  IFLAG = 2.
C
C ARCLEN  IS THE APPROXIMATE LENGTH OF THE ZERO CURVE.
C
C Y(1) = LAMBDA, (Y(2),...,Y(N+1)) = X, AND  Y  IS AN APPROXIMATE
C    ZERO OF THE HOMOTOPY MAP.  NORMALLY  LAMBDA = 1  AND  X  IS A
C    FIXED POINT OR ZERO OF  F(X).   IN ABNORMAL SITUATIONS,  LAMBDA
C    MAY ONLY BE NEAR 1 AND  X  NEAR A FIXED POINT OR ZERO.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION D1MACH, DNRM2
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION ABSERR, H, HOLD, RELERR, S, WK
        INTEGER IFLAGC, ITER, JW, LIMITD, LIMIT, NP1
        LOGICAL CRASH, START
C
C     SCALAR ARGUMENTS
C
        DOUBLE PRECISION ARCRE, ARCAE, ANSRE, ANSAE, ARCLEN
        INTEGER N, NFE, IFLAG
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION A(N), Y(N+1), YP(N+1), YOLD(N+1), YPOLD(N+1),
     &    QT(N+1,N+1), R((N+1)*(N+2)/2), FO(N+1), F1(N+1), ZO(N+1),
     &    DZ(N+1), W(N+1), T(N+1), YSAV(N+1), SSPAR(4), PAR(1)
        INTEGER IPAR(1)
C
        SAVE
```

```
C
C ***** END OF DECLARATIONS *****
C
C LIMITD IS AN UPPER BOUND ON THE NUMBER OF STEPS.  IT MAY BE
C CHANGED BY CHANGING THE FOLLOWING PARAMETER STATEMENT:
      PARAMETER (LIMITD =1000)
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C CHECK IFLAG
C
      IF (N .LE. 0 .OR. ANSRE .LE. 0.0 .OR. ANSAE .LT. 0.0)
     &   IFLAG = 7
      IF (IFLAG .GE. -2 .AND. IFLAG .LE. 0) GO TO 10
      IF (IFLAG .EQ. 2) GO TO 50
      IF (IFLAG .EQ. 3) GO TO 40
C
C ONLY VALID INPUT FOR IFLAG IS -2, -1, 0, 2, 3.
C
      IFLAG = 7
      RETURN
C
C ***** INITIALIZATION BLOCK  *****
C
 10   ARCLEN = 0.0
      IF (ARCRE .LE. 0.0) ARCRE = .5*SQRT(ANSRE)
      IF (ARCAE .LE. 0.0) ARCAE = .5*SQRT(ANSAE)
      NFE=0
      IFLAGC = IFLAG
      NP1=N+1
C
C SET INITIAL CONDITIONS FOR FIRST CALL TO STEPQF.
C
      START=.TRUE.
      CRASH=.FALSE.
      RELERR = ARCRE
      ABSERR = ARCAE
      HOLD=1.0
      H=0.1
      S=0.0
      YPOLD(1) = 1.0
      Y(1) = 0.0
      DO 20 JW=2,NP1
        YPOLD(JW)=0.0
 20   CONTINUE
C
C SET OPTIMAL STEP SIZE ESTIMATION PARAMETERS.
C
C     MINIMUM STEP SIZE HMIN
      IF (SSPAR(1) .LE. 0.0) SSPAR(1)= (SQRT(N+1.0)+4.0)*D1MACH(4)
```

```
C     MAXIMUM STEP SIZE HMAX
        IF (SSPAR(2) .LE. 0.0) SSPAR(2)= 1.0
C     MINIMUM STEP REDUCTION FACTOR BMIN
        IF (SSPAR(3) .LE. 0.0) SSPAR(3)= 0.1
C     MAXIMUM STEP EXPANSION FACTOR BMAX
        IF (SSPAR(4) .LE. 0.0) SSPAR(4)= 7.0
C
C LOAD  A  FOR THE FIXED POINT AND ZERO FINDING PROBLEMS.
C
        IF (IFLAGC .GE. -1) THEN
          CALL DCOPY(N,Y(2),1,A,1)
        ENDIF
C
  40    LIMIT=LIMITD
C
C ***** END OF INITIALIZATION BLOCK. *****
C
C ***** MAIN LOOP. *****
C
  50    DO 400 ITER=1,LIMIT
        IF (Y(1) .LT. 0.0) THEN
          ARCLEN = S
          IFLAG = 5
          RETURN
        END IF
C
C TAKE A STEP ALONG THE CURVE.
C
        CALL STEPQF(N,NFE,IFLAGC,START,CRASH,HOLD,H,WK,
     &    RELERR,ABSERR,S,Y,YP,YOLD,YPOLD,A,QT,R,FO,F1,ZO,DZ,
     &    W,T,SSPAR,PAR,IPAR)
C
C CHECK IF THE STEP WAS SUCCESSFUL.
C
        IF (IFLAGC .GT. 0) THEN
          ARCLEN=S
          IFLAG=IFLAGC
          RETURN
        END IF
C
        IF (CRASH) THEN
C
C         RETURN CODE FOR ERROR TOLERANCE TOO SMALL.
C
          IFLAG=2
C
C         CHANGE ERROR TOLERANCES.
C
          IF (ARCRE .LT. RELERR) ARCRE=RELERR
          IF (ANSRE .LT. RELERR) ANSRE=RELERR
```

```
          IF (ARCAE .LT. ABSERR) ARCAE=ABSERR
          IF (ANSAE .LT. ABSERR) ANSAE=ABSERR
C
C         CHANGE LIMIT ON NUMBER OF ITERATIONS.
C
          LIMIT = LIMIT - ITER
          RETURN
        END IF
C
C IF LAMBDA >= 1.0, USE ROOTQF TO FIND SOLUTION.
C
        IF (Y(1) .GE. 1.0) GOTO 500
C
  400   CONTINUE
C
C ***** END OF MAIN LOOP *****
C
C DID NOT CONVERGE IN  LIMIT  ITERATIONS, SET  IFLAG  AND RETURN.
C
        ARCLEN = S
        IFLAG = 3
        RETURN
C
C ***** FINAL STEP -- FIND SOLUTION AT LAMBDA=1 *****
C
C SAVE  YOLD  FOR ARC LENGTH CALCULATION LATER.
C
  500   CALL DCOPY(NP1,YOLD,1,YSAV,1)
C
C FIND SOLUTION.
C
        CALL ROOTQF(N,NFE,IFLAGC,ANSRE,ANSAE,Y,YP,YOLD,
     &    YPOLD,A,QT,R,DZ,ZO,W,T,FO,F1,PAR,IPAR)
C
C CHECK IF SOLUTION WAS FOUND AND SET  IFLAG  ACCORDINGLY.
C
        IFLAG=1
C
C     SET ERROR FLAG IF ROOTQF COULD NOT GET THE POINT ON THE ZERO
C     CURVE AT  LAMBDA = 1.0.
C
        IF (IFLAGC .GT. 0) IFLAG=IFLAGC
C
C CALCULATE FINAL ARC LENGTH.
C
        CALL DCOPY(NP1,Y,1,DZ,1)
        WK=-1.0
        CALL DAXPY(NP1,WK,YSAV,1,DZ,1)
        ARCLEN = S - HOLD + DNRM2(NP1,DZ,1)
C
```

```
C ***** END OF FINAL STEP *****
C
      RETURN
C
C ***** END OF SUBROUTINE FIXPQF *****
      END
      SUBROUTINE STEPQF(N,NFE,IFLAG,START,CRASH,HOLD,H,
     &           WK,RELERR,ABSERR,S,Y,YP,YOLD,YPOLD,A,QT,R,
     &           F0,F1,ZO,DZ,W,T,SSPAR,PAR,IPAR)
C
C SUBROUTINE  STEPQF  TAKES ONE STEP ALONG THE ZERO CURVE OF THE
C HOMOTOPY MAP  RHO(LAMBDA,X)  USING A PREDICTOR-CORRECTOR ALGORITHM.
C THE PREDICTOR USES A HERMITE CUBIC INTERPOLANT, AND THE CORRECTOR
C RETURNS TO THE ZERO CURVE USING A QUASI-NEWTON ALGORITHM, REMAINING
C IN A HYPERPLANE PERPENDICULAR TO THE MOST RECENT TANGENT VECTOR.
C  STEPQF  ALSO ESTIMATES A STEP SIZE  H  FOR THE NEXT STEP ALONG THE
C ZERO CURVE.
C
C
C ON INPUT:
C
C N = DIMENSION OF  X.
C
C NFE = NUMBER OF JACOBIAN MATRIX EVALUATIONS.
C
C IFLAG = -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C START = .TRUE. ON FIRST CALL TO  STEPQF, .FALSE. OTHERWISE.
C         SHOULD NOT BE MODIFIED BY THE USER AFTER THE FIRST CALL.
C
C HOLD = ||Y - YOLD|| ; SHOULD NOT BE MODIFIED BY THE USER.
C
C H = UPPER LIMIT ON LENGTH OF STEP THAT WILL BE ATTEMPTED.  H  MUST
C    BE SET TO A POSITIVE NUMBER ON THE FIRST CALL TO  STEPQF.
C    THEREAFTER,  STEPQF  CALCULATES AN OPTIMAL VALUE FOR  H, AND  H
C    SHOULD NOT BE MODIFIED BY THE USER.
C
C WK = APPROXIMATE CURVATURE FOR THE LAST STEP (COMPUTED BY PREVIOUS
C    CALL TO  STEPQF).  UNDEFINED ON FIRST CALL.  SHOULD NOT BE
C    MODIFIED BY THE USER.
C
C RELERR, ABSERR = RELATIVE AND ABSOLUTE ERROR VALUES.  THE ITERATION
C    IS CONSIDERED TO HAVE CONVERGED WHEN A POINT  Z=(LAMBDA,X)  IS
C    FOUND SUCH THAT
C       ||DZ|| .LE. RELERR*||Z|| + ABSERR,
C    WHERE  DZ  IS THE LAST QUASI-NEWTON STEP.
C
C S  = (APPROXIMATE) ARC LENGTH ALONG THE HOMOTOPY ZERO CURVE UP TO
C    Y(S) = (LAMBDA(S), X(S)).
C
```

```
C Y(1:N+1) = PREVIOUS POINT (LAMBDA(S),X(S)) FOUND ON THE ZERO CURVE
C    OF THE HOMOTOPY MAP.
C
C YP(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY
C    MAP AT  Y.  INPUT IN THIS VECTOR IS NOT USED ON THE FIRST CALL
C    TO  STEPQF.
C
C YOLD(1:N+1) = A POINT BEFORE  Y  ON THE ZERO CURVE OF THE HOMOTOPY
C    MAP.  INPUT IN THIS VECTOR IS NOT USED ON THE FIRST CALL TO
C    STEPQF.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE
C    HOMOTOPY MAP AT  YOLD.
C
C A(1:N) = PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QT(1:N+1,1:N+1) = HOLDS  Q TRANSPOSE  OF THE QR FACTORIZATION OF
C    THE AUGMENTED JACOBIAN MATRIX AT  Y.
C
C R((N+1)*(N+2)/2) = HOLDS THE UPPER TRIANGLE OF  R  OF THE QR
C    FACTORIZATION, STORED BY ROWS.
C
C FO(1:N+1), F1(1:N+1), ZO(1:N+1), DZ(1:N+1), W(1:N+1), T(1:N+1) ARE
C    WORK ARRAYS.
C
C SSPAR(1:4) = PARAMETERS USED FOR COMPUTATION OF THE OPTIMAL STEP SIZE.
C    SSPAR(1) = HMIN, SSPAR(2) = HMAX, SSPAR(3) = BMIN, SSPAR(4) = BMAX.
C    THE OPTIMAL STEP  H  IS RESTRICTED SUCH THAT
C       HMIN .LE. H .LE. HMAX, AND  BMIN*HOLD .LE. H .LE. BMAX*HOLD.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C NFE HAS BEEN UPDATED.
C
C IFLAG
C
C    = -2, -1, OR 0 (UNCHANGED) ON A NORMAL RETURN.
C
C    = 4 IF A JACOBIAN MATRIX WITH RANK <  N  HAS OCCURRED.  THE
C        ITERATION WAS NOT COMPLETED.
C
C    = 6 IF THE ITERATION FAILED TO CONVERGE.
C
C START = .FALSE. ON A NORMAL RETURN.
C
```

```
C CRASH
C
C    = .FALSE. ON A NORMAL RETURN.
C
C    = .TRUE. IF THE STEP SIZE  H  WAS TOO SMALL.  H  HAS BEEN
C      INCREASED TO AN ACCEPTABLE VALUE, WITH WHICH  STEPQF  MAY BE
C      CALLED AGAIN.
C
C    = .TRUE. IF  RELERR  AND/OR  ABSERR  WERE TOO SMALL.  THEY HAVE
C      BEEN INCREASED TO ACCEPTABLE VALUES, WITH WHICH  STEPQF  MAY
C      BE CALLED AGAIN.
C
C HOLD = ||Y-YOLD||.
C
C H = OPTIMAL VALUE FOR NEXT STEP TO BE ATTEMPTED.  NORMALLY  H  SHOULD
C     NOT BE MODIFIED BY THE USER.
C
C WK = APPROXIMATE CURVATURE FOR THE STEP TAKEN BY  STEPQF.
C
C S = (APPROXIMATE) ARC LENGTH ALONG THE ZERO CURVE OF THE HOMOTOPY
C     MAP UP TO THE LATEST POINT FOUND, WHICH IS RETURNED IN  Y.
C
C RELERR, ABSERR  ARE UNCHANGED ON A NORMAL RETURN.  THEY ARE POSSIBLY
C     CHANGED IF  CRASH  = .TRUE. (SEE DESCRIPTION OF  CRASH  ABOVE).
C
C Y, YP, YOLD, YPOLD  CONTAIN THE TWO MOST RECENT POINTS AND TANGENT
C     VECTORS FOUND ON THE ZERO CURVE OF THE HOMOTOPY MAP.
C
C QT, R  STORE THE QR FACTORIZATION OF THE AUGMENTED JACOBIAN MATRIX
C     EVALUATED AT  Y.
C
C
C CALLS  D1MACH, DAXPY, DCOPY, DDOT, DNRM2, DSCAL, F (OR RHO), FJAC
C     (OR RHOJAC), QRFAQF, QRSLQF, TANGQF, UPQRQF.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION D1MACH, DDOT, DNRM2, QOFS
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION ALPHA, DDOO1, DDOO11, DDO1, DDO11, DELS, ETA,
     &    FOURU, GAMMA, HFAIL, HTEMP, IDLERR, ONE, PO, P1, PPO, PP1,
     &    TEMP, TWOU, WKOLD
        INTEGER I, ITCNT, LITFH, J, JP1, NP1
        LOGICAL FAILED
C
C     SCALAR ARGUMENTS
```

```
C
      INTEGER N, NFE, IFLAG
      LOGICAL START, CRASH
      DOUBLE PRECISION HOLD, H, WK, RELERR, ABSERR, S
C
C     ARRAY DECLARATIONS
C
      DOUBLE PRECISION Y(N+1), YP(N+1), YOLD(N+1), YPOLD(N+1),
     &   A(N), QT(N+1,N+1), R((N+1)*(N+2)/2), FO(N+1), F1(N+1),
     &   ZO(N+1), DZ(N+1), W(N+1), T(N+1), SSPAR(4), PAR(1)
      INTEGER IPAR(1)
C
      SAVE
C
C ***** END OF DECLARATIONS *****
C
C DEFINITION OF HERMITE CUBIC INTERPOLANT VIA DIVIDED DIFFERENCES.
C
      DD01(P0,P1,DELS) = (P1-P0)/DELS
      DD001(P0,PP0,P1,DELS) = (DD01(P0,P1,DELS)-PP0)/DELS
      DD011(P0,P1,PP1,DELS) = (PP1-DD01(P0,P1,DELS))/DELS
      DD0011(P0,PP0,P1,PP1,DELS) = (DD011(P0,P1,PP1,DELS) -
     &                              DD001(P0,PP0,P1,DELS))/DELS
      QOFS(P0,PP0,P1,PP1,DELS,S) = ((DD0011(P0,PP0,P1,PP1,DELS)*
     &   (S-DELS) + DD001(P0,PP0,P1,DELS))*S + PP0)*S + P0
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C
C ***** INITIALIZATION *****
C
C ETA = PARAMETER FOR BROYDEN'S UPDATE.
C LITFH = MAXIMUM NUMBER OF QUASI-NEWTON ITERATIONS ALLOWED.
C
      ONE = 1.0
      TWOU = 2.0*D1MACH(4)
      FOURU = TWOU + TWOU
      NP1 = N+1
      FAILED = .FALSE.
      CRASH = .TRUE.
      ETA = 50.0*TWOU
      LITFH = 2*(INT(-LOG10(ABSERR+RELERR*DNRM2(NP1,Y,1)))+1)
C
C CHECK THAT ALL INPUT PARAMETERS ARE CORRECT.
C
C     THE ARCLENGTH  S MUST BE NONNEGATIVE.
C
      IF (S .LT. 0.0) RETURN
C
C     IF STEP SIZE IS TOO SMALL, DETERMINE AN ACCEPTABLE ONE.
```

```
C
        IF (H .LT. FOURU*(1.0+S)) THEN
          H=FOURU*(1.0 + S)
          RETURN
        END IF
C
C     IF ERROR TOLERANCES ARE TOO SMALL, INCREASE THEM TO ACCEPTABLE
C     VALUES.
C
        TEMP=DNRM2(NP1,Y,1) + 1.0
        IF (.5*(RELERR*TEMP+ABSERR) .LT. TWOU*TEMP) THEN
          IF (RELERR .NE. 0.0) THEN
            RELERR = FOURU*(1.0+FOURU)
            TEMP = 0.0
            ABSERR = MAX(ABSERR,TEMP)
          ELSE
            ABSERR=FOURU*TEMP
          END IF
          RETURN
        END IF
C
C     INPUT PARAMETERS WERE ALL ACCEPTABLE.
C
        CRASH = .FALSE.
C
C COMPUTE  YP  ON FIRST CALL.
C NOTE:  DZ  IS USED SIMPLY AS A WORK ARRAY HERE.
C
        IF (START) THEN
          CALL TANGQF(Y,YP,YPOLD,A,QT,R,W,DZ,T,N,IFLAG,NFE,PAR,IPAR)
          IF (IFLAG .GT. 0) RETURN
        END IF
C
C FO = (RHO(Y), YP*Y) TRANSPOSE (DIFFERENT FOR EACH PROBLEM TYPE).
C
         IF (IFLAG .EQ. -2) THEN
C
C          CURVE TRACKING PROBLEM.
C
           CALL RHO(A,Y(1),Y(2),FO,PAR,IPAR)
         ELSE IF (IFLAG .EQ. -1) THEN
C
C          ZERO FINDING PROBLEM.
C
           CALL F(Y(2),FO)
           DO 5 I=1,N
             FO(I) = Y(1)*FO(I) + (1.0-Y(1))*(Y(I+1)-A(I))
  5        CONTINUE
         ELSE
C
```

```
C          FIXED POINT PROBLEM.
C
           CALL F(Y(2),FO)
           DO 10 I=1,N
             FO(I) = Y(1)*(A(I)-FO(I))+Y(I+1)-A(I)
  10       CONTINUE
         END IF
C
C        DEFINE LAST ROW OF FO  =  YP*Y.
C
           FO(NP1) = DDOT(NP1,YP,1,Y,1)
C
C ***** END OF INITIALIZATION *****
C
C ***** COMPUTE PREDICTOR POINT ZO *****
C
  20     IF (START) THEN
C
C          COMPUTE ZO WITH LINEAR PREDICTOR USING Y, YP --
C          ZO = Y+H*YP.
C
           CALL DCOPY(NP1,Y,1,ZO,1)
           CALL DAXPY(NP1,H,YP,1,ZO,1)
C
         ELSE
C
C          COMPUTE ZO WITH CUBIC PREDICTOR.
C
           DO 30 I=1,NP1
             ZO(I) = QOFS(YOLD(I),YPOLD(I),Y(I),YP(I),HOLD,HOLD+H)
  30       CONTINUE
C
         END IF
C
C F1 = (RHO(ZO), YP*ZO) TRANSPOSE.
C
         IF (IFLAG .EQ. -2) THEN
           CALL RHO(A,ZO(1),ZO(2),F1,PAR,IPAR)
         ELSE IF (IFLAG .EQ. -1) THEN
           CALL F(ZO(2),F1)
           DO 40 I=1,N
             F1(I) = ZO(1)*F1(I) + (1.0-ZO(1))*(ZO(I+1)-A(I))
  40       CONTINUE
         ELSE
           CALL F(ZO(2),F1)
           DO 50 I=1,N
             F1(I) = ZO(1)*(A(I)-F1(I))+ZO(I+1)-A(I)
  50       CONTINUE
         END IF
         F1(NP1) = DDOT(NP1,YP,1,ZO,1)
```

```
C
C ***** END OF PREDICTOR SECTION *****
C
C ***** SET-UP FOR QUASI-NEWTON ITERATION *****
C
      IF (FAILED) THEN
C
C GENERATE QT = AUGMENTED JACOBIAN MATRIX FOR POINT ZO=(LAMBDA,X).
C
        IF (IFLAG .EQ. -2) THEN
C
C         CURVE TRACKING PROBLEM:
C         D(RHO) = (D RHO(A,LAMBDA,X)/D LAMBDA, D RHO(A,LAMBDA,X)/DX).
C
          DO 60 J = 1,NP1
            CALL RHOJAC(A,ZO(1),ZO(2),QT(1,J),J,PAR,IPAR)
 60       CONTINUE
        ELSE IF (IFLAG .EQ. -1) THEN
C
C         ZERO FINDING PROBLEM:
C         D(RHO) = (F(X) - X + A, LAMBDA*DF(X) + (1-LAMBDA)*I).
C
          CALL F(ZO(2),QT(1,1))
          DO 70 I=1,N
            QT(I,1) = A(I) - ZO(I+1) + QT(I,1)
 70       CONTINUE
          DO 80 J= 1,N
            JP1 = J+1
            CALL FJAC(ZO(2),QT(1,JP1),J)
            CALL DSCAL(N, ZO(1), QT(1,JP1), 1)
            QT(J,JP1) = 1.0 - ZO(1) + QT(J,JP1)
 80       CONTINUE
        ELSE
C
C         FIXED POINT PROBLEM:
C         D(RHO) = (A - F(X), I - LAMBDA*DF(X)).
C
          CALL F(ZO(2),QT(1,1))
          CALL DSCAL(N,-ONE,QT(1,1),1)
          CALL DAXPY(N,ONE,A,1,QT(1,1),1)
          DO 90 J=1,N
            JP1 = J+1
            CALL FJAC(ZO(2),QT(1,JP1),J)
            CALL DSCAL(N, -ZO(1), QT(1,JP1), 1)
            QT(J,JP1) = 1.0 + QT(J,JP1)
 90       CONTINUE
        END IF
C
C       DEFINE LAST ROW OF QT = YP.
C
```

```
          CALL DCOPY(NP1, YP, 1, QT(NP1,1), NP1)
C
C       COUNT JACOBIAN EVALUATION.
C
          NFE = NFE+1
C
C DO FIRST QUASI NEWTON STEP.
C
C       FACTOR AUG.
C
          CALL QRFAQF(QT,R,NP1,IFLAG)
          IF (IFLAG .GT. 0) RETURN
C
C       COMPUTE NEWTON STEP.
C
          CALL DCOPY(N,F1,1,DZ,1)
          CALL DSCAL(N,-ONE,DZ,1)
          DZ(NP1) = 0.0
          CALL QRSLQF(QT,R,DZ,W,NP1)
C
C       TAKE STEP AND SET F0 = F1.
C
          CALL DAXPY(NP1, ONE, DZ, 1, Z0, 1)
          CALL DCOPY(NP1, F1, 1, F0, 1)
C
C       F1 = (RHO(Z0), YP*Z0) TRANSPOSE.
C
          IF (IFLAG .EQ. -2) THEN
            CALL RHO(A,Z0(1),Z0(2),F1,PAR,IPAR)
          ELSE IF (IFLAG .EQ. -1) THEN
            CALL F(Z0(2),F1)
            DO 100 I=1,N
              F1(I) = Z0(1)*F1(I) + (1.0-Z0(1))*(Z0(I+1)-A(I))
  100       CONTINUE
          ELSE
            CALL F(Z0(2),F1)
            DO 110 I=1,N
              F1(I) = Z0(1)*(A(I)-F1(I))+Z0(I+1)-A(I)
  110       CONTINUE
          END IF
          F1(NP1) = DDOT(NP1,YP,1,Z0,1)
C
        ELSE
C
C IF NOT FAILED THEN DEFINE  DZ=Z0-Y  PRIOR TO MAIN LOOP.
C
          CALL DCOPY(NP1,Z0,1,DZ,1)
          CALL DAXPY(NP1,-ONE,Y,1,DZ,1)
        END IF
C
```

```
C ***** END OF PREPARATION FOR QUASI-NEWTON ITERATION *****
C
C ***** QUASI-NEWTON ITERATION *****
C
      DO 140 ITCNT = 1,LITFH
C
C PERFORM UPDATE FOR NEWTON STEP JUST TAKEN.
C
        CALL UPQRQF(NP1,ETA,DZ,F0,F1,QT,R,W,T)
C
C COMPUTE NEXT NEWTON STEP.
C
        CALL DCOPY(N,F1,1,DZ,1)
        CALL DSCAL(N,-ONE,DZ,1)
        DZ(NP1) = 0.0
        CALL QRSLQF(QT,R,DZ,W,NP1)
C
C TAKE STEP.
C
        CALL DAXPY(NP1, ONE, DZ, 1, Z0, 1)
C
C CHECK FOR CONVERGENCE.
C
        IF (DNRM2(NP1,DZ,1) .LE. RELERR*DNRM2(NP1,Z0,1)+ABSERR) THEN
           GO TO 160
        END IF
C
C IF NOT CONVERGED, PREPARE FOR NEXT ITERATION.
C
C       F0 = F1.
C
        CALL DCOPY(NP1, F1, 1, F0, 1)
C
C       F1 = (RHO(Z0), YP*Z0) TRANSPOSE.
C
        IF (IFLAG .EQ. -2) THEN
          CALL RHO(A,Z0(1),Z0(2),F1,PAR,IPAR)
        ELSE IF (IFLAG .EQ. -1) THEN
          CALL F(Z0(2),F1)
          DO 120 I=1,N
            F1(I) = Z0(1)*F1(I) + (1.0-Z0(1))*(Z0(I+1)-A(I))
  120     CONTINUE
        ELSE
          CALL F(Z0(2),F1)
          DO 130 I=1,N
            F1(I) = Z0(1)*(A(I)-F1(I))+Z0(I+1)-A(I)
  130     CONTINUE
        END IF
        F1(NP1) = DDOT(NP1,YP,1,Z0,1)
C
```

```
  140   CONTINUE
C
C ***** END OF QUASI-NEWTON LOOP *****
C
C ***** DIDN'T CONVERGE OR TANGENT AT NEW POINT DID NOT MAKE
C       AN ACUTE ANGLE WITH YPOLD -- TRY AGAIN WITH A SMALLER H *****
C
  150   FAILED = .TRUE.
        HFAIL = H
        IF (H .LE. FOURU*(1.0 + S)) THEN
          IFLAG = 6
          RETURN
        ELSE
          H = .5 * H
        END IF
        GO TO 20
C
C ***** END OF CONVERGENCE FAILURE SECTION *****
C
C ***** CONVERGED -- MOP UP AND RETURN *****
C
C COMPUTE TANGENT & AUGMENTED JACOBIAN AT  ZO.
C NOTE:  DZ  AND  F1  ARE USED SIMPLY AS WORK ARRAYS HERE.
C
  160   CALL TANGQF(ZO,T,YP,A,QT,R,W,DZ,F1,N,IFLAG,NFE,PAR,IPAR)
        IF (IFLAG .GT. 0) RETURN
C
C CHECK THAT COMPUTED TANGENT  T  MAKES AN ANGLE NO LARGER THAN
C 60 DEGREES WITH CURRENT TANGENT  YP.  (I.E. COS OF ANGLE < .5)
C IF NOT, STEP SIZE WAS TOO LARGE, SO THROW AWAY ZO, AND TRY
C AGAIN WITH A SMALLER STEP.
C
        ALPHA = DDOT(NP1,T,1,YP,1)
        IF (ALPHA .LT. 0.5) GOTO 150
        ALPHA = ACOS(ALPHA)
C
C SET UP VARIABLES FOR NEXT CALL.
C
        CALL DCOPY(NP1,Y,1,YOLD,1)
        CALL DCOPY(NP1,ZO,1,Y,1)
        CALL DCOPY(NP1,YP,1,YPOLD,1)
        CALL DCOPY(NP1,T,1,YP,1)
C
C UPDATE ARCLENGTH   S = S + ||Y-YOLD||.
C
        HTEMP = HOLD
        CALL DAXPY(NP1,-ONE,YOLD,1,ZO,1)
        HOLD = DNRM2(NP1,ZO,1)
        S = S+HOLD
C
```

```
C COMPUTE OPTIMAL STEP SIZE.
C   IDLERR = DESIRED ERROR FOR NEXT PREDICTOR STEP.
C   WK = APPROXIMATE CURVATURE = 2*SIN(ALPHA/2)/HOLD  WHERE
C        ALPHA = ARCCOS(YP*YPOLD).
C   GAMMA = EXPECTED CURVATURE FOR NEXT STEP, COMPUTED BY
C        EXTRAPOLATING FROM CURRENT CURVATURE  WK, AND LAST
C        CURVATURE  WKOLD.  GAMMA IS FURTHER REQUIRED TO BE
C        POSITIVE.
C
         WKOLD = WK
         IDLERR = SQRT(SQRT(ABSERR + RELERR*DNRM2(NP1,Y,1)))
C
C     IDLERR SHOULD BE NO BIGGER THAN 1/2 PREVIOUS STEP.
C
         IDLERR = MIN(.5*HOLD,IDLERR)
         WK = 2.0*ABS(SIN(.5*ALPHA))/HOLD
         IF (START) THEN
            GAMMA = WK
         ELSE
            GAMMA = WK + HOLD/(HOLD+HTEMP)*(WK-WKOLD)
         END IF
         GAMMA = MAX(GAMMA, 0.01*ONE)
         H = SQRT(2.0*IDLERR/GAMMA)
C
C     ENFORCE RESTRICTIONS ON STEP SIZE SO AS TO ENSURE STABILITY.
C        HMIN <= H <= HMAX, BMIN*HOLD <= H <= BMAX*HOLD.
C
         H = MIN(MAX(SSPAR(1),SSPAR(3)*HOLD,H),SSPAR(4)*HOLD,SSPAR(2))
         IF (FAILED) H = MIN(HFAIL,H)
         START = .FALSE.
C
C ***** END OF MOP UP SECTION *****
C
         RETURN
C
C ***** END OF SUBROUTINE STEPQF *****
         END
         SUBROUTINE TANGQF(Y,YP,YPOLD,A,QT,R,W,S,T,N,IFLAG,NFE,PAR,IPAR)
C
C SUBROUTINE  TANGQF  COMPUTES THE UNIT TANGENT VECTOR  YP  TO THE
C ZERO CURVE OF THE HOMOTOPY MAP AT  Y  BY GENERATING THE AUGMENTED
C JACOBIAN MATRIX
C
C           --           --
C           |  D(RHO(Y))   |
C     AUG = |        T     |,   WHERE RHO IS THE HOMOTOPY MAP,
C           |   YPOLD      |
C           --           --
C
C SOLVING THE SYSTEM
```

```
C                                  T
C         AUG*YPT = (0,0,...,0,1)    FOR YPT,
C
C AND FINALLY COMPUTING  YP = YPT/||YPT||.
C
C IN ADDITION, THE MATRIX AUG IS UPDATED SO THAT THE LAST ROW IS
C YP  INSTEAD OF  YPOLD  ON RETURN.
C
C
C ON INPUT:
C
C Y(1:N+1) = CURRENT POINT (LAMBDA(S), X(S)).
C
C YP(1:N+1)  IS UNDEFINED ON INPUT.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR AT THE PREVIOUS POINT ON THE
C    ZERO CURVE OF THE HOMOTOPY MAP.
C
C A(1:N)  IS THE PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C W(1:N+1), S(1:N+1), T(1:N+1)  ARE WORK ARRAYS.
C
C N  IS THE DIMENSION OF X, WHERE  Y=(LAMBDA(S),X(S)).
C
C IFLAG  IS -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C NFE  IS THE NUMBER OF JACOBIAN EVALUATIONS.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C Y, YPOLD, A, N  ARE UNCHANGED.
C
C YP(1:N+1)  CONTAINS THE NEW UNIT TANGENT VECTOR TO THE ZERO
C    CURVE OF THE HOMOTOPY MAP AT  Y(S) = (LAMBDA(S), X(S)).
C
C QT(1:N+1,1:N+1)  CONTAINS  Q TRANSPOSE  OF THE QR FACTORIZATION  OF
C    THE JACOBIAN MATRIX OF RHO EVALUATED AT  Y  AUGMENTED BY
C    YP TRANSPOSE.
C
C R(1:(N+1)*(N+2)/2)  CONTAINS THE UPPER TRIANGLE (STORED BY ROWS)
C    OF THE  R  PART OF THE QR FACTORIZATION  OF THE AUGMENTED JACOBIAN
C    MATRIX.
C
C IFLAG  = -2, -1, OR 0, (UNCHANGED) ON A NORMAL RETURN.
C        = 4 IF THE AUGMENTED JACOBIAN MATRIX HAS RANK LESS THAN N+1.
```

```
C
C NFE  HAS BEEN INCREMENTED BY 1.
C
C
C CALLS  DCOPY, DNRM2, DSCAL, F (OR RHO IF IFLAG = -2), FJAC
C    (OR RHOJAC, IF IFLAG = -2), R1UPQF (WHICH IS AN ENTRY POINT OF
C    UPQRQF), QRFAQF, QRSLQF.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION DNRM2
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION LAMBDA, ONE, YPNRM
        INTEGER I, J, JP1, NP1
C
C     SCALAR ARGUMENTS
C
        INTEGER N, IFLAG, NFE
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION Y(N+1), YP(N+1), YPOLD(N+1), A(N),
     &    QT(N+1,N+1), R((N+1)*(N+2)/2), W(N+1), S(N+1), T(N+1),PAR(1)
        INTEGER IPAR(1)
C
C ***** END OF DECLARATIONS *****
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
        ONE = 1.0
        NFE = NFE + 1
        NP1 = N + 1
        LAMBDA = Y(1)
C
C ***** DEFINE THE AUGMENTED JACOBIAN MATRIX *****
C
C QT = AUG.
C
        IF (IFLAG .EQ. -2) THEN
C
C         CURVE TRACKING PROBLEM:
C         D(RHO) = (D RHO(A,LAMBDA,X)/D LAMBDA, D RHO(A,LAMBDA,X)/DX).
C
          DO 10 J = 1,NP1
            CALL RHOJAC(A,LAMBDA,Y(2),QT(1,J),J,PAR,IPAR)
  10      CONTINUE
```

```
      ELSE IF (IFLAG .EQ. -1) THEN
C
C       ZERO FINDING PROBLEM:
C       D(RHO) = (F(X) - X + A, LAMBDA*DF(X) + (1-LAMBDA)*I)
C
        CALL F(Y(2),QT(1,1))
        DO 20 I=1,N
          QT(I,1) = A(I) - Y(I+1) + QT(I,1)
 20     CONTINUE
        DO 30 J= 1,N
          JP1 = J+1
          CALL FJAC(Y(2),QT(1,JP1),J)
          CALL DSCAL(N,LAMBDA,QT(1,JP1),1)
          QT(J,JP1) = 1.0 - LAMBDA + QT(J,JP1)
 30     CONTINUE
      ELSE
C
C       FIXED POINT PROBLEM:
C       D(RHO) = (A - F(X), I - LAMBDA*DF(X)).
C
        CALL F(Y(2),QT(1,1))
        CALL DSCAL(N,-ONE,QT(1,1),1)
        CALL DAXPY(N,ONE,A,1,QT(1,1),1)
        DO 50 J=1,N
          JP1 = J+1
          CALL FJAC(Y(2),QT(1,JP1),J)
          CALL DSCAL(N,-LAMBDA,QT(1,JP1),1)
          QT(J,JP1) = 1.0 + QT(J,JP1)
 50     CONTINUE
      END IF
C
C     DEFINE LAST ROW OF QT  = YPOLD.
C
      CALL DCOPY(NP1,YPOLD,1,QT(NP1,1),NP1)
C
C ***** END OF DEFINITION OF AUGMENTED JACOBIAN MATRIX *****
C
C                                              T
C ***** SOLVE SYSTEM  AUG*YPT = (0,...,0,1)  *****
C
C FACTOR MATRIX.
C
      CALL QRFAQF(QT,R,NP1,IFLAG)
C
C IF MATRIX IS SINGULAR, THEN QUIT.
C
C       IF (IFLAG .EQ. 4) RETURN
C
C ELSE SOLVE SYSTEM  R*YP = QT*(0,...,0,1)  FOR YP.
C
```

```
      DO 70 J=1,N
        YP(J) = 0.0
 70   CONTINUE
      YP(NP1) = 1.0
      CALL QRSLQF(QT,R,YP,W,NP1)
C
C     COMPUTE UNIT VECTOR.
C
      YPNRM = 1.0/DNRM2(NP1,YP,1)
      CALL DSCAL(NP1,YPNRM,YP,1)
C
C ***** SYSTEM SOLVED *****
C
C ***** UPDATE AUGMENTED SYSTEM SO THAT LAST ROW IS YP *****
C
C S=YP-YPOLD,  T = QT*E(NP1).
C
      CALL DCOPY(NP1,YP,1,S,1)
      CALL DAXPY(NP1,-ONE,YPOLD,1,S,1)
      CALL DCOPY(NP1,QT(1,NP1),1,T,1)
      CALL R1UPQF(NP1,S,T,QT,R,W)
C
      RETURN
C
C ***** END OF SUBROUTINE TANGQF *****
      END
      SUBROUTINE ROOTQF(N,NFE,IFLAG,RELERR,ABSERR,Y,YP,YOLD,
     &     YPOLD,A,QT,R,DZ,Z,W,T,FO,F1,PAR,IPAR)
C
C ROOTQF  FINDS THE POINT  YBAR = (1, XBAR)  ON THE ZERO CURVE OF THE
C HOMOTOPY MAP.  IT STARTS WITH TWO POINTS  YOLD=(LAMBDAOLD,XOLD)  AND
C Y=(LAMBDA,X)  SUCH THAT  LAMBDAOLD < 1 <= LAMBDA, AND ALTERNATES
C BETWEEN USING A SECANT METHOD TO FIND A PREDICTED POINT ON THE
C HYPERPLANE  LAMBDA=1, AND TAKING A QUASI-NEWTON STEP TO RETURN TO THE
C ZERO CURVE OF THE HOMOTOPY MAP.
C
C
C ON INPUT:
C
C N = DIMENSION OF X.
C
C NFE = NUMBER OF JACOBIAN MATRIX EVALUATIONS.
C
C IFLAG = -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C RELERR, ABSERR = RELATIVE AND ABSOLUTE ERROR VALUES.  THE ITERATION IS
C    CONSIDERED TO HAVE CONVERGED WHEN A POINT Y=(LAMBDA,X) IS FOUND
C    SUCH THAT
C
C    |Y(1) - 1| <= RELERR + ABSERR                AND
```

```
C
C     ||DZ|| <= RELERR*||Y|| + ABSERR,            WHERE
C
C     DZ  IS THE QUASI-NEWTON STEP TO Y.
C
C Y(1:N+1) = POINT (LAMBDA(S), X(S)) ON ZERO CURVE OF HOMOTOPY MAP.
C
C YP(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY MAP
C     AT  Y.
C
C YOLD(1:N+1) = A POINT DIFFERENT FROM  Y  ON THE ZERO CURVE.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY
C     MAP AT  YOLD.
C
C A(1:*) = PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QT(1:N+1,1:N+1) CONTAINS  Q  TRANSPOSE OF THE QR FACTORIZATION OF
C     THE AUGMENTED JACOBIAN MATRIX EVALUATED AT THE POINT Y.
C
C R((N+1)*(N+2)/2) CONTAINS THE UPPER TRIANGLE OF THE R PART OF
C     OF THE QR FACTORIZATION, STORED BY ROWS.
C
C DZ(1:N+1), Z(1:N+1), W(1:N+1), T(1:N+1), FO(1:N+1), F1(1:N+1)
C     ARE WORK ARRAYS USED FOR THE QUASI-NEWTON STEP AND THE SECANT
C     STEP.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C     WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C     RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C N, RELERR, ABSERR, AND A  ARE UNCHANGED.
C
C NFE  HAS BEEN UPDATED.
C
C IFLAG
C     = -2, -1, OR 0 (UNCHANGED) ON A NORMAL RETURN.
C
C     = 4 IF A SINGULAR JACOBIAN MATRIX OCCURRED.  THE
C         ITERATION WAS NOT COMPLETED.
C
C     = 6 IF THE ITERATION FAILED TO CONVERGE.  Y  AND  YOLD  CONTAIN
C         THE LAST TWO POINTS OBTAINED BY QUASI-NEWTON STEPS, AND  YP
C         CONTAINS A POINT OPPOSITE OF THE HYPERPLANE  LAMBDA=1  FROM
C         Y.
C
C Y  IS THE POINT ON THE ZERO CURVE OF THE HOMOTOPY MAP AT  LAMBDA = 1.
```

```
C
C YP  AND  YOLD  CONTAIN POINTS NEAR THE SOLUTION.
C
C CALLS  D1MACH, DAXPY, DCOPY, DDOT, DNRM2, F (OR RHO),
C        QRSLQF, ROOT, UPQRQF.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION D1MACH, DDOT, DNRM2, QOFS
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION AERR, DDO01, DDO011, DD01, DD011, DELS, ETA,
     &     ONE, P0, P1, PP0, PP1, QSOUT, RERR, S, SA, SB, SOUT,
     &     U, ZERO
        INTEGER ISTEP, I, LCODE, LIMIT,NP1
        LOGICAL BRACK
C
C     SCALAR ARGUMENTS
C
        DOUBLE PRECISION RELERR, ABSERR
        INTEGER N, NFE, IFLAG
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION Y(N+1), YP(N+1), YOLD(N+1), YPOLD(N+1), A(N),
     &     QT(N+1:N+1), R((N+1)*(N+2)/2), DZ(N+1), Z(N+1), W(N+1),
     &     T(N+1), FO(N+1), F1(N+1), PAR(1)
        INTEGER IPAR(1)
C
C ***** END OF DECLARATIONS *****
C
C
C DEFINITION OF HERMITE CUBIC INTERPOLANT VIA DIVIDED DIFFERENCES.
C
        DD01(P0,P1,DELS)=(P1-P0)/DELS
        DDO01(P0,PP0,P1,DELS)=(DD01(P0,P1,DELS)-PP0)/DELS
        DD011(P0,P1,PP1,DELS)=(PP1-DD01(P0,P1,DELS))/DELS
        DDO011(P0,PP0,P1,PP1,DELS)=(DD011(P0,P1,PP1,DELS)  -
     &                              DDO01(P0,PP0,P1,DELS))/DELS
        QOFS(P0,PP0,P1,PP1,DELS,S)=((DDO011(P0,PP0,P1,PP1,DELS)*
     &     (S-DELS) + DDO01(P0,PP0,P1,DELS))*S  + PP0)*S + P0
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C ***** INITIALIZATION *****
C
C ETA = PARAMETER FOR BROYDEN'S UPDATE.
```

```
C LIMIT = MAXIMUM NUMBER OF ITERATIONS ALLOWED.
C
        ONE=1.0
        ZERO=0.0
        U=D1MACH(4)
        RERR=MAX(RELERR,U)
        AERR=MAX(ABSERR,ZERO)
        NP1=N+1
        ETA = 100.0*U
        LIMIT = 2*(INT(-LOG10(AERR+RERR*DNRM2(NP1,Y,1)))+1)
C
C FO = (RHO(Y), YP*Y) TRANSPOSE.
C
        IF (IFLAG .EQ. -2) THEN
C
C         CURVE TRACKING PROBLEM.
C
          CALL RHO(A,Y(1),Y(2),FO,PAR,IPAR)
        ELSE IF (IFLAG .EQ. -1) THEN
C
C         ZERO FINDING PROBLEM.
C
          CALL F(Y(2),FO)
          DO 10 I=1,N
            FO(I) = Y(1)*FO(I) + (1.0-Y(1))*(Y(I+1)-A(I))
10        CONTINUE
        ELSE
C
C         FIXED POINT PROBLEM.
C
          CALL F(Y(2),FO)
          DO 20 I=1,N
            FO(I) = Y(1)*(A(I)-FO(I))+Y(I+1)-A(I)
20        CONTINUE
        END IF
        FO(NP1) = DDOT(NP1,YP,1,Y,1)
C
C ***** END OF INITIALIZATION BLOCK *****
C
C ***** COMPUTE FIRST INTERPOLANT WITH A HERMITE CUBIC *****
C
C FIND DISTANCE BETWEEN Y AND YOLD.  DZ=||Y-YOLD||.
C
        CALL DCOPY(NP1,Y,1,DZ,1)
        CALL DAXPY(NP1,-ONE,YOLD,1,DZ,1)
        DELS=DNRM2(NP1,DZ,1)
C
C USING TWO POINTS AND TANGENTS ON THE HOMOTOPY ZERO CURVE, CONSTRUCT
C THE HERMITE CUBIC INTERPOLANT Q(S).  THEN USE  ROOT  TO FIND THE S
C CORRESPONDING TO  LAMBDA = 1.  THE TWO POINTS ON THE ZERO CURVE ARE
```

```
C ALWAYS CHOSEN TO BRACKET  LAMBDA=1, WITH THE BRACKETING INTERVAL
C ALWAYS BEING [0, DELS].
C
        SA=0.0
        SB=DELS
        LCODE=1
40      CALL ROOT(SOUT,QSOUT,SA,SB,RERR,AERR,LCODE)
          IF (LCODE .GT. 0) GO TO 50
          QSOUT=QOFS(YOLD(1),YPOLD(1),Y(1),YP(1),DELS,SOUT)  - 1.0
          GO TO 40
C
C     IF  LAMBDA = 1  WERE BRACKETED,  ROOT  CANNOT FAIL.
C
50      IF (LCODE .GT. 2) THEN
          IFLAG=6
          RETURN
        ENDIF
C
C CALCULATE  Q(SA)  AS THE INITIAL POINT FOR A NEWTON ITERATION.
C
        DO 60 I=1,NP1
          Z(I)=QOFS(YOLD(I),YPOLD(I),Y(I),YP(I),DELS,SA)
60      CONTINUE
C
C CALCULATE DZ = Z-Y.
C
        CALL DCOPY(NP1,Z,1,DZ,1)
        CALL DAXPY(NP1,-ONE,Y,1,DZ,1)
C
C ***** END OF CALCULATION OF CUBIC INTERPOLANT *****
C
C TANGENT INFORMATION  YPOLD  IS NO LONGER NEEDED.  HEREAFTER,  YPOLD
C REPRESENTS THE MOST RECENT POINT WHICH IS ON THE OPPOSITE SIDE OF
C LAMBDA=1  FROM  Y.
C
C ***** PREPARE FOR MAIN LOOP *****
C
        CALL DCOPY(NP1,YOLD,1,YPOLD,1)
C
C INITIALIZE  BRACK  TO INDICATE THAT THE POINTS  Y  AND  YOLD  BRACKET
C LAMBDA=1,  THUS YOLD = YPOLD.
C
        BRACK = .TRUE.
C
C ***** MAIN LOOP *****
C
        DO 300 ISTEP=1,LIMIT
C
C UPDATE JACOBIAN MATRIX.
C
```

```
C       F1=(RHO(Z), YP*Z) TRANSPOSE.
C
          IF (IFLAG .EQ. -2) THEN
            CALL RHO(A,Z(1),Z(2),F1,PAR,IPAR)
          ELSE IF (IFLAG .EQ. -1) THEN
            CALL F(Z(2),F1)
            DO 80 I=1,N
              F1(I) = Z(1)*F1(I) + (1-Z(1))*(Z(I+1)-A(I))
  80        CONTINUE
          ELSE
            CALL F(Z(2),F1)
            DO 90 I=1,N
              F1(I) = Z(1)*(A(I)-F1(I))+Z(I+1)-A(I)
  90        CONTINUE
          END IF
          F1(NP1) = DDOT(NP1,YP,1,Z,1)
C
C
C PERFORM BROYDEN UPDATE.
C
          CALL UPQRQF(NP1,ETA,DZ,F0,F1,QT,R,W,T)
C
C QUASI-NEWTON STEP.
C
C       COMPUTE NEWTON STEP.
C
          CALL DCOPY(N,F1,1,DZ,1)
          CALL DSCAL(N,-ONE,DZ,1)
          DZ(NP1) = 0.0
          CALL QRSLQF(QT,R,DZ,W,NP1)
C
C       TAKE NEWTON STEP.
C
          CALL DCOPY(NP1,Z,1,W,1)
          CALL DAXPY(NP1,ONE,DZ,1,Z,1)
C
C       CHECK FOR CONVERGENCE.
C
          IF ((ABS(Z(1)-1.0) .LE. RERR+AERR) .AND.
     &        (DNRM2(NP1,DZ,1) .LE. RERR*DNRM2(N,Z(2),1)+AERR)) THEN
            CALL DCOPY(NP1,Z,1,Y,1)
            RETURN
          END IF
C
C PREPARE FOR NEXT ITERATION.
C
C       F0 = F1.
C
          CALL DCOPY(NP1,F1,1,F0,1)
C
```

```
C       IF  Z(1) = 1.0  THEN PERFORM QUASI-NEWTON ITERATION AGAIN
C       WITHOUT COMPUTING A NEW PREDICTOR.
C
          IF (ABS(Z(1)-1.0) .LE. RERR*AERR) THEN
             CALL DCOPY(NP1,Z,1,DZ,1)
             CALL DAXPY(NP1,-ONE,W,1,DZ,1)
             GOTO 300
          END IF
C
C       UPDATE  Y  AND  YOLD.
C
          CALL DCOPY(NP1,Y,1,YOLD,1)
          CALL DCOPY(NP1,Z,1,Y,1)
C
C       UPDATE  YPOLD  SUCH THAT  YPOLD  IS THE MOST RECENT POINT
C       OPPOSITE OF  LAMBDA=1  FROM  Y.  SET  BRACK = .TRUE.  IFF
C       Y & YOLD  BRACKET  LAMBDA=1  SO THAT  YPOLD=YOLD.
C
          IF ((Y(1)-1.0)*(YOLD(1)-1.0)  .GT. 0) THEN
            BRACK = .FALSE.
          ELSE
            BRACK = .TRUE.
            CALL DCOPY(NP1,YOLD,1,YPOLD,1)
          END IF
C
C       COMPUTE DELS = ||Y-YPOLD||.
C
          CALL DCOPY(NP1,Y,1,DZ,1)
          CALL DAXPY(NP1,-ONE,YPOLD,1,DZ,1)
          DELS=DNRM2(NP1,DZ,1)
C
C       COMPUTE  DZ  FOR THE LINEAR PREDICTOR   Z = Y + DZ,
C           WHERE  DZ = SA*(YOLD-Y).
C
          SA = (1.0-Y(1))/(YOLD(1)-Y(1))
          CALL DCOPY(NP1,YOLD,1,DZ,1)
          CALL DAXPY(NP1,-ONE,Y,1,DZ,1)
          CALL DSCAL(NP1,SA,DZ,1)
C
C       TO INSURE STABILITY, THE LINEAR PREDICTION MUST BE NO FARTHER
C       FROM  Y  THAN  YPOLD  IS.  THIS IS GUARANTEED IF  BRACK = .TRUE.
C       IF LINEAR PREDICTION IS TOO FAR AWAY, USE BRACKETING POINTS
C       TO COMPUTE LINEAR PREDICTION.
C
          IF (.NOT. BRACK) THEN
            IF (DNRM2(NP1,DZ,1)  .GT. DELS) THEN
C
C             COMPUTE  DZ = SA*(YPOLD-Y).
C
              SA = (1.0-Y(1))/(YPOLD(1)-Y(1))
```

```
            CALL DCOPY(NP1,YPOLD,1,DZ,1)
            CALL DAXPY(NP1,-ONE,Y,1,DZ,1)
            CALL DSCAL(NP1,SA,DZ,1)
          END IF
        END IF
C
C       COMPUTE PREDICTOR Z = Y+DZ, AND DZ = NEW Z  - OLD Z (USED FOR
C         QUASI-NEWTON UPDATE).
C
          CALL DAXPY(NP1,ONE,DZ,1,Z,1)
          CALL DCOPY(NP1,Z,1,DZ,1)
          CALL DAXPY(NP1,-ONE,W,1,DZ,1)
  300   CONTINUE
C
C ***** END OF MAIN LOOP. *****
C
C THE ALTERNATING OSCULATORY LINEAR PREDICTION AND QUASI-NEWTON
C CORRECTION HAS NOT CONVERGED IN  LIMIT  STEPS.  ERROR RETURN.
        IFLAG=6
        RETURN
C
C ***** END OF SUBROUTINE ROOTQF *****
        END
        SUBROUTINE QRFAQF(QT,R,N,IFLAG)
C
C SUBROUTINE  QRFAQF  COMPUTES THE QR FACTORIZATION OF A MATRIX  A,
C WHERE  R  IS AN UPPER TRIANGULAR MATRIX, AND  Q  IS AN ORTHOGONAL
C MATRIX WHICH IS THE PRODUCT OF N-1 HOUSEHOLDER TRANSFORMATIONS
C
C   Q=H1*H2*...*H(N-1).
C
C THE ROUTINE HAS TWO MAJOR STEPS.  FIRST, THE QR FACTORIZATION
C OF  A  IS COMPUTED, RESULTING IN DEFINING THE VECTOR  R,  AND
C STORING INFORMATION IN THE LOWER TRIANGLE OF  QT  WHICH WILL
C ENABLE THE CONSTRUCTION OF  Q TRANSPOSE.
C
C THE SECOND STEP CONSTRUCTS  Q TRANSPOSE FROM THE INFORMATION
C STORED IN  QT,  AND PLACES IT IN  QT.
C
C THE INFORMATION STORED IN THE LOWER TRIANGLE OF  QT  DURING THE FIRST
C STEP ARE THE VECTORS  UJ, WHICH DEFINE THE HOUSEHOLDER TRANSFORMATIONS
C
C                   T
C   HJ = I - (UJ*UJ  / PJ),  WHERE UJ[I]=0 FOR I=1...J-1,
C                             UJ[I]=QT[I,J], FOR I=J...N,
C                             PJ = THE JTH COMPONENT OF UJ.
C
C
C ON INPUT:
C
```

```
C QT(1:N,1:N)  CONTAINS THE MATRIX  A  TO BE FACTORED.
C
C R(1:N*(N+1)/2)  IS UNDEFINED.
C
C N  IS THE DIMENSION OF THE MATRIX TO BE FACTORED.
C
C IFLAG  IS UNDEFINED.
C
C
C ON OUTPUT:
C
C QT  CONTAINS  Q TRANSPOSE.
C
C R(1:N*(N+1)/2)  CONTAINS THE UPPER TRIANGLE OF  R  STORED BY ROWS.
C
C N  IS UNCHANGED.
C
C IFLAG = 4  IF THE MATRIX A IS SINGULAR.  OTHERWISE,  IFLAG
C    IS UNCHANGED.
C
C
C CALLS  DAXPY, DCOPY, DDOT, DNRM2, DSCAL.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION DDOT, DNRM2
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION ONE, TAU, TEMP
        INTEGER I, J, K, INDEXR, ISIGN
C
C     SCALAR ARGUMENTS
C
        INTEGER N, IFLAG
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION QT(N,N),R(N)
C
C ***** END OF DECLARATIONS *****
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
        ONE = 1.0
C
C ***** CALCULATION OF QR DECOMPOSITION, PLACING R IN THE VECTOR *****
C       R, AND PLACING THE UJ VECTORS IN THE LOWER TRIANGLE OF
```

```
C       QT.
C
        INDEXR = 1
        DO 20 K=1,N-1
          TEMP = DNRM2(N-K+1,QT(K,K),1)
          IF (TEMP .EQ. 0.0) THEN
C
C           MATRIX IS SINGULAR, SET  IFLAG  AND RETURN.
C
            IFLAG = 4
            RETURN
          ELSE
C
C           FORM QK AND PREMULTIPLY  QT  BY IT.
C                                                          T
C           UK = EK - ISIGN*X/||X||,  WHERE  HK = I-(UK*UK /PK),
C              PK = THE KTH COMPONENT OF UK,
C              EK = THE KTH NATURAL BASIS VECTOR,
C              X  = THE KTH COLUMN OF THE MATRIX  H(K-1)...H2*H1*QT,
C              ISIGN = THE SIGN OF PK.
C
C           GET SIGN.
C
            ISIGN = SIGN(ONE,QT(K,K))
C
C           COMPUTE R(K,K).
C
            R(INDEXR) = -ISIGN*TEMP
C
C           UPDATE KTH COLUMN.
C
            TEMP = ISIGN/TEMP
            CALL DSCAL(N-K+1,TEMP,QT(K,K),1)
            QT(K,K) = QT(K,K) + 1.0
C
C           UPDATE THE K+1ST - NTH COLUMNS OF  QT, AND  R.
C
            INDEXR = INDEXR + 1
            DO 10 J=K+1,N
              TAU = DDOT(N-K+1,QT(K,K),1,QT(K,J),1)/QT(K,K)
              R(INDEXR) = QT(K,J) - TAU*QT(K,K)
              INDEXR = INDEXR + 1
              CALL DAXPY(N-K,-TAU,QT(K+1,K),1,QT(K+1,J),1)
 10         CONTINUE
          END IF
 20     CONTINUE
        IF (QT(N,N) .EQ. 0.0) THEN
C
C         MATRIX IS SINGULAR, SET  IFLAG  AND RETURN.
C
```

```
          IFLAG = 4
          RETURN
        END IF
        R(INDEXR) = QT(N,N)
C
C ***** END OF FACTORING STEP *****
C
C ***** CONSTRUCT Q TRANSPOSE IN QT *****
C
C FORM Q BY MULTIPLYING ((I*H(N-1))*...)*H1.
C THIS IS DONE IN PLACE IN  QT  BY UPDATING ONLY THE LOWER
C RIGHT HAND CORNER OF QT  (QT(K,K) TO QT(N,N)).
C
C
        QT(N,N) = 1.0
        DO 40 K=N-1,1,-1
C
C         MULTIPLY  QT  BY H(K).
C
          TEMP = QT(K,K)
C
C         UPDATE ROW K.
C
          QT(K,K) = 1.0-QT(K,K)
          CALL DCOPY(N-K,QT(K+1,K),1,QT(K,K+1),N)
          CALL DSCAL(N-K,-ONE,QT(K,K+1),N)
C
C         UPDATE REMAINING ROWS.
C
          DO 30 I=N,K+1,-1
            TAU = -DDOT(N-K,QT(I,K+1),N,QT(K,K+1),N)
            QT(I,K) = -TAU
            TAU = TAU/TEMP
            CALL DAXPY(N-K,TAU,QT(K,K+1),N,QT(I,K+1),N)
  30      CONTINUE
  40    CONTINUE
C
C ***** END OF Q TRANSPOSE CONSTRUCTION *****
C
        RETURN
C
C ***** END OF SUBROUTINE QRFAQF *****
        END
        SUBROUTINE QRSLQF(QT,R,B,X,N)
C
C SUBROUTINE QRSLQF SOLVES THE SYSTEM  R*S = QT*B  FOR  S.
C
C
C ON INPUT:
C
```

```
C QT(1:N,1:N)  CONTAINS QT IN THE EQUATION ABOVE.
C
C R(1:N*(N+1)/2)  CONTAINS THE UPPER TRIANGLE OF  R  IN THE EQUATION
C     ABOVE, STORED BY ROWS.
C
C B(1:N)  CONTAINS  B  IN THE EQUATION ABOVE.
C
C N   IS THE DIMENSION OF THE PROBLEM.
C
C
C ON OUTPUT:
C
C QT  AND  R  ARE UNCHANGED.
C
C B   CONTAINS THE SOLUTION VECTOR S.
C
C X(1:N)  IS A WORK ARRAY WHICH CONTAINS QT*B ON OUTPUT.
C
C
C CALLS DDOT.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION DDOT
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION TAU
        INTEGER INDEXR, I, J
C
C     SCALAR ARGUMENTS
C
        INTEGER N
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION QT(N,N),R(N*(N+1)/2),B(N),X(N)
C
C ***** END OF DECLARATIONS *****
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C X = QT*B.
C
        DO 10 I=1,N
          X(I) = DDOT(N,QT(I,1),N,B,1)
  10    CONTINUE
C
```

```
C COMPUTE S USING BACK SUBSTITUTION.
C
      INDEXR = N*(N+1)/2
      B(N) = X(N)/R(INDEXR)
      INDEXR = INDEXR - 1
      DO 30 I=N-1,1,-1
        TAU = X(I)
        DO 20 J=N,I+1,-1
          TAU = TAU - R(INDEXR)*B(J)
          INDEXR = INDEXR - 1
 20     CONTINUE
        B(I) = TAU/R(INDEXR)
        INDEXR = INDEXR - 1
 30   CONTINUE
      RETURN
C
C ***** END OF SUBROUTINE QRSLQF *****
      END
      SUBROUTINE UPQRQF(N,ETA,S,FO,F1,QT,R,W,T)
C
C SUBROUTINE  UPQRQF  PERFORMS A BROYDEN UPDATE ON THE  Q R
C FACTORIZATION OF A MATRIX  A, (AN APPROXIMATION TO J(XO)),
C RESULTING IN THE FACTORIZATION  Q+ R+ OF
C
C       A+  =  A  +  (Y - A*S) (ST)/(ST * S),
C
C (AN APPROXIMATION TO J(X1))
C WHERE S = X1 - XO, ST = S TRANSPOSE,  Y = F(X1) - F(XO).
C
C THE ENTRY POINT  R1UPQF  PERFORMS THE RANK ONE UPDATE ON THE QR
C FACTORIZATION OF
C
C       A+ =  A + Q*(T*ST).
C
C
C ON INPUT:
C
C N  IS THE DIMENSION OF X AND F(X).
C
C ETA  IS A NOISE PARAMETER.  IF (Y-A*S)(I) .LE. ETA*(|F1(I)|+|FO(I)|)
C    FOR 1 .LE. I .LE. N, THEN NO UPDATE IS PERFORMED.
C
C S(1:N) = X1 - XO   (OR S FOR THE ENTRY POINT R1UPQF).
C
C FO(1:N) = F(XO).
C
C F1(1:N) = F(X1).
C
C QT(1:N,1:N)  CONTAINS THE OLD Q TRANSPOSE, WHERE  A = Q*R .
C
```

```
C R(1:N*(N+1)/2)  CONTAINS THE OLD R, STORED BY ROWS.
C
C W(1:N), T(1:N)  ARE WORK ARRAYS ( T  CONTAINS THE VECTOR T FOR THE
C    ENTRY POINT  R1UPQF ).
C
C
C ON OUTPUT:
C
C N  AND  ETA  ARE UNCHANGED.
C
C QT  CONTAINS Q+ TRANSPOSE.
C
C R   CONTAINS R+, STORED BY ROWS.
C
C S, F0, F1, W, AND T  HAVE ALL BEEN CHANGED.
C
C
C CALLS  DAXPY, DDOT, AND DNRM2.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION DDOT, DNRM2
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION C, DEN, ONE, SS, WW, YY
        INTEGER I, INDEXR, INDXR2, J, K
        LOGICAL SKIPUP
C
C     SCALAR ARGUMENTS
C
        DOUBLE PRECISION ETA
        INTEGER N
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION  S(N), F0(N), F1(N), QT(N,N), R(N*(N+1)/2),
     &    W(N), T(N), TT(2)
C
C ***** END OF DECLARATIONS *****
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
        ONE = 1.0
        SKIPUP = .TRUE.
C
C ***** DEFINE T AND S SUCH THAT *****
C
```

```
C           A+ = Q*(R + T*ST).
C
C T = R*S.
C
        INDEXR = 1
        DO 10 I=1,N
          T(I) = DDOT(N-I+1,R(INDEXR),1,S(I),1)
          INDEXR = INDEXR + N - I + 1
  10    CONTINUE
C
C W = Y - Q*T  = Y - A*S.
C
        DO 20 I=1,N
          W(I) = F1(I) - FO(I) - DDOT(N,QT(1,I),1,T,1)
C
C         IF W(I) IS NOT SMALL, THEN UPDATE MUST BE PERFORMED,
C         OTHERWISE SET W(I) TO 0.
C
          IF (ABS(W(I)) .GT. ETA*(ABS(F1(I)) + ABS(FO(I)))) THEN
            SKIPUP = .FALSE.
          ELSE
            W(I) = 0.0
          END IF
  20    CONTINUE
C
C  IF NO UPDATE IS NECESSARY, THEN RETURN.
C
        IF (SKIPUP) RETURN
C
C T = QT*W = QT*Y - R*S.
C
        DO 30 I=1,N
          T(I) = DDOT(N,QT(I,1),N,W,1)
  30    CONTINUE
C
C S = S/(ST*S).
C
        DEN = 1.0/DDOT(N,S,1,S,1)
        CALL DSCAL(N,DEN,S,1)
C
C ***** END OF COMPUTATION OF  T & S      *****
C       AT THIS POINT,  A+ = Q*(R + T*ST).
C
        ENTRY R1UPQF(N,S,T,QT,R,W)
C
C ***** COMPUTE THE QR FACTORIZATION Q- R- OF (R + T*S).  THEN,  *****
C       Q+ = Q*Q-,  AND  R+ = R-.
C
C FIND THE LARGEST  K  SUCH THAT  T(K) .NE. 0.
C
```

```
      K = N
   50 IF (T(K) .NE. 0.0 .OR. K .LE. 1) GOTO 60
        K=K-1
        GOTO 50
   60 CONTINUE
C
C COMPUTE THE INDEX OF R(K-1,K-1).
C
      INDEXR = (N + N - K + 3)*(K - 2) / 2 + 1
C
C ***** TRANSFORM R+T*ST INTO AN UPPER HESSENBERG MATRIX *****
C
C DETERMINE JACOBI ROTATIONS WHICH WILL ZERO OUT ROWS
C N, N-1,...,2  OF THE MATRIX  T*ST,  AND APPLY THESE
C ROTATIONS TO  R.  (THIS IS EQUIVALENT TO APPLYING THE
C SAME ROTATIONS TO  R+T*ST, EXCEPT FOR THE FIRST ROW.
C THUS, AFTER AN ADJUSTMENT FOR THE FIRST ROW, THE
C RESULT IS AN UPPER HESSENBERG MATRIX.  THE
C SUBDIAGONAL ELEMENTS OF WHICH WILL BE STORED IN  W.
C
C NOTE:  ROWS N,N-1,...,K+1 ARE ALREADY ALL ZERO.
C
      DO 90 I=K-1,1,-1
C
C       DETERMINE THE JACOBI ROTATION WHICH WILL ZERO OUT
C       ROW  I+1  OF THE  T*ST  MATRIX.
C
        IF (T(I) .EQ. 0.0) THEN
          C = 0.0
C       SS = SIGN(-T(I+1))= -T(I+1)/|T(I+1)|
          SS = -SIGN(ONE,T(I+1))
        ELSE
          DEN = DNRM2(2,T(I),1)
          C = T(I) / DEN
          SS = -T(I+1)/DEN
        END IF
C
C       PREMULTIPLY  R  BY THE JACOBI ROTATION.
C
        YY = R(INDEXR)
        WW = 0.0
        R(INDEXR) = C*YY - SS*WW
        W(I+1) = SS*YY + C*WW
        INDEXR = INDEXR + 1
        INDXR2 = INDEXR + N - I
        DO 70 J= I+1,N
C         YY = R(I,J)
C         WW = R(I+1,J)
            YY = R(INDEXR)
            WW = R(INDXR2)
```

```
C           R(I,J) = C*YY - SS*WW
C           R(I+1,J) = SS*YY + C*WW
              R(INDEXR) = C*YY - SS*WW
              R(INDXR2) = SS*YY + C*WW
            INDEXR = INDEXR + 1
            INDXR2 = INDXR2 + 1
  70      CONTINUE
C
C         PREMULTIPLY  QT  BY THE JACOBI ROTATION.
C
          DO 80 J=1,N
            YY = QT(I,J)
            WW = QT(I+1,J)
            QT(I,J) = C*YY - SS*WW
            QT(I+1,J) = SS*YY + C*WW
  80      CONTINUE
C
C         UPDATE  T(I)  SO THAT  T(I)*ST(J)  IS THE  (I,J)TH  COMPONENT
C         OF  T*ST, PREMULTIPLIED BY ALL OF THE JACOBI ROTATIONS SO
C         FAR.
C
          IF (T(I) .EQ. 0.0) THEN
            T(I) = DABS(T(I+1))
          ELSE
            T(I) = DNRM2(2,T(I),1)
          END IF
C
C         LET INDEXR = THE INDEX OF R(I-1,I-1).
C
          INDEXR = INDEXR - 2*(N - I) - 3
C
  90    CONTINUE
C
C     UPDATE THE FIRST ROW OF  R  SO THAT  R  HOLDS  (R+T*ST)
C     PREMULTIPLIED BY ALL OF THE ABOVE JACOBI ROTATIONS.
C
        CALL DAXPY(N,T(1),S,1,R,1)
C
C ***** END OF TRANSFORMATION TO UPPER HESSENBERG *****
C
C
C ***** TRANSFORM UPPER HESSENBERG MATRIX INTO UPPER *****
C       TRIANGULAR MATRIX.
C
C       INDEXR = INDEX OF R(1,1).
C
          INDEXR = 1
          DO 120 I=1,K-1
C
C           DETERMINE APPROPRIATE JACOBI ROTATION TO ZERO OUT
```

```
C           R(I+1,I).
C
            IF (R(INDEXR) .EQ. 0.0) THEN
              C = 0.0
              SS = -SIGN(ONE,W(I+1))
            ELSE
              TT(1) = R(INDEXR)
              TT(2) = W(I+1)
              DEN = DNRM2(2,TT,1)
              C = R(INDEXR) / DEN
              SS = -W(I+1)/DEN
            END IF
C
C           PREMULTIPLY  R  BY JACOBI ROTATION.
C
            YY = R(INDEXR)
            WW = W(I+1)
            R(INDEXR) = C*YY - SS*WW
            W(I+1) = 0.0
            INDEXR = INDEXR + 1
            INDXR2 = INDEXR + N - I
            DO 100 J= I+1,N
C             YY = R(I,J)
C             WW = R(I+1,J)
                YY = R(INDEXR)
                WW = R(INDXR2)
C             R(I,J) = C*YY -SS*WW
C             R(I+1,J) = SS*YY + C*WW
                R(INDEXR) = C*YY - SS*WW
                R(INDXR2) = SS*YY + C*WW
              INDEXR = INDEXR + 1
              INDXR2 = INDXR2 + 1
  100       CONTINUE
C
C           PREMULTIPLY  QT  BY JACOBI ROTATION.
C
            DO 110 J=1,N
              YY = QT(I,J)
              WW = QT(I+1,J)
              QT(I,J) = C*YY - SS*WW
              QT(I+1,J) = SS*YY + C*WW
  110       CONTINUE
  120     CONTINUE
C
C ***** END OF TRANSFORMATION TO UPPER TRIANGULAR *****
C
C
C ***** END OF UPDATE *****
C
C
```

```
      RETURN
C
C ***** END OF SUBROUTINE UPQRQF *****
      END
```

Appendix B Sparse Algorithm Code Listing

```
      SUBROUTINE FIXPQS(N,NFE,IFLAG,LENQR,ARCRE,ARCAE,ANSRE,ANSAE,
     &     ARCLEN,A,Y,YP,YOLD,YPOLD,QR,PIVOT,PP,RHOVEC,ZO,DZ,T,WORK,
     &     SSPAR,PAR,IPAR)
C
C SUBROUTINE  FIXPQS  FINDS A FIXED POINT OR ZERO OF THE
C N-DIMENSIONAL VECTOR FUNCTION  F(X), OR TRACKS A ZERO CURVE OF A
C GENERAL HOMOTOPY MAP  RHO(A,X,LAMBDA).  FOR THE FIXED POINT PROBLEM
C F(X) IS ASSUMED TO BE A C2 MAP OF SOME BALL INTO ITSELF.  THE
C EQUATION  X=F(X)  IS SOLVED BY FOLLOWING THE ZERO CURVE OF THE
C HOMOTOPY MAP
C
C  LAMBDA*(X - F(X)) + (1 - LAMBDA)*(X - A),
C
C STARTING FROM  LAMBDA = 0, X = A.  THE CURVE IS PARAMETERIZED
C BY ARC LENGTH  S, AND IS FOLLOWED BY SOLVING THE ORDINARY
C DIFFERENTIAL EQUATION  D(HOMOTOPY MAP)/DS = 0  FOR
C Y(S) = (X(S),LAMBDA(S)).  THIS IS DONE BY USING A HERMITE CUBIC
C PREDICTOR AND A CORRECTOR WHICH RETURNS TO THE ZERO CURVE IN A
C HYPERPLANE PERPENDICULAR TO THE TANGENT TO THE ZERO CURVE AT THE
C MOST RECENT POINT.
C
C FOR THE ZERO FINDING PROBLEM  F(X)  IS ASSUMED TO BE A C2 MAP SUCH
C THAT FOR SOME  R > 0,  X*F(X) >= 0  WHENEVER  NORM(X) = R.
C THE EQUATION  F(X) = 0  IS SOLVED BY FOLLOWING THE ZERO CURVE OF
C THE HOMOTOPY MAP
C
C  LAMBDA*F(X) + (1 - LAMBDA)*(X - A)
C
C EMANATING FROM  LAMBDA = 0, X = A.
C
C A  MUST BE AN INTERIOR POINT OF THE ABOVE MENTIONED BALLS.
C
C FOR THE CURVE TRACKING PROBLEM RHO(A,X,LAMBDA) IS ASSUMED TO
C BE A C2 MAP FROM  E**M X [0,1) X E**N  INTO  E**N, WHICH FOR
C ALMOST ALL PARAMETER VECTORS  A  IN SOME NONEMPTY OPEN SUBSET
C OF E**M SATISFIES
C
C  RANK [D RHO(A,X,LAMBDA)/D LAMBDA, D RHO(A,X,LAMBDA)/DX] = N
C
C FOR ALL POINTS  (X,LAMBDA)  SUCH THAT  RHO(A,X,LAMBDA) = 0.  IT IS
C FURTHER ASSUMED THAT
C
C         RANK [ D RHO(A,XO,0)/DX ] = N.
C
C WITH  A  FIXED, THE ZERO CURVE OF  RHO(A,X,LAMBDA)  EMANATING FROM
C LAMBDA = 0, X = XO  IS TRACKED UNTIL  LAMBDA = 1  BY SOLVING THE
C ORDINARY DIFFERENTIAL EQUATION  D RHO(A,X(S),LAMBDA(S))/DS = 0
C FOR  Y(S) = (X(S),LAMBDA(S)), WHERE  S  IS ARC LENGTH ALONG THE
```

```
C ZERO CURVE.  ALSO THE HOMOTOPY MAP  RHO(A,X,LAMBDA)  IS ASSUMED TO
C BE CONSTRUCTED SUCH THAT
C
C         D LAMBDA(0)/DS > 0.
C
C FOR THE FIXED POINT AND ZERO FINDING PROBLEMS, THE USER MUST SUPPLY
C A SUBROUTINE  F(X,V)  WHICH EVALUATES F(X) AT X AND RETURNS THE
C VECTOR F(X) IN  V,  AND A SUBROUTINE  FJACS(X,QR,LENQR,PIVOT)  WHICH
C EVALUATES THE (SYMMETRIC) JACOBIAN MATRIX OF F(X) AT X, AND RETURNS
C THE SYMMETRIC JACOBIAN MATRIX IN PACKED SKYLINE STORAGE FORMAT IN  QR.
C LENQR  AND  PIVOT  DESCRIBE THE DATA STRUCTURE IN  QR.  FOR THE CURVE
C TRACKING PROBLEM, THE USER MUST SUPPLY A SUBROUTINE
C  RHO(A,LAMBDA,X,V,PAR,IPAR)  WHICH EVALUATES THE HOMOTOPY MAP RHO
C AT (A,X,LAMBDA) AND RETURNS THE VECTOR RHO(A,X,LAMBDA) IN  V,
C AND A SUBROUTINE  RHOJS(A,LAMBDA,X,QR,LENQR,PIVOT,PP,PAR,IPAR)  WHICH
C RETURNS IN  QR  THE SYMMETRIC N X N JACOBIAN MATRIX [D RHO/DX]
C EVALUATED AT (A,X,LAMBDA) AND STORED IN PACKED SKYLINE FORMAT,
C AND RETURNS IN  PP  THE VECTOR  -(D RHO/D LAMBDA)  EVALUATED AT
C (A,X,LAMBDA).  LENQR  AND  PIVOT  DESCRIBE THE DATA STRUCTURE IN
C QR.
C *** NOTE THE MINUS SIGN IN THE DEFINITION OF  PP. ***
C
C
C FIXPQS DIRECTLY OR INDIRECTLY USES THE SUBROUTINES  D1MACH, F
C (OR RHO), FJACS (OR RHOJS), GMFADS, MULTDS, PCGQS, ROOTQS, STEPQS,
C SOLVDS,  AND THE BLAS ROUTINES  DAXPY, DCOPY, DDOT, DNRM2, AND  DSCAL.
C ONLY  D1MACH  CONTAINS MACHINE DEPENDENT CONSTANTS.  NO OTHER
C MODIFICATIONS BY THE USER ARE REQUIRED.
C
C
C ON INPUT:
C
C N  IS THE DIMENSION OF X, F(X), AND RHO(A,X,LAMBDA).
C
C IFLAG  CAN BE -2, -1, 0, 2, OR 3.  IFLAG  SHOULD BE 0 ON THE FIRST
C    CALL TO  FIXPQS  FOR THE PROBLEM  X=F(X), -1 FOR THE PROBLEM
C    F(X)=0, AND -2 FOR THE PROBLEM  RHO(A,X,LAMBDA)=0.    IN CERTAIN
C    SITUATIONS  IFLAG  IS SET TO 2 OR 3 BY  FIXPQS, AND  FIXPQS  CAN
C    BE CALLED AGAIN WITHOUT CHANGING  IFLAG.
C
C LENQR  IS THE LENGTH OF THE N-DIMENSIONAL ARRAY  QR.  I.E.
C    IT IS THE NUMBER OF NON-ZERO ENTRIES IN THE JACOBIAN
C    MATRIX [DF/DX] (OR [D RHO/DX]).
C
C ARCRE, ARCAE  ARE THE RELATIVE AND ABSOLUTE ERRORS, RESPECTIVELY,
C    ALLOWED THE ITERATION ALONG THE ZERO CURVE.  IF
C    ARC?E .LE. 0.0  ON INPUT, IT IS RESET TO  .5*SQRT(ANS?E).
C    NORMALLY  ARC?E  SHOULD BE CONSIDERABLY LARGER THAN  ANS?E.
C
C ANSRE, ANSAE  ARE THE RELATIVE AND ABSOLUTE ERROR VALUES USED FOR
```

```
C     THE ANSWER AT  LAMBDA = 1.  THE ACCEPTED ANSWER  Y = (X,LAMBDA)
C     SATISFIES
C
C       |Y(1) - 1| .LE. ANSRE * ANSAE       .AND.
C
C       ||DZ|| .LE. ANSRE*||Y|| + ANSAE      WHERE
C
C       DZ IS THE NEWTON STEP TO Y.
C
C A(1:*)  CONTAINS THE PARAMETER VECTOR  A.  FOR THE FIXED POINT
C     AND ZERO FINDING PROBLEMS,  A  NEED NOT BE INITIALIZED BY THE
C     USER, AND IS ASSUMED TO HAVE LENGTH  N.  FOR THE CURVE
C     TRACKING PROBLEM,  A  MUST BE INITIALIZED BY THE USER.
C
C Y(1:N+1)  CONTAINS THE STARTING POINT FOR TRACKING THE HOMOTOPY MAP.
C     (Y(1),...,Y(N)) = A  FOR THE FIXED POINT AND ZERO FINDING
C     PROBLEMS.  (Y(1),...,Y(N)) = XO  FOR THE CURVE TRACKING PROBLEM.
C     Y(N+1)  NEED NOT BE DEFINED BY THE USER.
C
C YP(1:N+1)  IS A WORK ARRAY CONTAINING THE TANGENT VECTOR TO THE
C     ZERO CURVE AT THE CURRENT POINT  Y.
C
C YOLD(1:N+1) IS A WORK ARRAY CONTAINING THE PREVIOUS POINT FOUND
C     ON THE ZERO CURVE.
C
C YPOLD(1:N+1) IS A WORK ARRAY CONTAINING THE TANGENT VECTOR TO
C     THE ZERO CURVE AT  YOLD.
C
C QR(1:LENQR)  IS A WORK ARRAY CONTAINING THE N X N SYMMETRIC
C     JACOBIAN MATRIX WITH RESPECT TO X STORED IN PACKED SKYLINE
C     STORAGE FORMAT.  LENQR  AND  PIVOT  DESCRIBE THE DATA
C     STRUCTURE IN  QR.  (SEE SUBROUTINE  PCGQS  FOR A DESCRIPTION
C     OF THIS DATA STRUCTURE).
C
C PIVOT(1:N+2)  IS A WORK ARRAY WHOSE FIRST N+1 COMPONENTS CONTAIN
C     THE INDICES OF THE DIAGONAL ELEMENTS OF THE N X N SYMMETRIC
C     JACOBIAN MATRIX (WITH RESPECT TO X) WITHIN  QR.
C
C PP(1:N)  IS A WORK ARRAY CONTAINING THE NEGATIVE OF THE LAST COLUMN
C     OF THE JACOBIAN MATRIX  -[D RHO/D LAMBDA].
C
C RHOVEC(1:N+1), ZO(1:N+1), DZ(1:N+1), T(1:N+1)  ARE ALL WORK ARRAYS
C     USED BY  STEPQS, TANGQS, AND ROOTQS  TO CALCULATE THE TANGENT
C     VECTORS AND NEWTON STEPS.
C
C WORK(1:8*(N+1)+LENQR)  IS A WORK ARRAY USED BY THE CONJUGATE GRADIENT
C     ALGORITHM TO SOLVE LINEAR SYSTEMS.
C
C SSPAR(1:4) =  (HMIN, HMAX, BMIN, BMAX)  IS A VECTOR OF PARAMETERS
C     USED FOR THE OPTIMAL STEP SIZE ESTIMATION.  A DEFAULT VALUE
```

```
C    CAN BE SPECIFIED FOR ANY OF THESE FOUR PARAMETERS BY SETTING IT
C    .LE. 0.0  ON INPUT.  SEE THE COMMENTS IN  STEPQS  FOR MORE
C    INFORMATION ABOUT THESE PARAMETERS.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C N, LENQR, A  ARE UNCHANGED.
C
C NFE  IS THE NUMBER OF JACOBIAN EVALUATIONS.
C
C IFLAG =
C
C   1   NORMAL RETURN.
C
C   2   SPECIFIED ERROR TOLERANCE CANNOT BE MET.  SOME OR ALL OF
C       ARCRE, ARCAE, ANSRE, ANSAE  HAVE BEEN INCREASED TO
C       SUITABLE VALUES.  TO CONTINUE, JUST CALL  FIXPQS  AGAIN
C       WITHOUT CHANGING ANY PARAMETERS.
C
C   3   STEPQS  HAS BEEN CALLED 1000 TIMES.  TO CONTINUE, CALL
C       FIXPQS  AGAIN WITHOUT CHANGING ANY PARAMETERS.
C
C   4   JACOBIAN MATRIX DOES NOT HAVE FULL RANK.  THE ALGORITHM
C       HAS FAILED (THE ZERO CURVE OF THE HOMOTOPY MAP CANNOT BE
C       FOLLOWED ANY FURTHER).
C
C   5   THE TRACKING ALGORITHM HAS LOST THE ZERO CURVE OF THE
C       HOMOTOPY MAP AND IS NOT MAKING PROGRESS.  THE ERROR
C       TOLERANCES  ARC?E  AND  ANS?E  WERE TOO LENIENT.  THE PROBLEM
C       SHOULD BE RESTRARTED BY CALLING  FIXPQS  WITH SMALLER ERROR
C       TOLERANCES AND  IFLAG = 0 (-1, -2).
C
C   6   THE NEWTON ITERATION IN  STEPQS  OR  ROOTQS  FAILED TO
C       CONVERGE.  THE ERROR TOLERANCES  ANS?E  MAY BE TOO STRINGENT.
C
C   7   ILLEGAL INPUT PARAMETERS, A FATAL ERROR.
C
C ARCRE, ARCAE, ANSRE, ANSAE  ARE UNCHANGED AFTER A NORMAL RETURN
C    (IFLAG = 1).  THEY ARE INCREASED TO APPROPRIATE VALUES ON THE
C    RETURN  IFLAG = 2.
C
C ARCLEN  IS THE APPROXIMATE LENGTH OF THE ZERO CURVE.
C
C Y(N+1) = LAMBDA, (Y(1),...,Y(N)) = X, AND  Y  IS AN APPROXIMATE
C    ZERO OF THE HOMOTOPY MAP.  NORMALLY  LAMBDA = 1  AND  X  IS A
```

```
C    FIXED POINT OR ZERO OF  F(X).    IN ABNORMAL SITUATIONS,  LAMBDA
C    MAY ONLY BE NEAR 1 AND  X  NEAR A FIXED POINT OR ZERO.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION D1MACH, DNRM2
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION ABSERR, H, HOLD, RELERR, S, WK
        INTEGER IFLAGC, ITER, JW, LIMITD, LIMIT, NP1, PCGWK
        LOGICAL CRASH, START
C
C     SCALAR ARGUMENTS
C
        DOUBLE PRECISION ARCRE, ARCAE, ANSRE, ANSAE, ARCLEN
        INTEGER N, NFE, IFLAG, LENQR
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION A(N), Y(N+1), YP(N+1), YOLD(N+1), YPOLD(N+1),
     &    QR(LENQR), PP(N), RHOVEC(N+1), ZO(N+1), DZ(N+1), T(N+1),
     &    WORK(8*(N+1)+LENQR), SSPAR(4), PAR(1)
        INTEGER PIVOT(N+2), IPAR(1)
C
        SAVE
C
C ***** END OF DECLARATIONS *****
C
C LIMITD  IS AN UPPER BOUND ON THE NUMBER OF STEPS.  IT MAY BE
C CHANGED BY CHANGING THE FOLLOWING PARAMETER STATEMENT:
        PARAMETER (LIMITD =1000)
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C CHECK IFLAG
C
        IF (N .LE. 0 .OR. ANSRE .LE. 0.0 .OR. ANSAE .LT. 0.0)
     &    IFLAG = 7
        IF (IFLAG .GE. -2 .AND. IFLAG .LE. 0) GO TO 10
        IF (IFLAG .EQ. 2) GO TO 50
        IF (IFLAG .EQ. 3) GO TO 40
C
C ONLY VALID INPUT FOR IFLAG IS -2, -1, 0, 2, 3.
C
        IFLAG = 7
        RETURN
C
```

```
C ***** INITIALIZATION BLOCK  *****
C
  10    ARCLEN = 0.0
        IF (ARCRE .LE. 0.0) ARCRE = .5*SQRT(ANSRE)
        IF (ARCAE .LE. 0.0) ARCAE = .5*SQRT(ANSAE)
        NFE=0
        IFLAGC = IFLAG
        NP1=N+1
        PCGWK = 2*N+3
C
C SET INITIAL CONDITIONS FOR FIRST CALL TO STEPQS.
C
        START=.TRUE.
        CRASH=.FALSE.
        RELERR = ARCRE
        ABSERR = ARCAE
        HOLD=1.0
        H=0.1
        S=0.0
        YPOLD(NP1) = 1.0
        Y(NP1) = 0.0
        DO 20 JW=1,N
          YPOLD(JW)=0.0
  20    CONTINUE
C
C SET OPTIMAL STEP SIZE ESTIMATION PARAMETERS.
C
C     MINIMUM STEP SIZE HMIN
        IF (SSPAR(1) .LE. 0.0) SSPAR(1)= (SQRT(N+1.0)+4.0)*D1MACH(4)
C     MAXIMUM STEP SIZE HMAX
        IF (SSPAR(2) .LE. 0.0) SSPAR(2)= 1.0
C     MINIMUM STEP REDUCTION FACTOR BMIN
        IF (SSPAR(3) .LE. 0.0) SSPAR(3)= 0.1
C     MAXIMUM STEP EXPANSION FACTOR BMAX
        IF (SSPAR(4) .LE. 0.0) SSPAR(4)= 7.0
C
C LOAD  A  FOR THE FIXED POINT AND ZERO FINDING PROBLEMS.
C
        IF (IFLAGC .GE. -1) THEN
          CALL DCOPY(N,Y,1,A,1)
        ENDIF
C
  40    LIMIT=LIMITD
C
C ***** END OF INITIALIZATION BLOCK. *****
C
C ***** MAIN LOOP. *****
C
  50    DO 400 ITER=1,LIMIT
        IF (Y(NP1) .LT. 0.0) THEN
```

```
          ARCLEN = S
          IFLAG = 5
          RETURN
        END IF
C
C TAKE A STEP ALONG THE CURVE.
C
        CALL STEPQS(N,NFE,IFLAGC,LENQR,START,CRASH,HOLD,H,WK,RELERR,
     &    ABSERR,S,Y,YP,YOLD,YPOLD,A,QR,PIVOT,PP,RHOVEC,ZO,DZ,T,
     &    WORK,SSPAR,PAR,IPAR)
C
C CHECK IF THE STEP WAS SUCCESSFUL.
C
        IF (IFLAGC .GT. 0) THEN
          ARCLEN=S
          IFLAG=IFLAGC
          RETURN
        END IF
C
        IF (CRASH) THEN
C
C         RETURN CODE FOR ERROR TOLERANCE TOO SMALL.
C
          IFLAG=2
C
C         CHANGE ERROR TOLERANCES.
C
          IF (ARCRE .LT. RELERR) ARCRE=RELERR
          IF (ANSRE .LT. RELERR) ANSRE=RELERR
          IF (ARCAE .LT. ABSERR) ARCAE=ABSERR
          IF (ANSAE .LT. ABSERR) ANSAE=ABSERR
C
C         CHANGE LIMIT ON NUMBER OF ITERATIONS.
C
          LIMIT = LIMIT - ITER
          RETURN
        END IF
C
C IF  LAMBDA >= 1.0,  USE  ROOTQS  TO FIND SOLUTION.
C
        IF (Y(NP1) .GE. 1.0) GOTO 500
C
  400   CONTINUE
C
C ***** END OF MAIN LOOP *****
C
C DID NOT CONVERGE IN  LIMIT  ITERATIONS, SET  IFLAG  AND RETURN.
C
        ARCLEN = S
        IFLAG = 3
```

```
      RETURN
C
C ***** FINAL STEP -- FIND SOLUTION AT LAMBDA=1 *****
C
C SAVE  YOLD  FOR ARC LENGTH CALCULATION LATER.
C
  500 CALL DCOPY(NP1,YOLD,1,T,1)
C
C FIND SOLUTION.
C
      CALL ROOTQS(N,NFE,IFLAG,LENQR,ANSRE,ANSAE,Y,YP,YOLD,YPOLD,
     &    A,QR,PIVOT,PP,RHOVEC,ZO,DZ,WORK(PCGWK),PAR,IPAR)
C
C CHECK IF SOLUTION WAS FOUND AND SET  IFLAG  ACCORDINGLY.
C
      IFLAG=1
C
C     SET ERROR FLAG IF ROOTQS COULD NOT GET THE POINT ON THE ZERO
C     CURVE AT  LAMBDA = 1.0 .
C
      IF (IFLAGC .GT. 0) IFLAG=IFLAGC
C
C CALCULATE FINAL ARC LENGTH.
C
      CALL DCOPY(NP1,Y,1,DZ,1)
      WK=-1.0
      CALL DAXPY(NP1,WK,T,1,DZ,1)
      ARCLEN=S - HOLD + DNRM2(NP1,DZ,1)
C
C ***** END OF FINAL STEP *****
C
      RETURN
C
C ***** END OF SUBROUTINE FIXPQS *****
      END
      SUBROUTINE STEPQS(N,NFE,IFLAG,LENQR,START,CRASH,HOLD,H,
     &           WK,RELERR,ABSERR,S,Y,YP,YOLD,YPOLD,A,QR,PIVOT,PP,
     &           RHOVEC,ZO,DZ,T,WORK,SSPAR,PAR,IPAR)
C
C SUBROUTINE  STEPQS  TAKES ONE STEP ALONG THE ZERO CURVE OF THE
C HOMOTOPY MAP  RHO(X,LAMBDA)  USING A PREDICTOR-CORRECTOR ALGORITHM.
C THE PREDICTOR USES A HERMITE CUBIC INTERPOLANT, AND THE CORRECTOR
C RETURNS TO THE ZERO CURVE USING A NEWTON ITERATION, REMAINING
C IN A HYPERPLANE PERPENDICULAR TO THE MOST RECENT TANGENT VECTOR.
C  STEPQS  ALSO ESTIMATES A STEP SIZE  H  FOR THE NEXT STEP ALONG THE
C ZERO CURVE.
C
C
C ON INPUT:
C
```

```
C N = DIMENSION OF  X.
C
C NFE = NUMBER OF JACOBIAN MATRIX EVALUATIONS.
C
C IFLAG = -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C LENQR = THE LENGTH OF THE ONE DIMENSIONAL ARRAY  QR.
C
C START = .TRUE. ON FIRST CALL TO  STEPQS, .FALSE. OTHERWISE.
C         SHOULD NOT BE MODIFIED BY THE USER AFTER THE FIRST CALL.
C
C HOLD = ||Y - YOLD|| ; SHOULD NOT BE MODIFIED BY THE USER.
C
C H = UPPER LIMIT ON LENGTH OF STEP THAT WILL BE ATTEMPTED.  H  MUST
C    BE SET TO A POSITIVE NUMBER ON THE FIRST CALL TO  STEPQS.
C    THEREAFTER,  STEPQS  CALCULATES AN OPTIMAL VALUE FOR  H, AND  H
C    SHOULD NOT BE MODIFIED BY THE USER.
C
C WK = APPROXIMATE CURVATURE FOR THE LAST STEP (COMPUTED BY PREVIOUS
C    CALL TO  STEPQS).  UNDEFINED ON FIRST CALL.  SHOULD NOT BE
C    MODIFIED BY THE USER.
C
C RELERR, ABSERR = RELATIVE AND ABSOLUTE ERROR VALUES.  THE ITERATION
C    IS CONSIDERED TO HAVE CONVERGED WHEN A POINT  Z=(X,LAMBDA)  IS
C    FOUND SUCH THAT
C       ||DZ|| .LE. RELERR*||Z|| + ABSERR,
C    WHERE  DZ  IS THE LAST NEWTON STEP.
C
C S  = (APPROXIMATE) ARC LENGTH ALONG THE HOMOTOPY ZERO CURVE UP TO
C    Y(S) = (X(S),LAMBDA(S)).
C
C Y(1:N+1) = PREVIOUS POINT (X(S),LAMBDA(S)) FOUND ON THE ZERO CURVE
C    OF THE HOMOTOPY MAP.
C
C YP(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY
C    MAP AT  Y.  INPUT IN THIS VECTOR IS NOT USED ON THE FIRST CALL
C    TO  STEPQS.
C
C YOLD(1:N+1) = A POINT BEFORE  Y  ON THE ZERO CURVE OF THE HOMOTOPY
C    MAP.  INPUT IN THIS VECTOR IS NOT USED ON THE FIRST CALL TO
C    STEPQS.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE
C    HOMOTOPY MAP AT  YOLD.
C
C A(1:N) = PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QR(1:LENQR)  IS A WORK ARRAY CONTAINING THE N X N SYMMETRIC
C    JACOBIAN MATRIX WITH RESPECT TO X STORED IN PACKED SKYLINE
C    STORAGE FORMAT.  LENQR  AND  PIVOT  DESCRIBE THE DATA
```

```
C     STRUCTURE IN  QR.  (SEE SUBROUTINE  PCGQS  FOR A DESCRIPTION
C     OF THIS DATA STRUCTURE).
C
C PIVOT(1:N+2)  IS A WORK ARRAY WHOSE FIRST N+1 COMPONENTS CONTAIN
C     THE INDICES OF THE DIAGONAL ELEMENTS OF THE N X N SYMMETRIC
C     JACOBIAN MATRIX (WITH RESPECT TO X) WITHIN  QR.
C
C PP(1:N)  IS A WORK ARRAY CONTAINING THE NEGATIVE OF THE LAST COLUMN
C     OF THE JACOBIAN MATRIX  -[D RHO/D LAMBDA].
C
C RHOVEC(1:N+1), ZO(1:N+1), DZ(1:N+1), T(1:N+1)  ARE ALL WORK ARRAYS
C     USED BY  STEPQS, TANGQS, AND ROOTQS  TO CALCULATE THE TANGENT
C     VECTORS AND NEWTON STEPS.
C
C WORK(1:8*(N+1)+LENQR)  IS A WORK ARRAY USED BY THE CONJUGATE GRADIENT
C     ALGORITHM TO SOLVE LINEAR SYSTEMS.
C
C SSPAR(1:4) = PARAMETERS USED FOR COMPUTATION OF THE OPTIMAL STEP SIZE.
C     SSPAR(1) = HMIN, SSPAR(2) = HMAX, SSPAR(3) = BMIN, SSPAR(4) = BMAX.
C     THE OPTIMAL STEP  H  IS RESTRICTED SUCH THAT
C        HMIN .LE. H .LE. HMAX, AND  BMIN*HOLD .LE. H .LE. BMAX*HOLD.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C     WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C     RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C N, LENQR, A  ARE UNCHANGED.
C
C NFE HAS BEEN UPDATED.
C
C IFLAG
C
C     = -2, -1, OR 0 (UNCHANGED) ON A NORMAL RETURN.
C
C     = 4 IF A JACOBIAN MATRIX WITH RANK <  N  HAS OCCURRED.  THE
C        ITERATION WAS NOT COMPLETED.
C
C     = 6 IF THE ITERATION FAILED TO CONVERGE.
C
C START = .FALSE. ON A NORMAL RETURN.
C
C CRASH
C
C     = .FALSE. ON A NORMAL RETURN.
C
C     = .TRUE. IF THE STEP SIZE  H  WAS TOO SMALL.  H  HAS BEEN
C       INCREASED TO AN ACCEPTABLE VALUE, WITH WHICH  STEPQS  MAY BE
```

```
C      CALLED AGAIN.
C
C    = .TRUE. IF  RELERR  AND/OR  ABSERR  WERE TOO SMALL.  THEY HAVE
C      BEEN INCREASED TO ACCEPTABLE VALUES, WITH WHICH  STEPQS  MAY
C      BE CALLED AGAIN.
C
C HOLD = ||Y-YOLD||.
C
C H = OPTIMAL VALUE FOR NEXT STEP TO BE ATTEMPTED.  NORMALLY  H  SHOULD
C     NOT BE MODIFIED BY THE USER.
C
C WK = APPROXIMATE CURVATURE FOR THE STEP TAKEN BY  STEPQS.
C
C S = (APPROXIMATE) ARC LENGTH ALONG THE ZERO CURVE OF THE HOMOTOPY
C     MAP UP TO THE LATEST POINT FOUND, WHICH IS RETURNED IN  Y.
C
C RELERR, ABSERR  ARE UNCHANGED ON A NORMAL RETURN.  THEY ARE POSSIBLY
C     CHANGED IF  CRASH  = .TRUE. (SEE DESCRIPTION OF  CRASH  ABOVE).
C
C Y, YP, YOLD, YPOLD  CONTAIN THE TWO MOST RECENT POINTS AND TANGENT
C     VECTORS FOUND ON THE ZERO CURVE OF THE HOMOTOPY MAP.
C
C
C CALLS  D1MACH, DAXPY, DCOPY, DDOT, DNRM2, DSCAL, F (OR RHO), FJACS
C     (OR RHOJS), PCGQS, TANGQS.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION D1MACH, DDOT, DNRM2, QOFS
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION ALPHA, CORDIS, DDOO1, DDOO11,DDO1,DDO11,DELS,
     &    FOURU, GAMMA, HFAIL, HTEMP, IDLERR, LAMBDA, OMEGA, ONE, PO,
     &    P1, PPO, PP1, SIGMA, TEMP, THETA, TWOU, WKOLD, XSTEP
        INTEGER I, ITCNT, LITFH, J, JP1, LK, LST, NP1, PCGWK, ZU
        LOGICAL FAILED
C
C     SCALAR ARGUMENTS
C
        INTEGER N, NFE, IFLAG, LENQR
        LOGICAL START, CRASH
        DOUBLE PRECISION HOLD, H, WK, RELERR, ABSERR, S
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION Y(N+1), YP(N+1), YOLD(N+1), YPOLD(N+1),
     &    A(N), QR(LENQR), PP(N), RHOVEC(N+1), ZO(N+1), DZ(N+1),
```

```
     &    T(N+1), WORK(8*(N+1)+LENQR), SSPAR(4), PAR(1)
          INTEGER PIVOT(N+2), IPAR(1)
C
          REAL WRGE(8),ACOF(12)
          DATA WRGE  /
     &   .8735115E+00, .1531947E+00, .3191815E-01, .3339946E-10,
     &   .4677788E+00, .6970123E-03, .1980863E-05, .1122789E-08/
          DATA ACOF  /
     &   .9043128E+00,-.7075675E+00,-.4667383E+01,-.3677482E+01,
     &   .8516099E+00,-.1953119E+00,-.4830636E+01,-.9770528E+00,
     &   .1040061E+01, .3793395E-01, .1042177E+01, .4450706E-01/
C
          SAVE
C
C ***** END OF DECLARATIONS *****
C
C DEFINITION OF HERMITE CUBIC INTERPOLANT VIA DIVIDED DIFFERENCES.
C
          DD01(P0,P1,DELS) = (P1-P0)/DELS
          DD001(P0,PP0,P1,DELS) = (DD01(P0,P1,DELS)-PP0)/DELS
          DD011(P0,P1,PP1,DELS) = (PP1-DD01(P0,P1,DELS))/DELS
          DD0011(P0,PP0,P1,PP1,DELS) = (DD011(P0,P1,PP1,DELS) -
     &                                  DD001(P0,PP0,P1,DELS))/DELS
          QOFS(P0,PP0,P1,PP1,DELS,S) = ((DD0011(P0,PP0,P1,PP1,DELS)*
     &    (S-DELS) + DD001(P0,PP0,P1,DELS))*S + PP0)*S + P0
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C
C ***** INITIALIZATION *****
C
C LITFH = MAXIMUM NUMBER OF NEWTON ITERATIONS ALLOWED.
C
          ONE = 1.0
          TWOU = 2.0*D1MACH(4)
          FOURU = TWOU + TWOU
          NP1 = N+1
          FAILED = .FALSE.
          CRASH = .TRUE.
          LITFH = 10
          PCGWK = 2*N+3
          ZU = 3*N+4
C
C CHECK THAT ALL INPUT PARAMETERS ARE CORRECT.
C
C     THE ARCLENGTH  S  MUST BE NONNEGATIVE.
C
          IF (S .LT. 0.0) RETURN
C
C     IF STEP SIZE IS TOO SMALL, DETERMINE AN ACCEPTABLE ONE.
```

```
C
        IF (H .LT. FOURU*(1.0+S)) THEN
          H=FOURU*(1.0 + S)
          RETURN
        END IF
C
C     IF ERROR TOLERANCES ARE TOO SMALL, INCREASE THEM TO ACCEPTABLE
C     VALUES.
C
        TEMP=DNRM2(NP1,Y,1) + 1.0
        IF (.5*(RELERR*TEMP+ABSERR) .LT. TWOU*TEMP) THEN
          IF (RELERR .NE. 0.0) THEN
            RELERR = FOURU*(1.0+FOURU)
            TEMP = 0.0
            ABSERR = MAX(ABSERR,TEMP)
          ELSE
            ABSERR=FOURU*TEMP
          END IF
          RETURN
        END IF
C
C     INPUT PARAMETERS WERE ALL ACCEPTABLE.
C
        CRASH = .FALSE.
C
C COMPUTE  YP  ON FIRST CALL.
C
        IF (START) THEN
C
C         INITIALIZE THE IDEAL ERROR USED FOR STEP SIZE ESTIMATION.
C
          IDLERR=SQRT(SQRT(ABSERR))
C
C         INITIALIZE STARTING POINTS FOR THE CONJUGATE GRADIENT
C         ALGORITHM TO ZERO.
C
          DO 10 J=1,2*N+2
            WORK(J)=0.0
  10      CONTINUE
          CALL TANGQS(Y,YP,YPOLD,A,QR,PIVOT,PP,RHOVEC,WORK,
     &              N,LENQR,IFLAG,NFE,PAR,IPAR)
          IF (IFLAG .GT. 0) RETURN
        END IF
C
C ***** COMPUTE PREDICTOR POINT ZO *****
C
  20    IF (START) THEN
C
C         COMPUTE ZO WITH LINEAR PREDICTOR USING Y, YP --
C         ZO = Y+H*YP.
```

```
C
          CALL DCOPY(NP1,Y,1,ZO,1)
          CALL DAXPY(NP1,H,YP,1,ZO,1)
C
        ELSE
C
C         COMPUTE ZO WITH CUBIC PREDICTOR.
C
          DO 30 I=1,NP1
            ZO(I) = QOFS(YOLD(I),YPOLD(I),Y(I),YP(I),HOLD,HOLD+H)
  30      CONTINUE
C
        END IF
C
C ***** END OF PREDICTOR SECTION *****
C
C ***** NEWTON ITERATION *****
C
        DO 140 ITCNT = 1,LITFH
C
C SET STARTING POINTS FOR CONJUGATE GRADIENT ALGORITHM.
C
        DO 40 J=ZU,ZU+2*N+1
          WORK(J) = 0.0
  40    CONTINUE
C
C COMPUTE QR = [D RHO/DX], RHOVEC=RHO, -PP= (D RHO/D LAMBDA).
C
        LAMBDA = ZO(NP1)
        IF (IFLAG .EQ. -2) THEN
C
C         CURVE TRACKING PROBLEM.
C
          CALL RHOJS(A,LAMBDA,ZO,QR,LENQR,PIVOT,PP,PAR,IPAR)
          CALL RHO(A,LAMBDA,ZO,RHOVEC,PAR,IPAR)
        ELSE IF (IFLAG .EQ. -1) THEN
C
C         ZERO FINDING PROBLEM.
C
          CALL FJACS(ZO,QR,LENQR,PIVOT)
          CALL DSCAL(LENQR,LAMBDA,QR,1)
          SIGMA=1.0-LAMBDA
          DO 50 J=1,N
            QR(PIVOT(J))=QR(PIVOT(J))+SIGMA
  50      CONTINUE
          CALL DCOPY(N,ZO,1,RHOVEC,1)
          CALL DAXPY(N,-ONE,A,1,RHOVEC,1)
          CALL F(ZO,PP)
          CALL DSCAL(N,-ONE,PP,1)
          CALL DAXPY(N,ONE,RHOVEC,1,PP,1)
```

```
          CALL DAXPY(N,-LAMBDA,PP,1,RHOVEC,1)
        ELSE
C
C         FIXED POINT PROBLEM.
C
          CALL FJACS(ZO,QR,LENQR,PIVOT)
          CALL DSCAL(LENQR,-LAMBDA,QR,1)
          DO 60 J=1,N
            QR(PIVOT(J)) = QR(PIVOT(J))+1.0
  60      CONTINUE
          CALL DCOPY(N,ZO,1,RHOVEC,1)
          CALL DAXPY(N,-ONE,A,1,RHOVEC,1)
          CALL F(ZO,PP)
          CALL DAXPY(N,-ONE,A,1,PP,1)
          CALL DAXPY(N,-LAMBDA,PP,1,RHOVEC,1)
        END IF
        RHOVEC(NP1) = 0.0
        NFE = NFE+1
C
C SOLVE SYSTEM TO FIND NEWTON STEP  DZ.
C
          CALL PCGQS(N,QR,LENQR,PIVOT,PP,YP,RHOVEC,DZ,
     &          WORK(PCGWK),IFLAG)
          IF (IFLAG .GT. 0) RETURN
C
C TAKE STEP.
C
          CALL DAXPY(NP1, ONE, DZ, 1, ZO, 1)
C
C CHECK FOR CONVERGENCE.
C
          XSTEP=DNRM2(NP1,DZ,1)
          IF (XSTEP .LE. RELERR*DNRM2(NP1,ZO,1)+ABSERR)  THEN
             GO TO 160
          END IF
C
  140   CONTINUE
C
C ***** END OF NEWTON LOOP *****
C
C ***** DIDN'T CONVERGE OR TANGENT AT NEW POINT DID NOT MAKE *****
C       AN ANGLE SMALLER THAN 60 DEGREES WITH  YPOLD --
C       TRY AGAIN WITH A SMALLER H
C
  150   FAILED = .TRUE.
        HFAIL = H
        IF (H .LE. FOURU*(1.0 + S)) THEN
          IFLAG = 6
          RETURN
        ELSE
```

```
          H = .5 * H
        END IF
        GO TO 20
C
C ***** END OF CONVERGENCE FAILURE SECTION *****
C
C ***** CONVERGED -- MOP UP AND RETURN *****
C
C COMPUTE TANGENT AT ZO.
C
  160   CALL TANGQS(ZO,T,YP,A,QR,PIVOT,PP,RHOVEC,WORK,N,
     &        LENQR,IFLAG,NFE,PAR,IPAR)
        IF (IFLAG .GT. 0) RETURN
C
C CHECK THAT COMPUTED TANGENT  T  MAKES AN ANGLE NO LARGER THAN
C 60 DEGREES WITH CURRENT TANGENT  YP.  (I.E. COS OF ANGLE < .5)
C IF NOT, STEP SIZE WAS TOO LARGE, SO THROW AWAY ZO, AND TRY
C AGAIN WITH A SMALLER STEP.
C
        ALPHA = DDOT(NP1,T,1,YP,1)
        IF (ALPHA .LT. 0.5) GOTO 150
        ALPHA = ACOS(ALPHA)
C
C COMPUTE CORRECTOR DISTANCE.
C
        DO 170 I=1,NP1
          WORK(PCGWK+I-1)=QOFS(YOLD(I),YPOLD(I),Y(I),YP(I),HOLD,HOLD+H)
  170   CONTINUE
        CORDIS=DNRM2(NP1,WORK(PCGWK+I-1),1)
C
C SET UP VARIABLES FOR NEXT CALL.
C
        CALL DCOPY(NP1,Y,1,YOLD,1)
        CALL DCOPY(NP1,ZO,1,Y,1)
        CALL DCOPY(NP1,YP,1,YPOLD,1)
        CALL DCOPY(NP1,T,1,YP,1)
C
C UPDATE ARCLENGTH   S = S + ||Y-YOLD||.
C
        HTEMP = HOLD
        CALL DAXPY(NP1,-ONE,YOLD,1,ZO,1)
        HOLD = DNRM2(NP1,ZO,1)
        S = S+HOLD
C
C COMPUTE IDEAL ERROR FOR STEP SIZE ESTIMATION.
C
        OMEGA=ISTEP/CORDIS
        IF (ITCNT .LE. 1) THEN
          THETA = 8.0
        ELSE IF (ITCNT .EQ. 4) THEN
```

```
            THETA = 1.0
          ELSE IF (ITCNT .LT. 4) THEN
            LK = 4*ITCNT-7
            IF (OMEGA .GE. WRGE(LK)) THEN
              THETA = 1.0
            ELSE IF (OMEGA .GE. WRGE(LK+1)) THEN
              THETA = ACOF(LK) + ACOF(LK+1)*LOG(OMEGA)
            ELSE IF (OMEGA .GE. WRGE(LK+2)) THEN
              THETA = ACOF(LK+2) + ACOF(LK+3)*LOG(OMEGA)
            ELSE
              THETA = 8.0
            END IF
          ELSE IF (ITCNT .GE. 7) THEN
            THETA = 0.125
          ELSE
            LK = 4*ITCNT - 16
            IF (OMEGA .GT. WRGE(LK)) THEN
              LST = 2*ITCNT - 1
              THETA = ACOF(LST) + ACOF(LST+1)*LOG(OMEGA)
            ELSE
              THETA = 0.125
            END IF
          END IF
          IDLERR=THETA*IDLERR
C
C     IDLERR SHOULD BE NO BIGGER THAN 1/2 PREVIOUS STEP.
C
          IDLERR = MIN(.5*HOLD,IDLERR)
C
C COMPUTE OPTIMAL STEP SIZE.
C   WK = APPROXIMATE CURVATURE = 2*SIN(ALPHA/2) WHERE
C        ALPHA = ARCCOS(YP*YPOLD).
C   GAMMA = EXPECTED CURVATURE FOR NEXT STEP, COMPUTED BY
C        EXTRAPOLATING FROM CURRENT CURVATURE  WK, AND LAST
C        CURVATURE  WKOLD.  GAMMA  IS FURTHER REQUIRED TO BE
C        POSITIVE.
C
          WKOLD = WK
          WK = 2.0*ABS(SIN(.5*ALPHA))/HOLD
          IF (START) THEN
             GAMMA = WK
          ELSE
             GAMMA = WK + HOLD/(HOLD+HTEMP)*(WK-WKOLD)
          END IF
          GAMMA = MAX(GAMMA, 0.01*ONE)
          H = SQRT(2.0*IDLERR/GAMMA)
C
C     ENFORCE RESTRICTIONS ON STEP SIZE SO AS TO ENSURE STABILITY.
C        HMIN <= H <= HMAX, BMIN*HOLD <= H <= BMAX*HOLD.
C
```

```
      H = MIN(MAX(SSPAR(1),SSPAR(3)*HOLD,H),SSPAR(4)*HOLD,SSPAR(2))
      IF (FAILED) H = MIN(HFAIL,H)
      START = .FALSE.
C
C ***** END OF MOP UP SECTION *****
C
      RETURN
C
C ***** END OF SUBROUTINE STEPQS *****
      END
      SUBROUTINE TANGQS(Y,YP,YPOLD,A,QR,PIVOT,PP,RHOVEC,WORK,N,LENQR,
     &     IFLAG,NFE,PAR,IPAR)
C
C SUBROUTINE  TANGQS  COMPUTES THE UNIT TANGENT VECTOR  YP  TO THE
C ZERO CURVE OF THE HOMOTOPY MAP AT  Y  BY GENERATING THE AUGMENTED
C JACOBIAN MATRIX
C
C           --           --
C           |  D(RHO(Y))   |
C     AUG = |       T      |,   WHERE RHO IS THE HOMOTOPY MAP
C           |   YPOLD      |
C           --           --
C
C SOLVING THE SYSTEM
C                                  T
C         AUG*YPT = (0,0,...,0,1)    FOR YPT.
C
C AND FINALLY COMPUTING  YP = YPT/||YPT||.
C
C
C ON INPUT:
C
C Y(1:N+1) = CURRENT POINT (X(S), LAMBDA(S)).
C
C YP(1:N+1)  IS UNDEFINED ON INPUT.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR AT THE PREVIOUS POINT ON THE
C    ZERO CURVE OF THE HOMOTOPY MAP.
C
C A(1:N)  IS THE PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QR(1:LENQR)  IS A WORK ARRAY CONTAINING THE N X N SYMMETRIC
C    JACOBIAN MATRIX WITH RESPECT TO X STORED IN PACKED SKYLINE
C    STORAGE FORMAT.  LENQR  AND  PIVOT  DESCRIBE THE DATA
C    STRUCTURE IN  QR.  (SEE SUBROUTINE  PCGQS  FOR A DESCRIPTION
C    OF THIS DATA STRUCTURE).
C
C PIVOT(1:N+2)  IS A WORK ARRAY WHOSE FIRST N+1 COMPONENTS CONTAINI
C    THE INDICES OF THE DIAGONAL ELEMENTS OF THE N X N SYMMETRIC
C    JACOBIAN MATRIX (WITH RESPECT TO X) WITHIN  QR.
```

```
C
C PP(1:N)  IS A WORK ARRAY CONTAINING THE NEGATIVE OF THE LAST COLUMN
C    OF THE JACOBIAN MATRIX  -[D RHO/D LAMBDA].
C
C RHOVEC(1:N+1), IS A WORK ARRAY USED TO CALCULATE THE TANGENT
C    VECTOR.
C
C WORK(1:8*(N+1)+LENQR)  IS A WORK ARRAY USED BY THE CONJUGATE GRADIENT
C    ALGORITHM TO SOLVE LINEAR SYSTEMS.
C
C N  IS THE DIMENSION OF X, WHERE  Y=(X(S),LAMBDA(S)).
C
C LENQR  IS THE LENGTH OF THE ONE-DIMENSIONAL ARRAY  QR.
C
C IFLAG  IS -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C NFE  IS THE NUMBER OF JACOBIAN EVALUATIONS.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C Y, YPOLD, A, N, LENQR  ARE UNCHANGED.
C
C YP(1:N+1) CONTAINS THE NEW UNIT TANGENT VECTOR TO THE ZERO
C    CURVE OF THE HOMOTOPY MAP AT  Y(S) = (X(S),LAMBDA(S)).
C
C IFLAG  = -2, -1, OR 0, (UNCHANGED) ON A NORMAL RETURN.
C        = 4 IF THE AUGMENTED JACOBIAN MATRIX HAS RANK LESS THAN N+1.
C
C NFE  HAS BEEN INCREMENTED BY 1.
C
C
C CALLS  DCOPY, DNRM2, DSCAL, F (OR RHO IF IFLAG = -2), FJACS
C    (OR RHOJS, IF IFLAG = -2), PCGQS.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION DNRM2
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION LAMBDA, ONE, SIGMA, YPNRM
        INTEGER J, NP1, PCGWK, ZU
C
```

```
C     SCALAR ARGUMENTS
C
        INTEGER N, LENQR, IFLAG, NFE
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION Y(N+1), YP(N+1), YPOLD(N+1), A(N),
     &    QR(LENQR), PP(N), RHOVEC(N+1), WORK(8*(N+1)+LENQR),PAR(1)
        INTEGER PIVOT(N+2), IPAR(1)
C
C ***** END OF DECLARATIONS *****
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
        ONE = 1.0
        NFE = NFE + 1
        NP1 = N + 1
        LAMBDA = Y(NP1)
        PCGWK = 2*N+3
        ZU = 3*N+4
C
C ***** DEFINE THE AUGMENTED JACOBIAN MATRIX *****
C
C COMPUTE JACOBIAN MATRIX, STORE IT IN [QR|-PP].
C
        IF (IFLAG .EQ. -2) THEN
C
C         CURVE TRACKING PROBLEM.
C
          CALL RHOJS(A,LAMBDA,Y,QR,LENQR,PIVOT,PP,PAR,IPAR)
        ELSE IF (IFLAG .EQ. -1) THEN
C
C         ZERO FINDING PROBLEM.
C
          CALL F(Y,PP)
          CALL DSCAL(N,-ONE,PP,1)
          CALL DAXPY(N,ONE,Y,1,PP,1)
          CALL DAXPY(N,-ONE,A,1,PP,1)
          CALL FJACS(Y,QR,LENQR,PIVOT)
          CALL DSCAL(LENQR,LAMBDA,QR,1)
          SIGMA = 1.0-LAMBDA
          DO 10 J=1,N
            QR(PIVOT(J))=QR(PIVOT(J))+SIGMA
  10      CONTINUE
        ELSE
C
C         FIXED POINT PROBLEM
C
          CALL F(Y,PP)
          CALL DAXPY(N,-ONE,A,1,PP,1)
```

```
          CALL FJACS(Y,QR,LENQR,PIVOT)
          CALL DSCAL(LENQR,-LAMBDA,QR,1)
          DO 20 J=1,N
            QR(PIVOT(J))=QR(PIVOT(J)) + 1.0
  20      CONTINUE
        ENDIF
C
C ***** END OF DEFINITION OF AUGMENTED JACOBIAN MATRIX *****
C
C                                              T
C ***** SOLVE SYSTEM  AUG*YPT = (0,...,0,1)  *****
C
C INITIALIZE STARTING POINT FOR THE CONJUGATE GRADIENT ALGORITHM
C TO BE THE SOLUTIONS FROM THE PREVIOUS CALL TO  TANGQS.
C
        CALL DCOPY(2*NP1,WORK,1,WORK(ZU),1)
C
C RHOVEC = -(0,...,0,1)**T
C
        DO 30 J=1,N
          RHOVEC(J)=0.0
30      CONTINUE
        RHOVEC(NP1) = -1.0
C
C SOLVE SYSTEM.
C
        CALL PCGQS(N,QR,LENQR,PIVOT,PP,YPOLD,RHOVEC,YP,WORK(PCGWK),
     &       IFLAG)
        IF (IFLAG .GT. 0) RETURN
C
C NORMALIZE THE TANGENT.
C
        YPNRM = 1.0/DNRM2(NP1,YP,1)
        CALL DSCAL(NP1,YPNRM,YP,1)
C
C SAVE SOLUTIONS FROM CONJUGATE GRADIENT ALGORITHM FOR NEXT CALL
C TO TANGQS.
C
        CALL DCOPY(2*NP1,WORK(ZU),1,WORK,1)
C
        RETURN
C
C ***** END OF SUBROUTINE TANGQS *****
        END
        SUBROUTINE ROOTQS(N,NFE,IFLAG,LENQR,RELERR,ABSERR,Y,YP,YOLD,
     &     YPOLD,A,QR,PIVOT,PP,RHOVEC,Z,DZ,WORK,PAR,IPAR)
C
C ROOTQS  FINDS THE POINT  YBAR = (XBAR, 1)  ON THE ZERO CURVE OF THE
C HOMOTOPY MAP.  IT STARTS WITH TWO POINTS  YOLD=(XOLD,LAMBDAOLD)  AND
C Y=(X,LAMBDA)  SUCH THAT  LAMBDAOLD < 1 <= LAMBDA, AND ALTERNATES
```

```
C BETWEEN USING A SECANT METHOD TO FIND A PREDICTED POINT ON THE
C HYPERPLANE  LAMBDA=1, AND TAKING A NEWTON STEP TO RETURN TO THE
C ZERO CURVE OF THE HOMOTOPY MAP.
C
C
C ON INPUT:
C
C N = DIMENSION OF X.
C
C NFE = NUMBER OF JACOBIAN MATRIX EVALUATIONS.
C
C IFLAG = -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C LENQR = THE LENGTH OF THE ONE-DIMENSIONAL ARRAY  QR.
C
C RELERR, ABSERR = RELATIVE AND ABSOLUTE ERROR VALUES.  THE ITERATION IS
C    CONSIDERED TO HAVE CONVERGED WHEN A POINT Y=(X,LAMBDA)  IS FOUND
C    SUCH THAT
C
C    |Y(N+1) - 1| <= RELERR + ABSERR                AND
C
C    ||DZ|| <= RELERR*||Y|| + ABSERR,              WHERE
C
C    DZ  IS THE NEWTON STEP TO Y.
C
C Y(1:N+1) = POINT (X(S),LAMBDA(S))  ON ZERO CURVE OF HOMOTOPY MAP.
C
C YP(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY MAP
C    AT  Y.
C
C YOLD(1:N+1) = A POINT DIFFERENT FROM  Y  ON THE ZERO CURVE.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY
C    MAP AT  YOLD.
C
C A(1:*) = PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QR(1:LENQR)  IS A WORK ARRAY CONTAINING THE N X N SYMMETRIC
C    JACOBIAN MATRIX WITH RESPECT TO X STORED IN PACKED SKYLINE
C    STORAGE FORMAT.  LENQR  AND  PIVOT  DESCRIBE THE DATA
C    STRUCTURE IN  QR.  (SEE SUBROUTINE  PCGQS  FOR A DESCRIPTION
C    OF THIS DATA STRUCTURE).
C
C PIVOT(1:N+2)  IS A WORK ARRAY WHOSE FIRST N+1 COMPONENTS CONTAIN
C    THE INDICES OF THE DIAGONAL ELEMENTS OF THE N X N SYMMETRIC
C    JACOBIAN MATRIX (WITH RESPECT TO X) WITHIN  QR.
C
C PP(1:N)  IS A WORK ARRAY CONTAINING THE NEGATIVE OF THE LAST COLUMN
C    OF THE JACOBIAN MATRIX  -[D RHO/D LAMBDA].
C
```

```
C RHOVEC(1:N+1), Z(1:N+1), DZ(1:N+1)  ARE ALL WORK ARRAYS
C    USED TO CALCULATE THE NEWTON STEPS.
C
C WORK(1:6*(N+1)+LENQR)  IS A WORK ARRAY USED BY THE CONJUGATE GRADIENT
C    ALGORITHM TO SOLVE LINEAR SYSTEMS.
C
C PAR(1:*)  AND  IPAR(1:*)  ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C
C ON OUTPUT:
C
C N, LENQR, RELERR, ABSERR, A  ARE UNCHANGED.
C
C NFE  HAS BEEN UPDATED.
C
C IFLAG
C    = -2, -1, OR 0 (UNCHANGED) ON A NORMAL RETURN.
C
C    = 4 IF A SINGULAR JACOBIAN MATRIX HAS OCCURRED.  THE
C        ITERATION WAS NOT COMPLETED.
C
C    = 6 IF THE ITERATION FAILED TO CONVERGE.  Y  AND  YOLD  CONTAIN
C        THE LAST TWO POINTS OBTAINED BY NEWTON STEPS, AND  YP
C        CONTAINS A POINT OPPOSITE OF THE HYPERPLANE  LAMBDA=1  FROM  Y.
C
C Y  IS THE POINT ON THE ZERO CURVE OF THE HOMOTOPY MAP AT  LAMBDA = 1.
C
C YP,  AND  YOLD  CONTAIN POINTS NEAR THE SOLUTION.
C
C CALLS  D1MACH, DAXPY, DCOPY, DDOT, DNRM2, F (OR RHO),
C        FJACS (OR RHOJS), PCGQS, ROOT
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION D1MACH, DDOT, DNRM2, QOFS
C
C     LOCAL VARIABLES
C
        DOUBLE PRECISION AERR, DD001, DD0011, DD01, DD011, DELS,
     &     LAMBDA, ONE, P0, P1, PP0, PP1, QSOUT, RERR, S, SA, SB,
     &     SIGMA, SOUT, U, ZERO
        INTEGER ISTEP, I, J, LCODE, LIMIT, NP1, ZU
        LOGICAL BRACK
C
C     SCALAR ARGUMENTS
C
```

```
      DOUBLE PRECISION RELERR, ABSERR
      INTEGER N, NFE, IFLAG, LENQR
C
C     ARRAY DECLARATIONS
C
      DOUBLE PRECISION Y(N+1), YP(N+1), YOLD(N+1), YPOLD(N+1), A(N),
     &    QR(LENQR), PP(N), RHOVEC(N+1), Z(N+1), DZ(N+1),
     &    WORK(6*(N+1)+LENQR), PAR(1)
      INTEGER PIVOT(N+2), IPAR(1)
C
C ***** END OF DECLARATIONS *****
C
C
C DEFINITION OF HERMITE CUBIC INTERPOLANT VIA DIVIDED DIFFERENCES.
C
      DD01(P0,P1,DELS)=(P1-P0)/DELS
      DD001(P0,PP0,P1,DELS)=(DD01(P0,P1,DELS)-PP0)/DELS
      DD011(P0,P1,PP1,DELS)=(PP1-DD01(P0,P1,DELS))/DELS
      DD0011(P0,PP0,P1,PP1,DELS)=(DD011(P0,P1,PP1,DELS)  -
     &                            DD001(P0,PP0,P1,DELS))/DELS
      QOFS(P0,PP0,P1,PP1,DELS,S)=((DD0011(P0,PP0,P1,PP1,DELS)*
     &    (S-DELS) + DD001(P0,PP0,P1,DELS))*S  + PP0)*S + P0
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C ***** INITIALIZATION *****
C
C LIMIT = MAXIMUM NUMBER OF ITERATIONS ALLOWED.
C
      ONE=1.0
      ZERO=0.0
      U=D1MACH(4)
      RERR=MAX(RELERR,U)
      AERR=MAX(ABSERR,ZERO)
      NP1=N+1
      LIMIT = 2*(INT(-LOG10(AERR+RERR*DNRM2(NP1,Y,1)))+1)
      ZU=N+2
C
C ***** END OF INITIALIZATION BLOCK *****
C
C ***** COMPUTE FIRST INTERPOLANT WITH A HERMITE CUBIC *****
C
C FIND DISTANCE BETWEEN Y AND YOLD.  DZ=||Y-YOLD||.
C
      CALL DCOPY(NP1,Y,1,DZ,1)
      CALL DAXPY(NP1,-ONE,YOLD,1,DZ,1)
      DELS=DNRM2(NP1,DZ,1)
C
C USING TWO POINTS AND TANGENTS ON THE HOMOTOPY ZERO CURVE, CONSTRUCT
C THE HERMITE CUBIC INTERPOLANT Q(S).  THEN USE  ROOT  TO FIND THE  S
```

```
C CORRESPONDING TO  LAMBDA = 1.  THE TWO POINTS ON THE ZERO CURVE ARE
C ALWAYS CHOSEN TO BRACKET  LAMBDA=1, WITH THE BRACKETING INTERVAL
C ALWAYS BEING [0, DELS].
C
        SA=0.0
        SB=DELS
        LCODE=1
40      CALL ROOT(SOUT,QSOUT,SA,SB,RERR,AERR,LCODE)
          IF (LCODE .GT. 0) GO TO 50
          QSOUT=QOFS(YOLD(NP1),YPOLD(NP1),Y(NP1),YP(NP1),DELS,SOUT)
     &       - 1.0
          GO TO 40
C
C     IF  LAMBDA = 1  WERE BRACKETED,  ROOT  CANNOT FAIL.
C
50      IF (LCODE .GT. 2) THEN
          IFLAG=6
          RETURN
        ENDIF
C
C CALCULATE  Q(SA)  AS THE INITIAL POINT FOR A NEWTON ITERATION.
C
        DO 60 I=1,NP1
          Z(I)=QOFS(YOLD(I),YPOLD(I),Y(I),YP(I),DELS,SA)
60      CONTINUE
C
C ***** END OF CALCULATION OF CUBIC INTERPOLANT *****
C
C TANGENT INFORMATION  YPOLD  IS NO LONGER NEEDED.  HEREAFTER,  YPOLD
C REPRESENTS THE MOST RECENT POINT WHICH IS ON THE OPPOSITE SIDE OF
C LAMBDA=1  FROM  Y.
C
C ***** PREPARE FOR MAIN LOOP *****
C
        CALL DCOPY(NP1,YOLD,1,YPOLD,1)
C
C INITIALIZE  BRACK  TO INDICATE THAT THE POINTS  Y  AND  YOLD  BRACKET
C LAMBDA=1,  THUS YOLD = YPOLD.
C
        BRACK = .TRUE.
C
C ***** MAIN LOOP *****
C
        DO 300 ISTEP=1,LIMIT
C
C SET STARTING POINTS FOR CONJUGATE GRADIENT ALGORITHM TO ZERO.
C
          DO 70 J=ZU,ZU+2*N+1
            WORK(J) = 0.0
 70       CONTINUE
```

```
C
C COMPUTE NEWTON STEP.
C
C       COMPUTE QR = [D RHO/DX], RHOVEC = RHO, -PP = (D RHO/D LAMBDA).
C
          LAMBDA = Z(NP1)
          IF (IFLAG .EQ. -2) THEN
C
C           CURVE TRACKING PROBLEM.
C
            CALL RHOJS(A,LAMBDA,Z,QR,LENQR,PIVOT,PP,PAR,IPAR)
            CALL RHO(A,LAMBDA,Z,RHOVEC,PAR,IPAR)
          ELSE IF (IFLAG .EQ. -1) THEN
C
C           ZERO FINDING PROBLEM.
C
            CALL FJACS(Z,QR,LENQR,PIVOT)
            CALL DSCAL(LENQR,LAMBDA,QR,1)
            SIGMA = 1.0-LAMBDA
            DO 80 J=1,N
              QR(PIVOT(J))=QR(PIVOT(J))+SIGMA
 80         CONTINUE
            CALL DCOPY(N,Z,1,RHOVEC,1)
            CALL DAXPY(N,-ONE,A,1,RHOVEC,1)
            CALL F(Z,PP)
            CALL DSCAL(N,-ONE,PP,1)
            CALL DAXPY(N,ONE,RHOVEC,1,PP,1)
            CALL DAXPY(N,-LAMBDA,PP,1,RHOVEC,1)
          ELSE
C
C           FIXED POINT PROBLEM.
C
            CALL FJACS(Z,QR,LENQR,PIVOT)
            CALL DSCAL(LENQR,-LAMBDA,QR,1)
            DO 90 J=1,N
              QR(PIVOT(J))=QR(PIVOT(J))+1.0
 90         CONTINUE
            CALL DCOPY(N,Z,1,RHOVEC,1)
            CALL DAXPY(N,-ONE,A,1,RHOVEC,1)
            CALL F(Z,PP)
            CALL DAXPY(N,-ONE,A,1,PP,1)
            CALL DAXPY(N,-LAMBDA,PP,1,RHOVEC,1)
          END IF
          RHOVEC(NP1) = 0.0
          NFE = NFE+1
C
C       SOLVE SYSTEM TO FIND NEWTON STEP.
C
          CALL PCGQS(N,QR,LENQR,PIVOT,PP,YP,RHOVEC,DZ,WORK,IFLAG)
          IF (IFLAG .GT. 0) RETURN
```

```
C
C       TAKE NEWTON STEP.
C
          CALL DAXPY(NP1,ONE,DZ,1,Z,1)
C
C       CHECK FOR CONVERGENCE.
C
          IF ((ABS(Z(NP1)-1.0) .LE. RERR*AERR) .AND.
     &        (DNRM2(NP1,DZ,1) .LE. RERR*DNRM2(N,Z,1)+AERR)) THEN
            RETURN
          END IF
C
C PREPARE FOR NEXT ITERATION.
C
C       IF LAMBDA COMPONENT OF  Z=1, THEN DO NOT COMPUTE A
C       NEW PREDICTOR, BUT RATHER CONTINUE WITH ANOTHER NEWTON
C       ITERATION.
C
          IF (ABS(Z(NP1)-1.0) .LT. RERR*AERR) GOTO 300
C
C       UPDATE  Y  AND  YOLD.
C
          CALL DCOPY(NP1,Y,1,YOLD,1)
          CALL DCOPY(NP1,Z,1,Y,1)
C
C       UPDATE  YPOLD  SUCH THAT  YPOLD  IS THE MOST RECENT POINT OPPOSITE
C       OF  LAMBDA=1  FROM  Y.  SET  BRACK = .TRUE.  IFF  Y & YOLD
C       BRACKET  LAMBDA=1  SO THAT  YPOLD=YOLD.
C
          IF ((YOLD(NP1)-1.0)*(Y(NP1)-1.0)  .GT. 0) THEN
            BRACK = .FALSE.
          ELSE
            BRACK = .TRUE.
            CALL DCOPY(NP1,YOLD,1,YPOLD,1)
          END IF
C
C       COMPUTE DELS = ||Y-YPOLD||.
C
          CALL DCOPY(NP1,Y,1,DZ,1)
          CALL DAXPY(NP1,-ONE,YPOLD,1,DZ,1)
          DELS=DNRM2(NP1,DZ,1)
C
C       COMPUTE  DZ  FOR THE LINEAR PREDICTOR   Z = DZ + Y,
C           WHERE  DZ = SA*(YOLD-Y).
C
          SA = (1.0-Y(NP1))/(YOLD(NP1)-Y(NP1))
          CALL DCOPY(NP1,YOLD,1,DZ,1)
          CALL DAXPY(NP1,-ONE,Y,1,DZ,1)
          CALL DSCAL(NP1,SA,DZ,1)
C
```

```
C       TO INSURE STABILITY, THE LINEAR PREDICTION MUST BE NO FARTHER
C       FROM  Y  THAN  YPOLD  IS.  THIS IS GUARANTEED IF  BRACK = .TRUE.
C       IF LINEAR PREDICTION IS TOO FAR AWAY, USE BRACKETING POINTS
C       TO COMPUTE LINEAR PREDICTION.
C
          IF (.NOT. BRACK) THEN
            IF (DNRM2(NP1,DZ,1)  .GT. DELS) THEN
C
C             COMPUTE DZ = SA*(YPOLD-Y).
C
              SA = (1.0-Y(NP1))/(YPOLD(NP1)-Y(NP1))
              CALL DCOPY(NP1,YPOLD,1,DZ,1)
              CALL DAXPY(NP1,-ONE,Y,1,DZ,1)
              CALL DSCAL(NP1,SA,DZ,1)
            END IF
          END IF
C
C       COMPUTE PREDICTOR Z = DZ+Y.
C
          CALL DCOPY(NP1,Y,1,Z,1)
          CALL DAXPY(NP1,ONE,DZ,1,Z,1)
  300   CONTINUE
C
C ***** END OF MAIN LOOP. *****
C
C THE ALTERNATING OSCULATORY LINEAR PREDICTION AND NEWTON
C CORRECTION HAS NOT CONVERGED IN  LIMIT  STEPS.  ERROR RETURN.
        IFLAG=6
        RETURN
C
C ***** END OF SUBROUTINE ROOTQS *****
        END
      SUBROUTINE PCGQS(NN,AA,LENAA,MAXA,PP,YP,RHO,START,WORK,IFLAG)
C
C THIS SUBROUTINE SOLVES A SYSTEM OF EQUATION USING THE METHOD OF
C CONJUGATE GRADIENTS.  THE SYSTEM TO BE SOLVED IS IN THE FORM
C
C   (AUG)*X = B,
C
C WHERE
C
C             +--           --+         +-     -+
C             |         |     |         |       |
C             |   AA    | -PP |         |       |
C       AUG = |         |     | ,   B = | -RHO  |
C             +---------------+         |       |
C             |      YP       |         |       |
C             +--           --+         +-     -+
C
C
```

```
C
C THE SYSTEM IS SOLVED BY SPLITTING  AUG  INTO TWO MATRICES
C AUG = M + L,  WHERE
C
C       +-        -+                                  +-     -+
C       |      |   |                                  |       |
C       |  AA  | C |                                  | -PP-C |
C   M = |      |   |  ,  L = U * [E(NN+1)**T],  U =   |       | ,
C       +------+---+                                  +-------+
C       |  C   | D |                                  |   0   |
C       +-        -+                                  +-     -+
C
C E(NN+1) IS THE (NN+1) X 1 VECTOR CONSISTING OF ALL ZEROS EXCEPT FOR
C A '1' IN ITS LAST POSITION.
C
C THE FINAL SOLUTION VECTOR,  X,  IS GIVEN BY
C
C            +-                                      -+
C            |           [SOL(U)]*[E(NN+1)**T]        |
C       X =  | I  -  ------------------------------  | * SOL(B)
C            |        {[(SOL(U))**T]*E(NN+1)}+1.0     |
C            +-                                      -+
C
C WHERE SOL(A)=[M**(-1)]*A.  THE TWO SYSTEMS  (MZ=U, MZ=B) ARE SOLVED
C BY A PRECONDITIONED CONJUGATE GRADIENT ALGORITHM.
C
C
C ON INPUT:
C
C NN  = THE DIMENSION OF THE MATRIX PACKED IN AA.
C
C AA(1:LENAA)  CONTAINS THE MATRIX  AA,  STORED IN PACKED SKYLINE
C    FORMAT.  LENAA  AND  MAXA  DESCRIBE THE DATA STRUCTURE.
C
C LENAA  = THE LENGTH OF THE ONE-DIMENSIONAL ARRAY  AA.
C
C MAXA(1:NN+2)  IN ITS FIRST  N+1  COMPONENTS CONTAINS THE INDICES OF
C    THE DIAGONAL ELEMENTS OF THE MATRIX STORED IN  AA.
C    MAXA(NN+2)  IS ASSIGNED THE VALUE  LENNAA + NN + 2.
C
C    AS AN EXAMPLE OF THE PACKED SKYLINE STORAGE FORMAT, CONSIDER THE
C    SYMMETRIC MATRIX
C
C
C                +--                 --+
C                |  1  3  0  0  0  |
C                |  3  2  0  7  0  |
C                |  0  0  4  6  0  |
C                |  0  7  6  5  9  |
C                |  0  0  0  9  8  |
```

```
C                +--             --+
C
C     THIS WOULD RESULT IN  NN=5, LENAA=9, MAXA=(1,2,4,5,8,10,16),
C     AND  AA=(1,2,3,4,5,6,7,8,9).
C
C PP(1:NN)  = THE NEGATIVE OF THE LAST COLUMN OF  AUG.
C
C YP(1:NN+1)  = THE LAST ROW OF  AUG.
C
C RHO(1:NN+1)  = THE NEGATIVE OF THE RIGHT HAND SIDE OF THE EQUATION TO
C     BE SOLVED.
C
C WORK(1:6*(NN+1)+LENAA)  IS A WORK ARRAY DIVIDED UP AS FOLLOWS:
C
C     WORK(1:NN+1) = TEMPORARY WORKING STORAGE.
C
C     WORK(NN+2:2*NN+2) = INTERMEDIATE SOLUTION VECTOR Z FOR  MZ=U.
C        THE INPUT VALUE IS USED AS THE INITIAL ESTIMATE FOR Z.
C
C     WORK(2*NN+3:3*NN+3) = INTERMEDIATE SOLUTION VECTOR Z FOR  MZ=B.
C
C     WORK(3*NN+4:4*NN+4) = STORAGE FOR THE RESIDUAL VECTORS.
C
C     WORK(4*NN+5:5*NN+5) = STORAGE FOR THE DIRECTION VECTORS.
C
C     WORK(5*NN+6:6*NN+6+LENAA) = STORAGE FOR THE PRECONDITIONING
C        MATRIX Q.
C
C
C ON OUTPUT:
C
C NN, AA, LENAA, MAXA, PP, YP, AND RHO  ARE UNCHANGED.
C
C START(1:N+1)  CONTAINS THE SOLUTION VECTOR  X.
C
C IFLAG  IS UNCHANGED UNLESS THE CONJUGATE GRADIENT ITERATION
C     FAILS TO CONVERGE IN 10*(NN+1) ITERATIONS (MOST LIKELY DUE
C     TO A SINGULAR JACOBIAN MATRIX).  IN THIS CASE,  PCGQS  RETURNS
C     IFLAG = 4, AND DOES NOT COMPUTE  X.
C
C
C CALLS  D1MACH, DAXPY, DCOPY, DDOT, DNRM2, DSCAL, GMFADS, MULTDS,
C     SOLVDS.
C
C ***** DECLARATIONS *****
C
C     FUNCTION DECLARATIONS
C
        DOUBLE PRECISION D1MACH, DDOT, DNRM2
C
```

```
C     LOCAL VARIABLES
C
        DOUBLE PRECISION AB, AU, BB, BU, DZNRM, PBNPRD, PUNPRD,
     &     RBNPRD, RBTOL, RNPRD, RUNPRD, RUTOL, TEMP, UNRM, ZLEN, ZTOL
        INTEGER DIR, I, IMAX, J, K, LENQ, NP1, Q, RES, ZB, ZU
        LOGICAL STILLU, STILLB
C
C     SCALAR ARGUMENTS
C
        INTEGER NN, LENAA, IFLAG
C
C     ARRAY DECLARATIONS
C
        DOUBLE PRECISION AA(LENAA), PP(NN), YP(NN+1), RHO(NN+1),
     &     START(NN+1), WORK(6*(NN+1)+LENAA)
        INTEGER MAXA(NN+2)
C
C ***** END OF DECLARATIONS *****
C
C ***** FIRST EXECUTABLE STATEMENT *****
C
C SET UP BASES FOR VECTORS STORED IN WORK ARRAY.
C
      NP1=NN+1
      ZU=NN+2
      ZB=(2*NN)+3
      RES=(3*NN)+4
      DIR=(4*NN)+5
      Q=(5*NN)+6
C
C INITIALIZE  PRECONDITIONING MATRIX  Q,  SET VALUES OF  MAXA(NN+1)
C AND  MAXA(NN+2),  COMPUTE PRECONDITIONER.
C
      CALL DCOPY(LENAA,AA,1,WORK(Q),1)
      CALL DCOPY(NP1,YP,1,WORK(Q+LENAA),1)
      MAXA(NN+1)=LENAA+1
      MAXA(NN+2)=LENAA+NN+2
      LENQ = MAXA(NN+2)-1
      CALL GMFADS(NP1,WORK(Q),LENQ,MAXA)
C
C COMPUTE ALL TOLERANCES NEEDED FOR EXIT CRITERIA.
C
      CALL DCOPY(NN,PP,1,WORK,1)
      WORK(NP1)=0.0D0
      CALL DAXPY(NP1,1.0D0,YP,1,WORK,1)
      RUTOL=ZTOL*DNRM2(NP1,WORK,1)
      RBTOL=ZTOL*DNRM2(NP1,RHO,1)
      IMAX=10*NP1
      STILLU=.TRUE.
      STILLB=.TRUE.
```

```
      ZTOL=100.0*D1MACH(4)
C
C ***** END OF INITIALIZATION *****
C
C ***** SOLVE SYSTEM  M Z = U *****
C
C COMPUTE INITIAL RESIDUAL VECTOR FOR THE SYSTEM M Z = U .
C     RES = (Q**(-T))*(U - M*Z.)

      CALL MULTDS(WORK(RES),AA,WORK(ZU),MAXA,NN,LENAA)
      WORK(RES+NN)= DDOT(NN,YP,1,WORK(ZU),1)
      CALL DAXPY(NP1,WORK(ZU+NN),YP,1,WORK(RES),1)
      CALL DSCAL(NP1,-1.0D0,WORK(RES),1)
      CALL DAXPY(NN,-1.0D0,PP,1,WORK(RES),1)
      CALL DAXPY(NN,-1.0D0,YP,1,WORK(RES),1)
      CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK(RES))
C
C COMPUTE INITIAL DIRECTION VECTOR.
C     DIR = (A**T)*(Q**(-T))*RES.
C
      CALL DCOPY(NP1,WORK(RES),1,WORK,1)
      CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
      CALL MULTDS(WORK(DIR),AA,WORK,MAXA,NN,LENAA)
      WORK(DIR+NN)=DDOT(NN,YP,1,WORK,1)
      CALL DAXPY(NP1,WORK(NP1),YP,1,WORK(DIR),1)
C
C COMPUTE INITIAL INNER PRODUCTS.
C
      RUNPRD=DDOT(NP1,WORK(RES),1,WORK(RES),1)
      PUNPRD=DDOT(NP1,WORK(DIR),1,WORK(DIR),1)
C
C REPEAT UNTIL CONVERGENCE OR TOO MANY ITERATIONS.
C
      J=1
C
C     DO WHILE ((STILLU) .AND. (J .LE. IMAX))
100   IF (.NOT. ((STILLU) .AND. (J .LE. IMAX)) ) GO TO 200
C
C        IF ||RESIDUAL|| IS STILL NOT SMALL ENOUGH, CONTINUE.
C
         IF (SQRT(RUNPRD) .GT. RUTOL) THEN
C
C           IF DIRECTION VECTOR IS ZERO, THEN RE-COMPUTE RESIDUAL,
C           DIRECTION VECTOR, AND INNER PRODUCTS FROM SCRATCH
C           (RATHER THAN FROM UPDATES OF PREVIOUS VALUES).
C
            IF (PUNPRD .EQ. 0.0) THEN
C
C              COMPUTE RESIDUAL.
C
```

```
            CALL MULTDS(WORK(RES),AA,WORK(ZU),MAXA,NN,LENAA)
            WORK(RES+NN)= DDOT(NN,YP,1,WORK(ZU),1)
            CALL DAXPY(NP1,WORK(ZU+NN),YP,1,WORK(RES),1)
            CALL DSCAL(NP1,-1.0D0,WORK(RES),1)
            CALL DAXPY(NN,-1.0D0,PP,1,WORK(RES),1)
            CALL DAXPY(NN,-1.0D0,YP,1,WORK(RES),1)
            CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK(RES))
C
C           COMPUTE DIRECTION VECTOR.
C
            CALL DCOPY(NP1,WORK(RES),1,WORK,1)
            CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
            CALL MULTDS(WORK(DIR),AA,WORK,MAXA,NN,LENAA)
            WORK(DIR+NN)=DDOT(NN,YP,1,WORK,1)
            CALL DAXPY(NP1,WORK(NP1),YP,1,WORK(DIR),1)
C
C           COMPUTE INNER PRODUCTS
C
            RUNPRD=DDOT(NP1,WORK(RES),1,WORK(RES),1)
            PUNPRD=DDOT(NP1,WORK(DIR),1,WORK(DIR),1)
C
C           CHECK FOR CONVERGENCE.
C
            IF (SQRT(RUNPRD) .LE. RUTOL) THEN
               STILLU=.FALSE.
            ENDIF
         ENDIF
         IF (STILLU) THEN
C
C           UPDATE SOLUTION VECTOR.
C             Z = Z + AU*DIR, WHERE AU= RUNPRD/PUNPRD.
C
            AU=RUNPRD/PUNPRD
            CALL DAXPY(NP1,AU,WORK(DIR),1,WORK(ZU),1)
C
C           COMPUTE RELATIVE CHANGE IN THE SOLUTION.
C
            DZNRM=AU*SQRT(PUNPRD)
            ZLEN=DNRM2(NP1,WORK(ZU),1)
C
C           IF RELATIVE CHANGE IN SOLUTIONS IS SMALL ENOUGH, EXIT.
C
            IF ( (DZNRM/ZLEN) .LT. ZTOL) STILLU=.FALSE.
         ENDIF
      ELSE
         STILLU=.FALSE.
      ENDIF
C
C     IF NO EXIT CRITERIA FOR  MZ=U  HAVE BEEN MET, UPDATE RESIDUAL,
C       DIRECTION VECTORS, AND INNER PRODUCTS FOR NEXT ITERATION.
```

```
C
         IF (STILLU) THEN
C
C           UPDATE RESIDUAL VECTOR; COMPUTE <RES,RES>.
C             RES = RES - AU*(Q**(-1))*M*DIR.
C
            CALL MULTDS(WORK,AA,WORK(DIR),MAXA,NN,LENAA)
            WORK(NP1)=DDOT(NN,YP,1,WORK(DIR),1)
            CALL DAXPY(NP1,WORK(DIR+NN),YP,1,WORK,1)
            CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
            CALL DAXPY(NP1,-AU,WORK,1,WORK(RES),1)
            RNPRD=DDOT(NP1,WORK(RES),1,WORK(RES),1)
C
C           UPDATE DIRECTION VECTOR; COMPUTE <DIR,DIR>.
C             DIR = (M**T)*(Q**(-T))*RES + BU*DIR,
C             WHERE  BU = RNPRD/RUNPRD. (NOTE: START IS USED AS
C             A WORK ARRAY HERE).
C
            BU=RNPRD/RUNPRD
            RUNPRD=RNPRD
            CALL DCOPY(NP1,WORK(RES),1,WORK,1)
            CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
            CALL MULTDS(START,AA,WORK,MAXA,NN,LENAA)
            START(NP1)=DDOT(NN,YP,1,WORK,1)
            CALL DAXPY(NP1,WORK(NP1),YP,1,START,1)
            CALL DAXPY(NP1,BU,WORK(DIR),1,START,1)
            CALL DCOPY(NP1,START,1,WORK(DIR),1)
            PUNPRD=DDOT(NP1,WORK(DIR),1,WORK(DIR),1)
         ENDIF
C
         J=J+1
      GO TO 100
200   CONTINUE
C     END DO
C
C SET ERROR FLAG IF THE CONJUGATE GRADIENT ITERATION DID NOT CONVERGE.
C
      IF (J .GT. IMAX) THEN
         IFLAG=4
         RETURN
      ENDIF
C
C ***** END OF M Z = U SYSTEM *****
C
C ***** SOLVE SYSTEM M Z = B *****
C
C
C COMPUTE INITIAL RESIDUAL VECTOR FOR THE SYSTEM M Z = B .
C
      CALL MULTDS(WORK(RES),AA,WORK(ZB),MAXA,NN,LENAA)
```

```
      WORK(RES+NN)=DDOT(NN,YP,1,WORK(ZB),1)
      CALL DAXPY(NP1,WORK(ZB+NN),YP,1,WORK(RES),1)
      CALL DSCAL(NP1,-1.0D0,WORK(RES),1)
      CALL DAXPY(NP1,-1.0D0,RHO,1,WORK(RES),1)
      CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK(RES))
C
C COMPUTE INITIAL DIRECTION VECTOR.
C
      CALL DCOPY(NP1,WORK(RES),1,WORK,1)
      CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
      CALL MULTDS(WORK(DIR),AA,WORK,MAXA,NN,LENAA)
      WORK(DIR+NN)=DDOT(NN,YP,1,WORK,1)
      CALL DAXPY(NP1,WORK(NP1),YP,1,WORK(DIR),1)
C
C COMPUTE INITIAL INNER PRODUCTS.
C
      RBNPRD=DDOT(NP1,WORK(RES),1,WORK(RES),1)
      PBNPRD=DDOT(NP1,WORK(DIR),1,WORK(DIR),1)
C
C REPEAT UNTIL CONVERGENCE, OR TOO MANY ITERATIONS.
C
      J=1
C
C     DO WHILE ( STILLB  .AND. (J .LE. IMAX) )
300   IF (.NOT. ( STILLB  .AND. (J .LE. IMAX) ) ) GO TO 400
C
C        IF ||RESIDUAL|| IS STILL NOT SMALL ENOUGH, CONTINUE.
C
         IF (SQRT(RBNPRD) .GT. RBTOL) THEN
C
C           IF DIRECTION VECTOR IS ZERO, RE-COMPUTE RESIDUAL,
C             DIRECTION VECTOR, AND INNER PRODUCTS FROM SCRATCH.
C
            IF (PBNPRD .EQ. 0.0) THEN
C
C              COMPUTE RESIDUAL.
C
               CALL MULTDS(WORK(RES),AA,WORK(ZB),MAXA,NN,LENAA)
               WORK(RES+NN)=DDOT(NN,YP,1,WORK(ZB),1)
               CALL DAXPY(NP1,WORK(ZB+NN),YP,1,WORK(RES),1)
               CALL DSCAL(NP1,-1.0D0,WORK(RES),1)
               CALL DAXPY(NP1,-1.0D0,RHO,1,WORK(RES),1)
               CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK(RES))
C
C              COMPUTE DIRECTION VECTOR.
C
               CALL DCOPY(NP1,WORK(RES),1,WORK,1)
               CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
               CALL MULTDS(WORK(DIR),AA,WORK,MAXA,NN,LENAA)
               WORK(DIR+NN)=DDOT(NN,YP,1,WORK,1)
```

```
            CALL DAXPY(NP1,WORK(NP1),YP,1,WORK(DIR),1)
C
C           COMPUTE INNER PRODUCTS.
C
            RBNPRD=DDOT(NP1,WORK(RES),1,WORK(RES),1)
            PBNPRD=DDOT(NP1,WORK(DIR),1,WORK(DIR),1)
C
C           CHECK FOR CONVERGENCE.
C
            IF (SQRT(RBNPRD) .LE. RBTOL) THEN
               STILLB=.FALSE.
            ENDIF
         ENDIF
         IF (STILLB) THEN
C
C           UPDATE SOLUTION VECTOR.
C             Z = Z + AB*DIR, WHERE AB=RBNPRD/PBNPRD.
C
            AB=RBNPRD/PBNPRD
            CALL DAXPY(NP1,AB,WORK(DIR),1,WORK(ZB),1)
C
C           COMPUTE RELATIVE CHANGE IN SOLUTIONS.
C
            DZNRM=AB*SQRT(PBNPRD)
            ZLEN=DNRM2(NP1,WORK(ZB),1)
C
C           IF RELATIVE CHANGE IN SOLUTIONS IS SMALL ENOUGH, EXIT.
C
            IF ( (DZNRM/ZLEN) .LT. ZTOL) STILLB=.FALSE.
         ENDIF
      ELSE
         STILLB=.FALSE.
      ENDIF
C
C     IF NO EXIT CRITERIA FOR  MZ=B  HAVE BEEN MET, UPDATE RESIDUAL,
C       DIRECTION VECTORS, AND INNER PRODUCTS.
C
      IF (STILLB) THEN
C
C        UPDATE RESIDUAL VECTOR; COMPUTE <RES,RES>.
C          RES = RES - AB*(Q**(-1))*M*DIR.
C
         CALL MULTDS(WORK,AA,WORK(DIR),MAXA,NN,LENAA)
         WORK(NP1)=DDOT(NN,YP,1,WORK(DIR),1)
         CALL DAXPY(NP1,WORK(DIR+NN),YP,1,WORK,1)
         CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
         CALL DAXPY(NP1,-AB,WORK,1,WORK(RES),1)
         RNPRD=DDOT(NP1,WORK(RES),1,WORK(RES),1)
C
C        UPDATE DIRECTION VECTOR; COMPUTE <DIR,DIR>.
```

```
C              DIR = (A**T)*(Q**(-T))*RES  + BB*DIR,
C              WHERE BB=RNPRD/RBNPRD.
C            (NOTE:  START IS USED AS A WORK ARRAY HERE).
C
               BB=RNPRD/RBNPRD
               RBNPRD=RNPRD
               CALL DCOPY(NP1,WORK(RES),1,WORK,1)
               CALL SOLVDS(NP1,WORK(Q),LENQ,MAXA,WORK)
               CALL MULTDS(START,AA,WORK,MAXA,NN,LENAA)
               START(NP1)=DDOT(NN,YP,1,WORK,1)
               CALL DAXPY(NP1,WORK(NP1),YP,1,START,1)
               CALL DAXPY(NP1,BB,WORK(DIR),1,START,1)
               CALL DCOPY(NP1,START,1,WORK(DIR),1)
               PBNPRD=DDOT(NP1,WORK(DIR),1,WORK(DIR),1)
            ENDIF
C
            J=J+1
         GO TO 300
400      CONTINUE
C        END DO
C
C SET ERROR FLAG IF THE CONJUGATE GRADIENT ITERATION DID NOT CONVERGE.
C
         IF (J .GT. IMAX) THEN
            IFLAG=4
            RETURN
         ENDIF
C
C ***** END OF  M Z = B  SYSTEM *****
C
C COMPUTE FINAL SOLUTION VECTOR  X,  AND RETURN IT IN START.
C
         TEMP=-WORK(ZB+NN)/(1.0D0+WORK(ZU+NN))
         CALL DCOPY(NP1,WORK(ZB),1,START,1)
         CALL DAXPY(NP1,TEMP,WORK(ZU),1,START,1)
C
         RETURN
         END
```

The vita has been removed from the scanned document