FINAL REPORT

NASA CONTRACT NAS8-21484

Solution of Systems of Nonlinear Equations

Submitted by Victor J. Law, Principal Investigator

for Tulane University, New Orleans, Louisiana

ABSTRACT

A generalized method called Diagonal Discrimination for nonlinear algebraic and transcendental equations is described. A computer program which implements the Diagonal Discrimination technique has been written in FORTRAN IV for the Univac 1108 computer. A users manual is included which gives detailed instructions of how to implement the computer program.

I. Diagonal Discrimination for Nonlinear Equation Solving

The method of Diagonal Discrimination (DD) was first described by Fariss and Law (1). A brief description of the algorithm will now be given.

DD belongs to a class of methods such that the computations always begin from a point $\underline{x}^o$ in n-dimensional space and a move or increment, $\underline{\Delta x}$, is computed such that $\underline{x}^1 = x^o + \underline{\alpha\Delta x}$ forms a search vector along which a "better" point is sought. This logic is repeated until no further improvement is possible. The choice of the scalar α is made by a one-dimensional search procedure.

The success of the method depends on a property of the $\underline{\Delta x}$ vector which shall be called <u>truncation convergence</u>. An algorithm for minimization has this property if, for sufficiently small α the objective function $q(\underline{x}^o + \underline{\alpha\Delta x})$ takes on a smaller value than $q(\underline{x}^o)$. What this means is that $\underline{\Delta x}$ must point in a direction such that q decreases at least locally. Hence, a better point can always be found by truncating α to some small positive value.

DD uses a unique combination of the method of weighted steepest descent and the Gauss-Newton method to minimize a sum of squares function. A brief review of these methods will now be given.

I. 1. Formulation as a Minimization Problem.

The specific problem to which attention is now given is that of finding the value of an n-vector $\underline{x}$ such that the equations

$$f_j(x) = 0; \qquad j = 1, 2, \ldots, n \tag{1-1}$$

are satisfied. The f functions, in general, are nonlinear. This problem may easily be formulated as a minimization problem by forming the sum of squares of residual as an objective function.

$$q = 1/2 \sum_{j=1}^{n} f_j^{\,2} \tag{1-2}$$

Clearly, if q is minimized to its absolute minimum of zero, then a solution to the original problem has been obtained.

I. 2. Ordinary Steepest Descent. Perhaps the oldest and still very popular method for unconstrained minimization is the method of steepest descent (SD). Strictly speaking, this method is a continuous one rather than a discrete one in that the path of steepest descent is a continuous curve. For practical use, however, the direction of steepest descent is found at the base point, $\underline{x}^o$, and this direction is used to form the search vector.

The direction of steepest descent is given by

$$\underline{\Delta x} = \frac{-\gamma \underline{g}(\underline{x}^o)}{|\underline{g}(\underline{x}^o)|} \tag{1-3}$$

where $\underline{g} = \frac{\partial q}{\partial \underline{x}}$, i.e., the gradient of q

γ = steepest descent distance factor

The search vector then becomes

$$\underline{x} = \underline{x}^{o} + \alpha \, \underline{\Delta x} \tag{1-4}$$

The value of α is usually selected by performing a one-dimensional search for a minimum in q with respect to α. Perhaps the most popular one-dimensional search is the Golden Section Search (2). A more sophisticated method is described by Fariss and Law (1) and is the one implemented in the computer programs listed in this report. It is also possible to simply find a value of α for which q is smaller than at the base point rather than finding the minimum. There is no generally "best" procedure to use. Judicious selection of γ can greatly enhance the performance of (SD). It is efficient to select this value of γ based on the optimal α from the previous iteration. One means of accomplishing this is to use the following relationship:

$$\frac{\gamma^{(i+1)}}{\gamma^{(i)}} = \begin{cases} a & ; \alpha_o^{(i)} = 0 \\ b & ; \alpha_o^{(i)} > b \\ \alpha_o^{(i)} & ; 0 < \alpha_o^{(i)} \leq b \end{cases} \tag{1-5}$$

where $\alpha_o^{(i)}$ = optimal value of α from the ith iteration

$\gamma^{(i)}$ = γ from iteration i

$\gamma^{(i+1)}$ = γ for iteration i+1

a = lower limit on correction factor

b = upper limit on correction factor

This formula is purely arbitrary and merely attempts to update γ based on past experience while not allowing large changes in it from one iteration to the next. The previous γ is multiplied by a factor between a and b, with a corresponding to $\alpha_o^{(i)} \doteq 0$ and b to $\alpha_o^{(i)} > b$. If $\alpha_o^{(i)} = 1$, γ is not changed. Recommended values for a and b are 1/4 and 4, respectively.

The $\underline{\Delta x}$ vector of Equation (1-3) is normal to the objective function contour at the base point and is guaranteed to have the truncation convergence property. This is obvious in that

$$\left.\frac{dq}{d\alpha}\right|_{\alpha\to 0} = \underline{g}^T \left.\frac{d\underline{x}}{d\alpha}\right|_{\alpha\to 0} = -\underline{g}^T\underline{g} = -\left|\underline{g}\right|^2 \qquad (1\text{-}6)$$

The most serious drawback of the method of steepest descent is the zig-zag tendency especially when near the solution. This property is best explained in the two-dimensional case. Referring to Figure 1-1, it is easily seen that successive directions of steepest descent will be orthogonal (perpendictular in 2-dimensions).

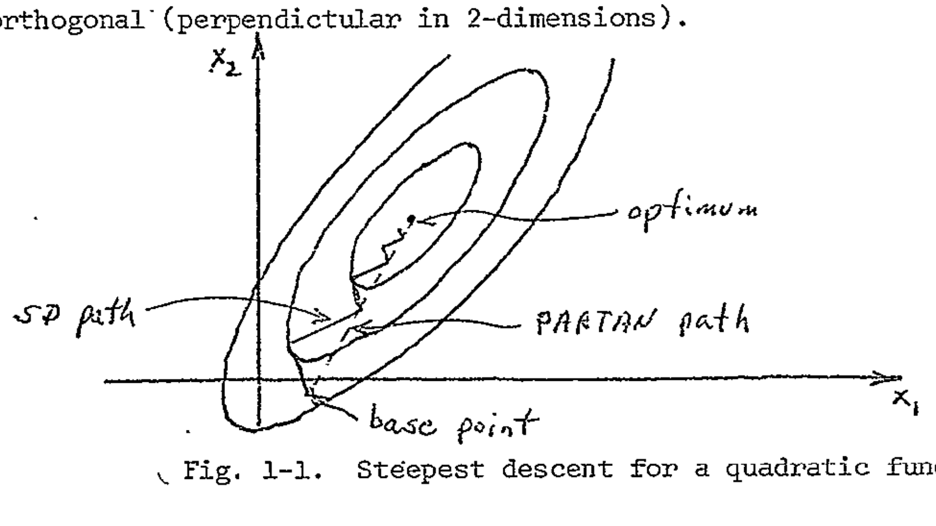

Fig. 1-1. Steepest descent for a quadratic function.

In order to overcome this difficulty, several techniques have been suggested [see, for example, Booth (3)]. One particular modificaton is the Method of Parallel Tangents developed by Shah, Buehler and Kempthorne (4). Weighted steepest descent is another modification and will be described in the sequel.

I. 3. <u>The Gauss-Newton Method (GN)</u>. This method uses Newton's method as a basis. That is, the equation for determing $\underline{\Delta x}$ is given by

$$\frac{\partial^2 q}{\partial \underline{x}^2} \Delta x = \frac{\partial q}{\partial \underline{x}} \tag{1-7}$$

A useful and effective approximation is made for the Hessian, $\frac{\partial^2 q}{\partial \underline{x}^2}$, however. In terms of the sum of squares q function, Equation (1-2),

$$\frac{\partial q}{\partial x_i} = \sum_{j=1}^{n} f_j \frac{\partial f_j}{\partial x_i} \tag{1-8}$$

and

$$\frac{\partial^2 q}{\partial x_i \partial x_k} = \sum_{j=1}^{n} \frac{\partial f_j}{\partial x_i} \frac{\partial f_j}{\partial x_k} + f_j \frac{\partial^2 f_j}{\partial x_i \partial x_k} \tag{1-9}$$

Now, for equation solving the $f_j \to 0$ as the solution is approached. Therefore, the so-called Gauss-Newton approximation is to omit the second term in the expression for $\frac{\partial^2 q}{\partial x_i \partial x_k}$. That is,

$$\frac{\partial^2}{\partial x_i \partial x_k} \approx \sum_{j=1}^{n} \frac{\partial f_j}{\partial x_i} \frac{\partial f_j}{\partial x_k} = G_{ik}$$

Thus, the relation for step size determination becomes

$$G\, \underline{\Delta x} = -\underline{g} \qquad (1\text{-}10)$$

In the absence of singularity of the G matrix, the (GN) procedure is very efficient at converging to the solution from a near point (i.e., where the sum of squares function becomes nearly quadratic and the f_j are small). This behavior could be anticipated from the approximate quadratic representation of q by using the G matrix as an approximation to $\partial^2 q/\partial \underline{x}^2$. Thus, quadratic convergence is obtained in the neighborhood of the solution.

As a protection against very long search trials, it is customary to limit the length of the search vector to some arbitrary value. While this will usually force convergence, it has one distinct disadvantage. In order to clearly understand why this is so, consider a problem where only one variable causes the singularity. In such a case the moves predicted for all other variables would be quite good. Thus, the truncation of the entire search vector is clearly inefficient in that only the move in the maverick variable need be truncated. This justifies the need for the following two operations:

(1) Sorting those variables which cause the singularity in G.

(2) Selectively truncating only the moves for these variables.

More will be said about this later.

The (GN) method is equivalent to the well known Newton-Raphson (5) (NR) method for equation solving. The (NR) method uses as its basis the linearization of each equation in the neighborhood of the base point. Thus,

$$f_j(\underline{x}) \simeq f_j(\underline{x}^o) + \sum_{i=1}^{n} \frac{\partial f_j(\underline{x}^o)}{\partial x_i} \Delta x_i \quad ; \; j=1, 2, \ldots, n \qquad (1\text{-}11)$$

The move is then determined so that each $f_j(\underline{x})$ would become zero if all functions are linear. In vector-matrix form, these relations become

$$J \, \underline{\Delta x} = -\underline{f} \qquad (1\text{-}12)$$

where

$$J = \text{the Jacobian matrix with } J_{ij} = \frac{\partial f_i}{\partial x_j}$$

In equation (1-12) is premultiplied by J^T (superscript T indicates the matrix transpose), there results

$$(J^T J) \, \underline{\Delta x} = -J^T \underline{f} \qquad (1\text{-}13)$$

Which is identical to Equation (1-11). That is,

$$G = J^T J \tag{1-14}$$

$$\frac{\partial q}{\partial \underline{x}} = J^T \underline{f} \tag{1-15}$$

Equation (1-14) constitutes a proof that G is always at least positive semi-definite in that this property always holds for a product of a matrix and its transpose.

I. 4. Weighted Steepest Descent (WSD). Weighted steepest descent is a modification of the method of steepest descent. It arises from modifying the elements of the search vector by non-equal positive multiplicative factors chosen so as to produce a more effective vector. Specifically, these factors may be selected so that, under favorable circumstances (i.e., when G is strongly positive definite), the search vector will coincide with the one produced by Newton's method, provided the method is applied in a coordinate system where, in relation to the quantity being minimized, there is no local interaction of variables. The necessity of using such a coordinate system will be made clear in the sequel.

The coordinate system required may be created by transforming the G matrix into diagonal form. Let T by a non-singular transformation matrix such that

$$T^T G T = D \tag{1-16}$$

where D is diagonal. Methods for computing such transformation matrices are discussed by Wilde and Beightler (6). If this transformation is applied to Equation (1-10), there results

$$T^T G T T^{-1} \underline{\Delta x} = -T^T g \tag{1-17}$$

By defining the new variables (coordinates)

$$\underline{y} = T^{-1} \underline{\Delta x} \tag{1-18}$$

Equation (1-17) for the search vector $\underline{y}$ in terms of transformed coordinates may be written as

$$D\underline{y} = -T^T \underline{g} \tag{1-19}$$

On the same basis, the steepest descent equation becomes

$$\underline{y} = -T^T \underline{g} \tag{1-20}$$

In order for coincidence to exist between Newtonian and steepest descent vectors, Equation (1-20) must be modified to

$$\underline{y} = -kD^{-1} T^T \underline{g} \tag{1-21}$$

in which kD^{-1} supplies the necessary weighting. The scalar factor k is indeterminate (but positive), since steepest descent is intended to define only the <u>direction</u> of the search vector.

The general form of the weighted steepest descent equations, on a transformed coordinate basis, may then be written as

$$\underline{y} = -\underline{W}T^T\underline{g} \tag{1-22}$$

in which $\underline{W}$ is a diagonal matrix of positive elements. Consequently, coincidence between Newtonian and steepest descent vectors can be achieved only if all diagonal elements d_{ii}, are non-zero positive, that is, if the G is positive definite. Furthermore, experience has shown that, <u>if reasonable external scaling has been applied</u>, weighting factors which are excessively large to yield an effective search vector will result from Equation (1-21) when one or more are orders of magnitude smaller than others. It may be concluded from this that the favorable circumstances under which it is feasible and reasonable to weight steepest descent so as to force coincidence with the Newtonian search vector are confined to cases where all d_{ii} are positive and of the same order of magnitude.

An important feature of the weighted steepest descent method is that it can be adapted to deviate from Newton's method when circumstances do not warrant their coincidence. This can be achieved by adapting the calculation of the weight factors. The suggested method for doing this will now be described.

First, assume that the diagonal elements, d_{ii} and the corresponding columns of the transformation matrix, T, are rearranged to achieve descending algebraic order. That is, $d_{11}>d_{22}>\ldots>d_{nn}$. If k in equation (3-: is chosen as d_{11}, then

$$w_{11} = 1$$

and ideal weighting (i.e., that which causes coincidence with Newton's method) is given by

$$w_{ii} = \frac{d_{11}}{d_{ii}} \quad ; \quad i = 2, 3, \ldots, n \tag{1-23}$$

An adaptation of the ideal case which has been used successfully is, for i=1, 2, ..., n

$$w_{ii} = \begin{cases} \dfrac{d_{11}}{d_{ii}} & \text{if } \dfrac{d_{11}}{d_{ii}} \leq \beta \text{ and } d_{ii} > 0 \\ \beta & \text{if } \dfrac{d_{11}}{d_{ii}} > \beta \text{ or } d_{ii} \leq 0 \end{cases} \tag{1-24}$$

In Equation (1-24), β is the maximum weight factor allowed. Thus, if $d_{ii} \ll d_{11}$ or if $d_{ii} \leq 0$, w_{ii} deviates from the ideal one and is assigned the value β. It should be pointed out that Equation (1-24) is not the only choice which can be made for non-ideal weighting but is one which has been used successfully.

An outline of the weighted steepest descent method follows:

1) Select a distance factor γ and a base point $\underline{x}^o$.

2) Compute $\underline{g}$ and G at the base point.

3) Find the transformation matrix T and the corresponding diagonal matrix D. Reorder D and columns of T so that the resulting d_{ii} are in descending algebraic order.

4) Compute a set of weighting factors from Equation (1-24).

5) Compute the $\underline{y}$ vector as follows:

   a) First compute an interim $\underline{y}$ indicated here by $\underline{y}^i$ as follows:

$$y_i^i = -w_{ii}\, p_i \quad ; \quad i = 1, 2, \ldots n$$

where $\underline{p} = T^T \underline{g}$

   b) Set $$y^i_{max} = \max \left| y^i_i \right| \qquad (1\text{-}25)$$

$$\mu = \gamma / y^i_{max} \qquad (1\text{-}26)$$

   c) Then, $$y_i = -\mu\, w_{ii}\, p_i \qquad (1\text{-}27)$$

   Note that the y_i of largest magnitude has a magnitude equal to γ.

6) Perform a one-dimensional search along the vector

$$\underline{x} = \underline{x}^o + \alpha\, \underline{\Delta x} \qquad (1\text{-}28)$$

where $\underline{\Delta x} = T\underline{y}$ (1-29)

7) Update γ based on the experience of the one-dimensional search [see Equations (1-5)].

8) Go to step (2) and repeat until convergence is achieved.

Equations like (1-22) and (1-23) could also be applied to define a weighted steepest descent algorithm for the original, untransposed coordinate system. The best analogy to Equation (1-24) would involve using the diagonal elements of the G in place of the d_{ii}. This approach requires substantially less computation to determine a search vector, since only the diagonal of the G is required, and the diagonalization calculation is not used. However, if interaction is at all significant, there will be no coincidence at all with the Newtonian vector, and this approach will be subject to most of the inefficient convergence problems usually experienced with ordinary steepest descent. One significant reason for this is that it is impossible, in general, to weight the Δx_i to make the SD search vector coincide with the NM search vector. To see this more clearly, consider a quadratic function with contours as shown in Figure 1-2.

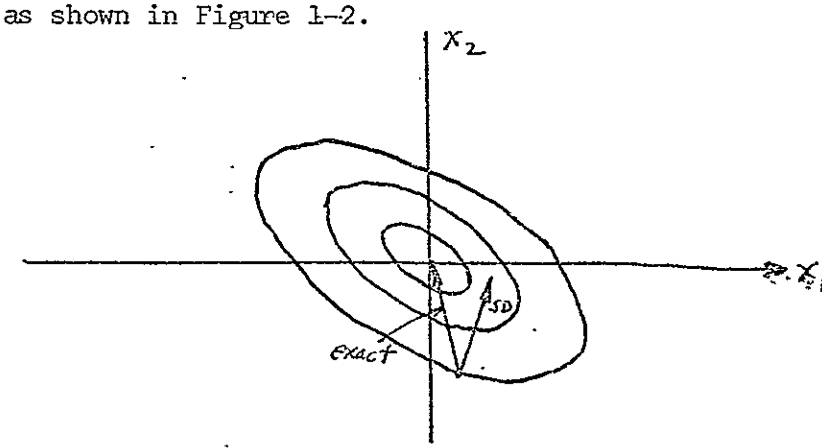

Fig. 1-2. SD and Exact Search Vectors.

The SD direction produces positive values for both Δx_1 and Δx_2 whereas the exact search vector requires Δx_1 to be negative and Δx_2 to be positive. Thus, no positive weighting factors exist. It is not proper to use negative weight factors since the property of truncation convergence would be destroyed. To see this, consider a diagonal matrix of weights, V. Then a weighted SD direction would be

$$\underline{\Delta x} = -V\underline{g} \tag{1-30}$$

Then Equation (1-6) becomes

$$\left.\frac{dq}{d\alpha}\right|_{\alpha\to 0} = \underline{g}^T \left.\frac{d\underline{x}}{d\alpha}\right|_{\alpha\to 0} -\underline{g}^T V\underline{g} \tag{1-31}$$

Thus $\left.\frac{dq}{d\alpha}\right|_{\alpha\to 0}$ will be always negative only if V is positive definite. Hence, each scale factor must be positive.

Therefore, scaling the $\underline{y}$ vector is another means of accelerating the convergence of steepest descent. As long as the d_{ii} are positive, the ideal weighting factors given by Equation (1-23) can be quite effective toward this end. However, if there is a large difference in the magnitudes of the d_{ii} or if any d_{ii} are negative or zero, the ideal weighting is either dangerous or not possible. In these situations it has been found that the concept of weighting the $\underline{y}$ vector is still useful but must be suitably modified. One practical means of accomplishing this is that given by Equation (1-24).

I. 5. Diagonal Discrimination (DD). It was discovered that (WSD) had one significant shortcoming. That is, no consideration is given to the relation between the d_{ii} and the corresponding elements of the transformed gradient, $\underline{p}$. Thus, even if d_{ii} is such that ideal weighting is allowed, the y_i produced by (WSD) might be excessively large because of a large p_i. The quadratic approximation of q at $\underline{x}^o$ upon which Newton's method is based is likely to be poor at points far removed from the base point. Therefore, only in the very fortunate but, unlikely (at least in practical applications) case that q is almost exactly quadratically dependent upon a particular y_i would a long move in that variable be warranted. In order to account for this situation the weighted steepest descent method was modified into another method called diagonal discrimination (DD).

(DD) is a method which involves choosing a superior search vector based upon the following two principles:

1) Transformation of the coordinate axes in order to remove local interaction between variables.

2) Computing the elements of the search vector discriminantly depending upon the relationships between the diagonal elements of the transformed Hessian and the length of a move as predicted by Newton's method in the transformed coordinate system.

The development of the method is conveniently begun by considering Equation (1-19). Assume again that the columns of T have been rearranged so that the d_{ii} appear in descending algebraic order.

The logic of DD presupposes that reasonable external scaling of the variables has been accomplished so that the following assumptions are likely to be valid:

1) The acceptability of a move (from a base point) calculated by Newton's method can be judged from its length--long moves are suspect.

2) In regard to the diagonalized G, positive d_{ii} which are several orders of magnitude smaller than the largest one are indicative of a linear dependence of q on the associated y coordinate. Negative d_{ii} are associated with those y coordinates for which the objective function tends to exhibit a local maximum rather than a minimum.

Diagonal Discrimination then attempts to combine the best features of the (G-N) method and (WSD) by computing the $\underline{y}$ vector components discriminantly. That is, if the (G-N) move for a particular y_i is not too long and if the associated d_{ii} is positive and not greatly different from d_{11}, then the (G-N) calculation is used. Otherwise, y_i is found by (WSD) logic. A step-wise presentation of the logic for DD follows:

1) Select a base point; $\underline{x}^o$, a steepest descent distance factor, γ, and a steepest descent threshold factor, δ.

2) Compute $\underline{g}$ and G at the base point.

3) Find the T and D. Order D and T so that the d_{ii} are in descending algebraic order.

4) Starting with y_1, the elements of $\underline{y}$ are computed by (G-N) logic, that is

$$y_i = -p_i / d_{ii} \qquad (1\text{-}32)$$

until either

a) $i = n$

b) $d_{ii} \leq \varepsilon\, d_{11}$

c) $y_i > \delta$

5) When the (G-N) sequence is terminated for reason (b) or (c) above, at the kth y variable, then a switch is made to weighted steepest descent logic treating y_K as the first (WSD) variable. That is, y_K is assigned a weighting factor of 1 and y_K, y_{K+1}, ..., y_n are computed by the (WSD) method as described above.

6) If any $d_{ii} \leq \varepsilon\, d_{11}$, set $y_i = 0$ (these are called null-effect parameters and are discussed in the next subsection).

7) The resulting $\underline{y}$ vector is converted back into a $\underline{\Delta x}$ vector by

$$\underline{\Delta x} = T\underline{y} \qquad (1\text{-}33)$$

and a one-dimensional search is performed along the $\underline{x}$ vector as given by Equation (1-28).

8) The distance factor, γ, is updated using Equation (1-5).

9) Go to step (2) and repeat until convergence is achieved.

It can be shown in that the DD search vector has the property of truncation convergence. A discussion of parameter selection (β, γ, δ, ε) is given below.

Selection of β:

The choice of β, the maximum allowable weight factor in Equation (1-24) is not critical. That is, the sensitivity of WSD to β is relatively slight. A value of about 10^4 has been used satisfactorily in almost all applications of DD.

Selection of γ:

If the variables have been scaled (see next section) in any reasonable way their expected value should be about unity, at least within a few orders of magnitude. A change of 20-50% would, therefore, be considered relatively large. Therefore, if γ is chosen to be 0.2, the length of the SD search vector will limit each individual variable increment to be less than 20% of unity. The actual initial choice of γ is not too critical since it is updated at each iteration.

Selection of δ:

The steepest descent threshold factor, δ, should be chosen with the philosophy that if some y_i as calculated by Newton's method is greater than δ, then a switch should be made to WSD logic. Again, if reasonable external scaling has been applied, a value of about 0.2 for δ has been found to be satisfactory.

Selection of ε:

The parameter ε is used to distinguish between small d_{ii} and those which should be treated as zero. The choice of ε depends upon the number of significant digits carried in the arithmetic. For digital computer which carry d digits, it is recommended that ε be chosen as follows:

$$= 10^{-d} \tag{1-34}$$

I. 6. <u>Singularity of G and Null Effect Variables</u>. The singularity of the G matrix is usually caused by a phenomenon which shall be called <u>null effect</u>. Those variables which cause this condition will be called <u>null effect variables</u>. Null effect occurs when perturbation of a parameter or of some linear combination of parameters has no significant effect on <u>any</u> of the residuals in the sum of squares function. In a well posed problem, null effect should not, of course, be present at the solution point. For systems of nonlinear equations, however, null effect is common at points removed from the solution. This is caused by local redundancies or inconsistencies in the linearized versions of the equations. In either case, two or more linear equations become parallel to each other and hence no solution exists. As an example, consider the problem illustrated in Fig. 1-3.

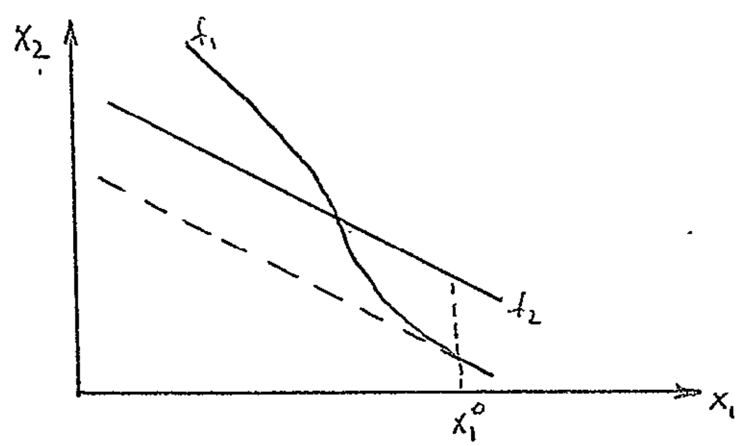

Figure 1-3. Illustration of Null Effect

I. 7. External Scaling. External scaling of variables is desirable (and highly recommended), primarily because of the problem of heterogeneous physical dimensions. Variables which have different physical dimensions will naturally have different ranges of variation and different impact on the objec function. In any numerical procedure it is desirable that the variables have similar ranges of variation because of round-off considerations. Vastly different effects on the objective function lead to very elongated contours of the objective function. The latter problem results in large differences in the diagonal elements of the transformed Gauss-Newton matrix, G.

The most natural way to scale variables is by way of dimensional analysis. That is, some variables may be made dimensionless by choosing a natural scale factor. An example of this for temperature as the variables would be to set

$$t^* = \frac{t - t_L}{t_U - t_L} \tag{1-35}$$

where t^* = dimensionless temperature

t_U, t_L = upper and lower bounds on the range of temperature to be considered

In addition to having no physical units, the dimensionless temperature would also, conveniently, vary between zero and one.

However, it is not always possible or practical to scale the problem as suggested on the basis of dimensional analysis. Another way of arriving at scaled variables which has proven to be effective and satisfactory is to choose an increment for numerical differentiation which will cause a change in the objective function in the last three or four digits (as available on the computer to be used). Then choose an external scale factor which will cause the scaled increment for differentiation to be the same for all variables. That is,

$$e_i = \frac{\delta y_i}{c_d} \tag{1-36}$$

where c_d = increment for differentiation of scaled variables

e_i = external scale factor for x_i

δy_i = chosen increment for numerical differentiation of unscaled variables

This technique can usually be carried out easily if the problem solver has some basic knowledge of the problem and if sound engineering judgement is applied. Variable scaling is automatically provided in the computer programs described herein.

II. User's Manual for Implementing Computer Programs.

II. 1. Basic Characteristics. The basic characteristics of the system are as follows:

1) The user sets up arrays containing starting values or simply a range for the parameters being sought, and then starts DD action by the appropriate subroutine calls.
2) Thereafter, until the iterative procedure is terminated by the subroutines, the values contained in the parameter array are under control of the subroutines. The user must provide programming to calculate residuals for the equations for any set of values in the parameter array, on demand by the DD algorithm.
3) When control is released, the parameter array will contain "best" values, i.e., those which cause the sum of squares of residuals to be a minimum.
4) All required partial derivatives are obtained numerically.
5) There are no internal limits on the size of problems due to dimensions, etc. It is possible to solve problems as large as about 200 parameters and 200 equations.
6) No special NASA system features are required.

II. 2. Skeletal Illustration. A skeletal illustration of a main program which meets the basic requirements for using the subroutines which implement diagonal discrimination is shown in the following:

```
    DIMENSION X(··), SCALE (··), S(··), SX(··), G(··), XX(··), MODE (··),
              RHS(··), DXBAR(··), DXX(··) DQ(··), DQX(··), YBASE(··),
              Y(··), D2Q(··), XMIN(··), XMAX(··)
    COMMON / DDCOM / COM(9), KCOM(13)
    EQUIVALENCE (KCOM(1), NCASE), (KCOM(2), NVAR), (KCOM(3), IPRNT)
    NCASE  =  (specify a number of equations)
    NVAR   =  (specify number of unknown parameters)
    IPRNT  =  (specify output option)

                                                XMIN( )=
 X( ) = (make initial guesses)   or                        (specify range on each variable)
                                                XMAX( )=

 KODE = 0                                       KODE    = 1

10  CALL CONST
    CALL DDSET (KODE, NVAR, X, XMIN, XMAX, SCALE)
20  CALL DDRG (S, SX, G, XX, MODE, RHS, DXBAR, DXX, DQ, DQX, Y, YBASE,
    DY, X, SCALE, D2Q, SCALR)
    IF (KCOM(5) .LE. 0) GO TO 100
    CALL NEWSCL (X, SCALE)
    Y( ) = (calculate residuals for all equations)
    GO TO 20
100 IF (KCOM(8) .EQ. 0) GO TO 200
    GO TO 10
200 ... (problem completed)
```

The dimension statements establish memory requirements for all arrays. Minimum space requirements for these are outlined below:

Array Name	Minimum Dimension
X,SCALE,XX,MØDE,RHS,DXBAR,DXX,DQ,DQX,D2Q,SCALR, XMIN,XMAX	NVAR
Y,YBASE	NCASE
S,SX,G	$(NAR)^2$
DY	(NVAR)X(NCASE)

where NVAR = Number of parameters to be determined

NCASE = Number of equations

The CØMMØN statement creates linkage of the listed variables between the main program and the D.D. subroutines.

The regression calculations are initiated by calling subroutine DDRG, statement 20 above. However, prior to this, <u>four</u> things must have been done:

1) Specify NVAR and NCASE.

2) Set the print control, IPRNT.

3) Put starting values (initial guesses) for the parameter in the X array or specify lower and upper bounds on each parameter (XMIN and XMAX).

4) Set regression controls by calling subroutines CONST and DDSET.

The action required after the return from subroutine DDRG depends on the value of KCOM(5) as set by that subroutine:

When KCOM(5)=0, the subroutines have probably completed their operation. If a normal finish was obtained, KCOM(8) will be returned as a zero. At this point the X array will contain the parameter values corresonding to the problem solution, and the final sum of squares, SIGMA, will be $\leq 10^{-9}$. If KCOM(8)≠0, a false solution was found and a restart (from a different initial guess) should be initiated by returning to the call of CONST. When KCOM(5)>0, the main program must calculate the values of the "residuals" and place these values in the Y array. If the equations are written so that all terms appear on only one side of the equal sign (e.g., $x_1^2 + 2x_2^2 = 0$), then the Y array contains the current value of that side of the equation [e.g., Y(1) = X(1)**2+2.0*X(2)**2].

If it is desired to start the solution of another set of equations, NVAR, NCASE, IPRNT and MASCT should be reset and subroutines CONST and DDSET should be called again before the next initial call of DDRG.

II. 3. Internal Print Control and Output Description. The D.D. subroutines brought into action by the call of DDRG furnish considerable information during the progress of the calculation if the contained print statements are not suppressed. Suppression can be accomplished by setting IPRNT=0. Information for only the last iteration is obtained if IPRRNT=1. However, when new problems are run, it is strongly recommended that the complete internal print facility be used by setting IPRNT=2.

Output items appear in the following sequence:

1) Heading line
2) Table of starting values and scale factors for parameters
3) Table of regression controls
4) Iteration line
5) Parameter, Internal variable table
6) Search table
7) Termination explanation

Items 4 through 6 are repeated for each iteration when IPRNT=2.

Items described in the Iteration Line are as follows:

ITER NO	Iteration number, beginning with one
INACTIVE	Number of "null effect" parameters for this iteration (see Section I.6.)
DIST	This is the distance of the desired first try along the search vector, in the scaled parameter space.
FSD	This is the current value of the steepest descent distance factor.
SIGMA	Current value of the sum of squares

Under PARAMETERS are three columns giving respectively the parameter number, the current value and the parameter increment corresponding to the desired first try along the search vector.

Under INTERNAL VARIABLES are four columns, all referring to the transformed coordinates. They are respectively the diagonalized Gauss-Newton matrix, elements, d_{ii}, first partial derivatives of the sum of squares, coordinate increment for first try along the search vector and the MODE of incrementing the transformed coordinates. These MODES are as follows:

0 - Gauss-Newton

1 - Steepest Descent

-2 - Null Effect

The one-dimensional search table under SRCHS contains two columns which contain

FX - The factor multiplied by the search vector increment to obtain a trial point. The last one printed is the optimum one.

ACTION - Hollerith information which describes the progress of the search

SIGMA - The value of the sum of squares for the corresponding value of FX

The termination explanation is self-explanatory.

II. 4. Storage Requirements. The large bulk of the required storage for the D.D. program is taken up by the arrays contained in the DIMENSION statement. If there are N equations and N unknowns, this requirement calls for $15N + 3N^2$ words of storage. For a 200 variable problem, this would require about 43,000 words. Thus, this size problem should be well within the storage limits of the Univac 1108 computer.

II. 5. Special Features. There are two special features of the D.D. program which deserve mention. These are an automatic restart procedure and the ability to treat sets of underdetermined or overdetermined nonlinear equations.

The automatic restart procedure operates as follows:

> When KCOM(5)=0, the problem has been solved satisfactorily only if KCOM(8)=0. If this is not the case, then a return to the call of CONST and DDSET causes a new (random) starting point to be generated within the feasible space (i.e., XMAX(I) $\geq$ X(I) $\geq$ XMIN(I), I=1, 2, ..., NVAR). Note, therefore, that XMIN and XMAX must be specified even if the KØDE=0 option were used on the initial start of the problem. This restart procedure will be generated a maximum of five times. The restart procedure can be suppressed by deleting the test of KCOM(8). That is, at statement 100, the problem is complete.

In order to solve sets of underdetermined equations (i.e., more variables than equations) or overdetermined equations (i.e., fewer variables than equations), it is necessary only to provide the proper values for NVAR and NCASE. In the case of overdetermined sets, a compromise solution (i.e., a minimum sum of squares solution) is all that can be expected. Therefore, the automatic restart procedure should be suppressed as described above.

SAMPLE PROBLEM

A sample problem is included to illustrate further the programming requirements and usage. In particular, the system of linear algebraic equations

$$2x_1 - x_2 = 1$$
$$x_1 + x_2 = 1$$

are solved from the initial guess $x_1 = 1/2$, $x_2 = 1/2$. The answers are $x_1 = 2/3$, $x_2 = 1/3$. The solution was not reached in one iteration because the required step size exceeded the limits normally provided and WSD was used.

A listing of the main program and printed output are given below.

```
      DIMENSION X(2),SCALE(2),S(4),SX(4),G(4),XX(2),MODE(2),
     1 RHS(2),DXBAR(2),DXX(2),DQ(2),DQX(2),YBASE(2),Y(2),D2Q(2),
     2 SCALR(2),DY(4),XMIN(2),XMAX(2)
      COMMON /TDCOM/
     1            SSCAL,FIT,FDSCD,FDERV,DSTMN,FSD,SIGMA,DXMAX,DSCAL,NCASE,
     2 NVAR,IPRNT,MAXCT,ICODE,INEW,ICASE,KLUE,ITER,NKICK,NRET,
     3 IVAR,NEWSTR
      NCASE = 2
      NVAR = 2
      IPRNT = 2
      DO 2 I=1,NVAR
      XMIN(I) = 0.0
    2 XMAX(I) = 1.0
      X(1) = 0.5
      X(2) = 0.5
      KODE = 0
   10 CALL CONST
      CALL BOSET(KODE,NVAR,X,XMIN,XMAX,SCALE,NEWSTR)
    1 CALL DDRG (S,SX,G,XX,MODE,RHS,DXBAR,DXX,DQ,DQX,Y,YBASE,DY,X,
     1                SCALE,D2Q,SCALR)
      IF(ICODE .LE. 0) GO TO 100
      CALL NEWSCL(X,SCALE)
C
    8 Y(1) = 2.*X(1) - X(2) - 1.0
      Y(2) = X(1) + X(2) - 1.0
      GO TO 1
C
  100 IF(KLUE .EQ. 0) GO TO 200
      GO TO 10
  200 CALL EXIT
      END
```

NONLINEAR EQUATIONS VIA D.D.

	STARTING VALUE	SCALE
1	0.500000E 00	0.5000E 00
2	0.500000E 00	0.5000E 00

PARAMETER VALUES

```
FIT    = 0.1000E-08
MAXCNT =    20
DXMAX  = 0.1000E 02
DSCALE = 0.2000E 01
SSCALE = 0.2000E 01
FSD    = 0.2000E 00
NVAR   =    2
NCASE  =    2
FDERIV = 0.1000E-02
DSTMIN = 0.5000E-01
FDSCRD = 0.0000E-38
```

ITER NO	-INACTIVE-	DIST	FSD	SIGMA
1	0	0.25476	0.20000	0.250000E 00

	PARAMETERS			INTERNAL VARIABLES			
1	0.500000E 00	0.842193E-01	*	0.1250E 01	-0.5000E 00	0.20000	1
2	0.500000E 00	-0.789034E-01	*	0.4500E 00	0.1500E 00	-0.15781	1

SRCHS

FX	ACTION	SIGMA
		2.50000E-01
1.0000E 00	TRY MORE	6.38643E-02
2.0000E 00	TAKE	1.41297E-04

ITER NO	-INACTIVE-	DIST	FSD	SIGMA
2	0	0.02589	0.27826	0.141297E-03

	PARAMETERS			INTERNAL VARIABLES			
1	0.668439E 00	-0.177206E-02	*	0.2234E 01	-0.1997E-07	0.00000	0
2	0.342193E 00	-0.885993E-02	*	0.2108E 00	0.5457E-02	-0.02589	0

SRCHS

FX	ACTION	SIGMA
		1.41297E-04
1.0000E 00	TRY MORE	3.13638E-14
2.0000E 00	TAKE PREV	1.41297E-04

ITER NO	-INACTIVE-	DIST	FSD	SIGMA
3	0	0.00000	0.27826	0.313638E-13

	PARAMETERS			INTERNAL VARIABLES			
1	0.666667E 00	0.769900E-07	*	0.2222E 01	-0.2633E-06	0.00000	0
2	0.333333E 00	-0.993354E-08	*	0.2000E 00	0.5960E-08	-0.00000	0

D.D. FINISH NORMAL

FOLDOUT FRAME #1

FOLDOUT FRAME #2

LISTING OF FORTRAN IV COMPUTER PROGRAMS

```
      1 SUBROUTINE NEWFSD (FX, DQDZB, FSD)
C
     11 IF (FX) 12,12,21
     12 FACT = 0.25
     13 GO TO 31
C
     21 Z = 0.88255*ATAN(0.56654*ALOG(FX))
     22 FACT = EXP(Z)
C
     31 FSD = FSD * FACT
     32 RETURN
C
        END
```

```
      SUBROUTINE CONST
C     THIS SUBROUTINE SETS THE NORMAL REGRESSION CONTROLS
C
C
      COMMON /TDCOM/
     1       SSCAL,FIT,FDSCD,FDERV,DSTMN,FSD,SIGMA,DXMAX,DSCAL,NCASE,
     2 NVAR,IPRNT,MAXCT,ICODE,INEW,ICASE,KLUE,ITER,NKICK,NRET,
     3 IVAR,NEWSTR
      SSCAL = 2.0
      FIT = 0.1
      FDSCD = 1.0
      FDERV = 0.001
      DSTMN = 0.05
      FSD = 0.2
      DXMAX = 10.0
      DSCAL = 2.0
      KLUE = -1
      NKICK = 1
      MAXCT = 20
      FIT = 1.E-09
      FDSCD = 0.0
      NEWSTR = 0
C
      RETURN
      END
```

```
      1 SUBROUTINE TDVEC  (N, NRES, SIGMA, DQ, ND2Q, D2Q, FSD, FSCAL,
     11  FDSCR , DX, MODE, INACT, NSD, NOEX)
C
C     THIS SUBROUTINE PERFORMS ROTATIONAL DISCRIMINATION LOGIC TO
C          DETERMINE A SEARCH INCREMENT VECTOR
C
C     INPUT=
C         N       - NO. OF VARIABLES (SPACE DIMENSION)
C         NRES    - NO. OF RESIDUALS
C         SIGMA   - SUM OF SQUARED RESIDUALS
C         DQ      - ARRAY OF PARTIAL DERIVATIVES OF SIGMA
C         ND2Q    - REPEAT CYCLE FOR D2Q
C         D2Q     - MATRIX OF SECOND PARTIAL DERIVATIVES OF SIGMA
C         FSD     - LIMITING DISTANCE FOR VARIABLE MOVE
C         FSCAL   - LOG(10) OF UPPER BOUND FOR INTERNAL SCALING FACTORS
C         FDSCRD- UPPER BOUND FOR ACCEPTABLE STANDARD DEVIATION IN
C                   VARIABLE ESTIMATION  --  SET ZERO IF STATISTICAL
C                   TEST IS TO BE IGNORED
C
C     OUTPUT=
C         DX      - SEARCH INCREMENT VECTOR
C         MODE    - ARRAY INDICATING CALCULATION MECHANISM FOR ELEMENTS
C                   IN DX=
C                      0  -  GAUSS-NEWTON
C                      1  -  STEEPEST-DESCENT
C                      2  -  SD TRUNCATED BY FSD LIMIT
C                     -1  -  DEACTIVATED DUE TO STANDARD DEVIATION FOR
C                              MOVE, AS CALC BY GAUSS-NEWTON, BEING
C                              ESTIMATE BEING ABOVE UPPER BOUNDS AND
C                              LESS THAN ST DEV OF ESTIMATE
C                     -2  -  DEACTIVATED DUE TO SECOND DERIV BEING
C                              .LE. 10-9 X LARGEST SECOND DERIV
C         INACT   - NO. OF DEACTIVATED VARIABLES
C         NSD     - NO. OF VARIABLES IN SD (MODES 1 OR 2)
C         NOEX    - NO. OF VARIABLES NOT DEACTIVATED BUT WITH EXCESSIVE
C                   ST DEV OF ESTIMATE
C
C          NOTE=
C            THIS PROGRAM USES ONLY THE DIAGONAL ELEMENTS OF D2Q. IF
C            THESE HAVE ALREADY BEEN CONDENSED, I.E., PUT IN A ONE-
C            DIMENSIONAL ARRAY OF DIAGONAL ELEMENTS ONLY, USE THIS ARRAY
C            AS D2Q AND SET ND2Q = 0.
C
      2 DIMENSION DQ(1), DX(1), MODE(1), D2Q(1)
C
     11 NSD = 0
     12 INACT = 0
        NOEX = 0
         ISD = 0.0
     13 IF (N) 999,999,14
     14 ND2QP  = ND2Q + 1
     15 DCHK = 1.0E-9 * ABS(D2Q(1))
     16 VARIA  = SIGMA/SQRT(FLOAT(NRES))
     17 IF (FDSCR ) 21,21,18
```

```
   18 VCHK = VARIA /FDSCR **2
C
   21 II = 1
   22 DO 92 I = 1,N
      KEX = 0
   23 IF (FDSCR ) 31,31,24
   24 IF (D2Q(II) - VCHK) 25,31,31
   25 IF (DQ(I)**2 - VARIA *D2Q(II)) 26,30,30
   26 MODE(I) = -1
   27 INACT = INACT + 1
   28 DX(I) = 0.
   29 GO TO 91
C
   30 KEX = 1
   31 IF (D2Q(II) - DCHK) 32,32,34
   32 MODE(I) = -2
   33 GO TO 27
   34 NOEX = NOEX + KEX
C
   41 IF (NSD) 42,42,61
   42 IF (ABS(DQ(I)) - FSD*D2Q(II)) 43,43,51
   43 DX(I) = -DQ(I)/D2Q(II)
   44 MODE(I) = 0
   45 GO TO 91
C
   51 ISD = I
   52 IISD = II
   53 CALL RESCAL (N-ISD+1, ND2Q, D2Q(IISD), FSCAL, DX(I))
   54 DX(I) = -SIGN(FSD,DQ(I))
   55 MODE(I) = 1
   56 NSD = NSD + 1
   57 GO TO 91
C
   61 IF (DQ(I)) 71,62,71
   62 DX(I) = 0.
   63 MODE(I) = 0
   64 GO TO 91
C
   71 IF (ABS(DQ(I)) - ABS(DQ(ISD))/DX(I)**2) 72,72,81
   72 DX(I) = -DQ(I) * FSD * DX(I)**2 / ABS(DQ(ISD))
   73 GO TO 55
C
   81 DX(I) = -SIGN(FSD, DQ(I))
   82 MODE(I) = 2
   83 GO TO 56
C
   91 II = II + ND2QP
   92 CONTINUE
C
      IF(ISD .EQ. 0) GO TO 999
      YBIG = 0.0
      DO 93 I = ISD,N
      TEST = ABS(DX(I)) - YBIG
      IF(TEST .GT. 0.0) YBIG = ABS(DX(I))
   93 CONTINUE
```

```
      DO 94 I=ISD,N
   94 DX(I) =(DX(I) / YBIG) * FSD
C
  999 RETURN
C
      END
```

```
      1 SUBROUTINE CONRE  (N, FDSCR , FIT, SIGMA, NCASE, INACT, NOEX, NSD,
     11  ITER, MODE, DX, NDER2, DER2, IEXIT)
C
      2 DIMENSION MODE(1), DX(1), DER2(1)
C
C           THIS SUBROUTINE DETERMINES WHETHER REMAINING STEPS FOR
C           VARIABLES ARE SMALL ENOUGH TO SATISFY CONVERGENCE
C           CRITERION.
C
     11 IEXIT = 0
     12 IF (N) 998,998,21
C
     21 IF (FDSCR ) 22,22,31
     22 IF (SIGMA - FIT) 998,998,999
C
     31 IF (NOEX) 32,32,999
     32 IF (NSD - (N-INACT)/4) 33,33,999
     33 IF (ITER) 34,34,35
     34 IF (INACT) 35,35,999
     35 NDF = NCASE - N + INACT
     37 IF (NDF) 999,999,38
     38 EPS = SQRT(SIGMA)/FLOAT(NDF)
C
     41 II = 1
     42 DO 45 I = 1,N
     43 IF (MODE(I)) 45,44,44
     44 IF (SQRT(ABS(DER2(II)))*ABS(DX(I)) - FIT*EPS) 45,45,999
     45 II = II + NDER2 + 1
     46 GO TO 998
C
    998 IEXIT = 1
    999 RETURN
C
        END
```

```
      1 SUBROUTINE MATMPY (MI, MJ, MK, MA, MB, MC, A, B, C)
C
      2 DIMENSION A(1), B(1), C(1)
C
C      MATRIX MULTIPLICATION     C = A X B
C            C(MI,MJ) = A(MI,MK) X B(MK,MJ)
C
C REPEAT CYCLES=   ABS VALUES OF MA, MB, MC
C  TO USE INVERSE OF A,B, OR C, MAKE MA,MB, OR MC NEGATIVE
C
C IF A OR B IS DIAGONAL, SET MI OR MJ = 0
C   (TRUE DIMENSIONS MI=MJ=MK)
C
C TO USE CONDENSED DIAGONAL FORM (XCOND(I) = X(I,I))
C  SET MA,MB, OR MC = 0
C
C      THIS VERSION HAS NO DOUBLE PRECISION OPTION
C
     10 KREV = 0
     11 IF (MI) 12,41,12
     12 IF (MJ) 13,51,13
     13 JREM = MJ
        JREMO = JREM
        KRESE  = MK
        IREM = MI
C
     21 IF (MA) 23,22,22
     22 IKOIN  = 1
        IKINC = MA
        GO TO 24
     23 IKOIN  = IABS(MA)
        IKINC = 1
     24 IF (MB) 26,25,25
     25 KJOIN  = MB
        KJINC = 1
        GO TO 27
     26 KJOIN  = 1
        KJINC = IABS(MB)
     27 KJXIN  = 0
        MMC = MC
        GO TO 71
C
     41 IF (MJ) 42,61,42
     42 JREM = MJ
        JREMO = JREM
        IREM = JREM
        KRESE  = 1
        IKOIN  = 1 + IABS(MA)
        MMB = MB
        MMC = MC
     43 IF (MMB) 45,44,44
     44 KJOIN  = MMB
        KJXIN  = 1
        GO TO 46
```

```
45 KJOIN  = 1
   KJXIN  = IABS(MMB)
46 GO TO 71
C
51 JREM = MI
   JREMO = JREM
   IREM = JREM
   KRESE  = 1
   IKOIN  = 1 + IABS(MB)
   MMB = -MA
   MMC = -MC
   KREV = 1
   GO TO 43
C
61 IF (MC) 62,65,62
62 IJDEL = IABS(MC)
   IJO = 1
   DO 64 I = 1,MK
   IJ = IJO
   DO 63 J = 1,MK
   C(IJ) = 0.
63 IJ = IJ + 1
64 IJO = IJO + IJDEL
65 IREM = MK
   KRESE  = 1
   JREM = 1
   JREMO = 1
66 IKOIN  = 1 + IABS(MA)
   KJXIN  = 1 + IABS(MB)
   IJOIN  = 1 + IABS(MC)
   GO TO 101
C
71 IF (MMC) 73,72,72
72 IJINC = MMC
   IJOIN  = 1
   GO TO 74
73 IJINC = 1
   IJOIN  = IABS(MMC)
74 GO TO 101
C
101 IJ = 1
    IJO = IJ
    IK = 1
    IKO = IK
    KJ = 1
102 KJO = KJ
    KJX = KJ
103 KREM = KRESE
    SUM = 0.
C
111 IF (KREV) 112,112,115
112 IF (A(IK)) 113,121,113
113 IF (B(KJ)) 114,121,114
114 SUM = SUM + A(IK)*B(KJ)
    GO TO 121
```

```
  115 IF (A(KJ)) 116,121,116
  116 IF (B(IK)) 117,121,117
  117 SUM = SUM + A(KJ)*B(IK)
C
  121 KREM = KREM - 1
      IF (KREM) 123,123,122
  122 IK = IK + IKINC
      KJ = KJ + KJINC
      GO TO 111
C
  123 C(IJ) = SUM
      IJ = IJ + IJINC
      JREM = JREM - 1
      IF (JREM) 131,131,124
  124 KJO = KJO + KJOIN
      KJ = KJO
      IK = IKO
      GO TO 103
C
  131 IREM = IREM - 1
      IF (IREM) 999,999,132
  132 IJO = IJO + IJOIN
      IJ = IJO
      JREM = JREMO
      IKO = IKO + IKOIN
      IK = IKO
      KJ = KJX + KJXIN
      GO TO 102
C
  999 RETURN
C
       END
```

```
      1 SUBROUTINE RESCAL (N, NDIAG, DIAG, F10, SCAL)
C
C          INPUT=
C            N - NO. OF ITEMS TO BE RESCALED
C            NDIAG - LEFT DIMENSION OF CONTROLLING MATRIX DIAG
C                    (IF IDAG IS ONE-DIMENSIONAL. SET NDIAG=0)
C            DIAG - CONTROLLING MATRIX -
C                   IF TWO-DIMENSIONAL. CONTROLLING ELEMENTS MUST
C                   BE ON DIAGONAL
C            F10 - MAXIMUM CYCLES OF 10 ALLOWED IN SCAL
C          OUTPUT=
C            SCAL - ARRAY OF N RESCALE FACTORS
C
      2 DIMENSION DIAG(1), SCAL(1)
C
     11 IF (N - 1) 999,13,12
     12 IF (F10) 13,13,21
     13 DO 14 I = 1,N
     14 SCAL(I) = 1.0
     15 GO TO 999
C
     21 DMAX = DIAG(1)
     22 II = 1
     23 DO 25 I = 2,N
     24 II = II + NDIAG + 1
     25 DMAX = AMAX1(DMAX,DIAG(II))
     26 IF (DMAX) 13,13,31
C
     31 II = 1
     32 FACT = ALOG(10.0)*F10
     33 DO 39 I = 1,N
     34 IF (DIAG(II)/DMAX) 35,35,37
     35 SCAL(I) = EXP(FACT)
     36 GO TO 38
     37 SCAL(I) = EXP(FACT*(1.0 - EXP(0.5*ALOG(DIAG(II)/DMAX)/FACT)))
     38 II = II + NDIAG + 1
     39 CONTINUE
C
    999 RETURN
C
        END
```

```
      1 SUBROUTINE IDENT (N, S, NS)
C       THIS IS SUBROUTINE IDENT
C
C           CALL IDENT (N, S, NS) CAUSES AN IDENTITY MATRIX (NXN) TO
C           BE PLACED IN S, WITH LEFT DIMENSION OF S = NS.
C
C           IF NS = 0, ACTION IS SKIPPED
C
      2 DIMENSION S(1)
C
     11 IF (N) 999,999,12
     12 IF (NS) 999,999,13
     13 IJX = 1
     14 DO 19 I = 1,N
     15 IJ = IJX
     16 DO 18 J = 1,N
     17 S(IJ) =0.
     18 IJ = IJ + 1
     19 IJX = IJX + NS
C
     21 IJ = 1
     22 INC = NS + 1
     23 DO 25 I = 1,N
     24 S(IJ) = 1.0
     25 IJ = IJ + INC
C
    999 RETURN
C
        END
```

```
      1 SUBROUTINE MOVE (N, A, B)
C       THIS IS SUBROUTINE MOVE
C
      2 DIMENSION A(1), B(1)
C
     11 IF (N) 999,999,12
     12 DO 13 I = 1,N
     13 B(I) = A(I)
C
    999 RETURN
C
        END
```

```
      1 SUBROUTINE ORDS (N, NSEP, MDIAG, DIAG, MS, NS, S)
C       THIS IS SUBROUTINE ORDS
C
      2 DIMENSION DIAG(1), S(1)
C
C       THIS SUBROUTINE INTERCHANGES THE DIAGONAL ELEMENTS OF
C       MATRIX DIAG SO THAT THE NSEP SMALLEST VALUES OCCUPY THE
C       LAST NSEP DIAGONAL POSITIONS IN DESCENDING ORDER.  IT
C       ALSO MAKES THE CORRESPONDING CHANGES IN MATRIX S BY
C       INTERCHANGING COLUMNS
C
C           INPUT=
C              N - ABS(N) = NO. OF ROWS AND COLUMNS IN DIAG AND S
C                  SGN(N) = +, ORDERING IS TO BE BASED ON MAGNITUDE
C                  SGN(N) = -, ORDERING IS TO BE ALGEBRAIC
C              NSEP = NO. OF DIAGONAL ELEMENTS TO BE ORDERED AT BOTTOM
C              MDIAG = LEFT DIMENSION OF DIAG (IF DIAG IS A CONDENSED,
C                           I.E., ONE-DIMENSIONAL ARRAY, SET MDIAG = 0)
C              DIAG = MATRIX WHOSE DIAGONAL ELEMENTS ARE TO BE
C                           REORDERED
C              MS = LEFT DIMENSION OF MATRIX S
C              NS = NO. OF ELEMENTS PER COLUMN IN S
C              S = MATRIX WHOSE COLUMNS ARE TO BE REORDERED
C
C           OUTPUT=
C              DIAG, S = REORDERED MATRICES
C
     11 NN = IABS(N)
     12 IF (NN) 999,999,13
     13 NSTOP = MAX0(1, NN-NSEP)
C
     21 NREP = NN
     22 IF (NREP - NSTOP) 999,999,23
     23 IMIN = 1
     24 IF (N) 25,25,27
     25 XMIN = DIAG(1)
     26 GO TO 31
     27 XMIN = ABS(DIAG(1))
C
     31 II = 1
   2031 IIMIN = 1
     32 DO 45 I = 2,NREP
     33 II = II + MDIAG + 1
     34 IF (N) 35,35,37
     35 X = DIAG(II)
     36 GO TO 41
     37 X = ABS(DIAG(II))
C
     41 IF (X - XMIN) 42,42,45
     42 XMIN = X
     43 IMIN = I
     44 IIMIN = II
     45 CONTINUE
```

```
   51 IF (IMIN - NREP) 52,71,52
   52 X = DIAG(IIMIN)
   53 DIAG(IIMIN) = DIAG(II)
   54 DIAG(II) = X
   55 IF (NS) 71,71,61
C
   61 IJMIN = 1 + MS*(IMIN - 1)
   62 IJ = 1 + MS*(NREP - 1)
   63 DO 68 I = 1,NS
   64 X = S(IJMIN)
   65 S(IJMIN) = S(IJ)
   66 S(IJ) = X
   67 IJMIN = IJMIN + 1
   68 IJ = IJ + 1
C
   71 NREP = NREP - 1
   72 GO TO 22
C
  999 RETURN
C
      END
```

```
      SUBROUTINE DDRG (S,SX,G,XX,MODE,RHS,DXBAR,DXX,DQ,DQX,Y,YBASE,DY,
     1 X,SCALE,D2Q,SCALR)
C
C     THIS IS THE UNIVAC 1108 VERSION OF DDRG
C
C     PREPARED FOR NASA UNDER CONTRACT NAS8-21484
C     BY V. J. LAW, PRINCIPLE INVESTIGATOR
C
C     DOCUMENTATION FOR USING THIS SUBROUTINE MAY BE FOUND
C     IN THE FINAL CONTRACT REPORT
C
      DIMENSION SX(1),SCALR(1),XX(1),X(1),DQX(1),DY(1),Y(1),YBASE(1),
     1 D2Q(1),DQ(1),S(1),DXBAR(1),DXX(1),RHS(1),SCALE(1),        G(1)
      DIMENSION MODE(1)
      DIMENSION ACTION(7),CONST(34)
C
      COMMON /TDCOM/
     1          SSCAL,FIT,FDSCD,FDERV,DSTMN,FSD,SIGMA,DXMAX,DSCAL,NCASE,
     2 NVAR,IPRNT,MAXCT,ICODE,INEW,ICASE,KLUE,ITER,NKICK,NRET,
     3 IVAR,NEWSTR
C
      IF(KLUE) 9,10,10
    9 KLUE = 0
      NRET = 1
   10 GO TO (20,50,60,102), NRET
C
C     LIST STARTING VALUES
C
   20 IF(IPRNT) 31,31,22
   22 WRITE(6,1022)(I,X(I),SCALE(I),I=1,NVAR)
 1022 FORMAT(1H1,4X,29HNONLINEAR EQUATIONS VIA  D.D.,5X//12X,
10221  8HSTARTING/13X,5HVALUE,12X,5HSCALE/1H /(I6,E18.6,E15.4))
      WRITE(6,1023)
     1  FIT,MAXCT,DXMAX,DSCAL,SSCAL,FSD,NVAR,NCASE,FDERV,DSTMN,FDSCD
 1023 FORMAT(1H0,4X,19HPARAMETER     VALUES/7X,
10231  8HFIT     =,E11.4/7X,8HMAXCNT =,I6/7X,
10232  8HDXMAX   =,E11.4/7X,8HDSCALE =,E11.4/7X,8HSSCALE  =,E11.4/7X,
10233  8HFSD     =,E11.4/7X,8HNVAR    =,I6/7X,8HNCASE   =,I6/7X,
10234    8HFDERIV =,E11.4/7X,8HDSTMIN =,E11.4/7X,8HFDSCRD =,E11.4)
C
C     INITIALIZE
C
   31 ITER = 1
      ICODE = 1
      INEW = 1
      IQUIT = 1
      NN = NCASE * NVAR
      KICK = 0
      LKICK = 0
      NSQU = NVAR * NVAR
      CALL IDENT(NVAR,SX,NVAR)
      ILIM = 1
      FXMAX = 1.E+08
C
```

```
C       NOTE-- SX IS S FROM PREVIOUS ITERATION
        DO 32 I=1,NVAR
   32   SCALR(I) = 1.0
C
C       CALC AT BASE POINT
C
   41   CALL MOVE (NVAR,X,XX)
        NRET = 2
        GO TO 998
C
   50   CALL MOVE (NCASE,Y,YBASE)
       CALL MATMPY(1,1,NCASE,-NCASE,NCASE,1,YBASE,YBASE,SIGMA)
C
C       ACCUMULATE G MATRIX AND RHS VECTOR
C
        DO 51 I=1,NVAR
        DQX(I) = SCALR(I) * FDERV
   51   RHS(I) = 0.0
        DO 52 J=1,NSQU
   52   G(J) = 0.0
        IVAR = 1
   55   DO 56 I=1,NVAR
        ISX = I + NVAR * (IVAR - 1)
   56   X(I) = XX(I) + DQX(IVAR) * SCALE(I) * SX(ISX)
        NRET = 3
        GO TO 998
C
   60   DO 61 ICASE = 1, NCASE
        IJ = ICASE + NCASE*(IVAR - 1)
   61   DY(IJ)     =(Y(ICASE) - YBASE(ICASE))/DQX(IVAR)
C
        IF(IVAR - NVAR) 72,80,80
   72   IVAR = IVAR + 1
        GO TO 55
C
   80   CALL MATMPY(NVAR,NVAR,NCASE,-NCASE,NCASE,NVAR,DY,DY,G)
        CALL MATMPY(NVAR,1,NCASE,-NCASE,NCASE,NVAR,DY,YBASE,RHS)
C
C       APPLY DD LOGIC FOR SEARCH VECTOR
C
        CALL DIAG(NVAR,NVAR,G,S,X,1.E-08)
        CALL ORDS (-NVAR,NVAR,NVAR,G,NVAR,NVAR,S)
        II = 1
        DO 83 I=1,NVAR
        D2Q(I) = G(II)
   83   II = II + NVAR + 1
        CALL MATMPY(NVAR,NVAR,NVAR,NVAR,NVAR,NVAR,SX,S,G)
        CALL MATMPY(1,NVAR,NVAR,1,NVAR,1,RHS,S,DQ)
        CALL TDVEC(NVAR,NCASE,SIGMA,DQ,0,D2Q,FSD,SSCAL,FDSCD,DXBAR,
      1  MODE,INACT,NSD,NOEX)
        CALL MATMPY(NVAR,1,NVAR,NVAR,1,1,G,DXBAR,DXX)
        CALL MATMPY(1,1,NVAR,1,1,1,DQ,DXBAR,DQDZ)
C
C       CHECK STEP SIZE
C
```

```
      CALL MATMPY(1,1,NVAR,1,1,1,DXBAR,DXBAR,DIST)
      DIST = SQRT(DIST)
      DO 84 I=1,NVAR
   84 DXX(I) = SCALE(I) * DXX(I)
      IF(IPRNT - 1) 87,87,85
 1084 IF(IPRNT - 1)88,85,88
   85 WRITE(6,1085)
     1                ITER,INACT,DIST,FSD,SIGMA
 1085 FORMAT(1H0,3X,7HITER NO,4X,10H-INACTIVE-,5X,4HDIST,8X,3HFSD,
     1 61X,5HSIGMA/I8,10X,I3,5X,2F12.5,54X,E15.6)
      WRITE(6,2085)
 2085 FORMAT(1H0,17X,10HPARAMETERS,34X,18HINTERNAL VARIABLES/1H )
      DO 86 I=1,NVAR
      WRITE(6,1086)
     1                I,XX(I),DXX(I),D2Q(I),DQ(I),DXBAR(I),MODE(I)
   86 CONTINUE
 1086 FORMAT(I6,2E16.6,8X,1H*,E15.4,E14.4,F11.5,I5)
   88 GO TO (87,1200,1202,1204),IQUIT
C
C     TEST FOR CONVERGENCE
C
   87 CALL CONRE(NVAR,FDSCD,FIT,SIGMA,NCASE,INACT,NOEX,NSD,
     1                ITER,MODE,DXBAR,0,D2Q,IEXIT)
      IF(IEXIT)90,90,200
   90 IF(ITER - MAXCT) 91,202,202
   91 IF(KICK - NKICK) 1091,204,204
 1091 IF(LKICK) 4091,4091,2091
 2091 IF(IPRNT - 1)4091,4091,3091
 3091 WRITE(6,5091) KICK
 5091 FORMAT(1H0,4X,39HABNORMALLY SMALL STEP EXIT SUPPRESSED -,I3)
      LKICK = 0
 4091 IF(DIST - DXMAX) 92,92,93
   92 FX  = 1.0
      GO TO 94
   93 FX  = DXMAX / DIST
   94 CONTINUE
C
C     SET UP ONE-DIMENSIONAL SEARCH
C
      FXMIN = FDERV / DIST
      QO = SIGMA
      Q = QO
      QS = 0.0
      ICON = 0
      IF(IPRNT - 1)317,317,316
  316 PRINT 1316
 1316 FORMAT(1H0,4X,5HSRCHS/7X,2HFX,10X,6HACTION,14X,5HSIGMA)
  317 CALL SRCHS(ICON,QS,DQDZ,FX,NVAR,FXMIN,FXMAX,ILIM,ACTION,CONST)
C
      IF(ICON + 2)206,206,322
  322 IF(IPRNT - 1)326,326,323
  323 IF(ACTION(1))324,324,325
  324 PRINT 1324,ACTION(2),(ACTION(I),I=5,7),Q
 1324 FORMAT(2X,1PE13.4,4X,3A4,1PE17.5)
      GO TO 326
```

```
  325  PRINT 1325,(ACTION(I),I=2,7),Q
 1325  FORMAT(5X,6A4,1PE19.5)
  326  IF(ICON) 122,122,331
C
  331  DO 332 I=1,NVAR
  332  X(I) = XX(I) + FX*DXX(I)
       NRET = 4
       GO TO 998
C
  102  CALL MATMPY(1,1,NCASE,-NCASE,NCASE,1,Y,Y,Q)
       QS = Q - Q0
       GO TO 317
C
  122  IF(NSD) 124,124,123
  123 CALL NEWFSD(FX,Q,FSD)
  124  DO 125 I=1,NVAR
  125  X(I) = XX(I) + FX * DXX(I)
       CALL MOVE (NSQU,G,SX)
      CALL RESCAL(NVAR,0,D2Q,DSCAL,SCALR)
       XVAR = NVAR
       XSD = NSD + 1
       IF(FX * DIST - FDERV * SQRT(XVAR) / XSD) 126,126,127
  126  KICK = KICK + 1
       LKICK = 1
  127  CONTINUE
       ITER = ITER + 1
       GO TO 41
C
C      TERMINATION (SET ICODE = 0)
C            KLUE = 0      O.K.
C                   1      MAX NO ITERATIONS
C                   3      LAST STEP ABNORMALLY SMALL
C                  -1      SEARCH NO GO
  200  KLUE = 0
       IQUIT = 2
       GO TO 1084
 1200  IF(IPRNT)999,999,201
  201  WRITE(6,1201)
 1201  FORMAT(1H0,4X,18HD.D. FINISH NORMAL)
       GO TO 999
C
  202  KLUE = 1
       IQUIT = 3
       GO TO 1084
 1202  IF(IPRNT)999,999,203
  203  WRITE(6,1203)
 1203  FORMAT(1H0,4X,41HD.D. HAS REACHED MAX NUMBER OF ITERATIONS)
       GO TO 999
C
  204  KLUE = 3
       IQUIT = 4
       GO TO 1084
 1204  IF(IPRNT)999,999,205
  205  WRITE(6,1205)
 1205  FORMAT(1H0,4X,55HD.D. TERMINATED DUE TO LAST STEP BEING ABNORMALL
```

```
     1Y SMALL)
      GO TO 999
C
 206  KLUE = -1
      WRITE(6,1206)
 1206 FORMAT(1H0,4X,59HSOLUTION   NOT POSSIBLE,ONE-D SEARCH TRUNCATED B
     1EYOND FXMIN/5X,52HERROR PROBABLY DUE TO INCORRECT FUNCTION CALCULA
     2TION)
C
  999 ICODE = 0
      IF(KLUE .NE. 0) NEWSTR = NEWSTR + 1
      IF(NEWSTR .GT. 5) NEWSTR = 0
  998 INEW = NRET - 2
      RETURN
C
      END
```

```
      SUBROUTINE NEWSCL(X,SCALE)
      DIMENSION X(1),SCALE(1)
      COMMON /TDCOM/ COM(9),KCOM(13)
      INEW = KCOM(6)
      NVAR = KCOM(2)
      IF(INEW)8,3,8
    3 DO 7 I=1,NVAR
      IF(ABS(X(I)) - 1.E-08)6,6,5
    5 SCALE(I) = ABS(X(I))
      GO TO 7
    6 SCALE(I) = 1.E-08
    7 CONTINUE
    8 RETURN
      END
```

```
      SUBROUTINE DDSET(KODE,NVAR,X,XMIN,XMAX,SCALE,NEWSTR)
      DIMENSION X(1),XMIN(1),XMAX(1),SCALE(1)
      Z = NEWSTR
      CALL FPRNDM(Z)
      IF(KODE .EQ. 0) GO TO 3
      CALL FPRNDM(Z)
      DO 2 I=1,NVAR
    2 X(I) = (XMIN(I) + XMAX(I)) * Z
    3 DO 4 I=1,NVAR
      SCALE(I) = ABS(X(I))
      IF(SCALE(I) .LT. 1.E-08) SCALE(I) = 1.E-08
    4 CONTINUE
      KODE = 0
      RETURN
      END
```

```
  1 SUBROUTINE SRCHS (ICON, Q, DQ, FX, NVAR, FXMIN, FXMAX, ILIM,
 11   ACTION, CONST)
C
  3 DIMENSION ACTION(7), V(41)
  4 DIMENSION CONST(34), ACONST(9)
  5 EQUIVALENCE (ACONST(1),ICONX),(ACONST(2),QO),(ACONST(3),DQO),
 51   (ACONST(4),FXOPT),(ACONST(5),FXP),(ACONST(6),QP),
 52   (ACONST(7),FXPP),(ACONST(8),QPP),(ACONST(9),ILIMX)
  6 DATA V/4HDQ N,4H.O.K,4H.   ,3*4H    ,4HTRY ,4HOPTI,4HM   ,4HNG, ,
 61   4HSTOP,4HCUT ,4HSTOP,4HPED ,4HOPTI,4HM OK,4HTAKE,4H BES,4HT    ,
 62   4HQUIT,4H ON ,4HPREV,4HTAKE,4H PRE,4HV    ,4HTRY ,4HMORE,4HQUIT,
 63   4H ON ,4HLOW ,4HREVE,4HRSE ,4HNO G,4HO    ,4HNO , ,4HTRY ,4HMORE,
 64   4H    ,4H  LI,4HMITE,4HD    /
C
  8 CALL MOVE (3, V(4), ACTION(5))
  9 CALL MOVE (9, CONST, ACONST)
 10 XNV = FLOAT(NVAR+3)
 11 IF (ICON) 12,12,51
 12 ICONX = ICON
 13 ILIMX = ILIM
 14 ILIM = 0
 15 QO = Q
 16 DQO = DQ
2016 IF (ICONX) 31,17,17
 17 IF (DQ) 31,21,21
C
C          DQ N.G., EXIT WITH ICON = -2, FX = 0
C
 21 FX = 0.
 22 ICON = -2
 23 ACTION(1) = 1.0
 24 CALL MOVE (3, V(1), ACTION(2))
 25 GO TO 999
C
 31 CONTINUE
C
C          EXIT FOR FIRST POINT WITH INITIAL FX  (ICON = 1)
C
 41 ICON = 1
 42 GO TO 980
C
C          PUT FX IN ACTION,
C          TAKE PRIMARY BRANCH
C
 51 ACTION(1) = 0.
 52 ACTION(2) = FX
 54 GO TO (61,81,151,181,211,261,271,501,531,541), ICON
C
C          ICON = 1
C          CHOOSE BETWEEN CUT AND GROW
C
 61 IF (Q - QO) 201,2061,2061
2061 IF (FX - FXMIN) 83,83,62
C
```

```
C             CUT CHOSEN
C             OPTIMIZE USING INITIAL SLOPE AND 2 POINTS
C             IF OPT MORE THAN 1/4, ACCEPT TENTATIVELY (ICON = 2)
C
   62 IF (ICONX) 401,2062,2062
 2062 ASSIGN 71 TO NCAL3
   63 CALL SETR (1, 0, Q0, 1, CONST(10))
   64 CALL SETR (1, FX, Q, 2, CONST(10))
   65 CALL SETR (2, Q0, DQ0, 3, CONST(10))
   66 CALL CALXR (3, CONST(10))
   67 GO TO NCAL3, (71,204)
C
   71 CALL OPTR (FXOPT, DELOPT, CONST(10))
   73 FXP = FX
   74 QP = Q
   75 IF (FXOPT - 0.25*FX) 121,121,76
   76 CALL MOVE (3, V(7), ACTION(5))
   77 ICON = 2
   78 FX = FXOPT
   79 GO TO 930
C
C             ICON = 2
C             EXAMINE TENTATIVE POINT
C             IF O.K., KEEP
C             IF N.O.K. AND FX NOT TOO SMALL, CUT AND IGNORE DQ0
C             IF N.O.K. AND FX TOO SMALL, QUIT
C
   81 IF (Q - Q0) 141,82,82
   82 IF (FX - FXMIN) 83,83,2082
 2082 ICONX = -1
 3082 GO TO 401
   83 CALL MOVE (2, V(10), ACTION(5))
   84 FXOPT = 0.
   85 CONTINUE
C
   91 Q = Q0
   92 FX = 0.
   93 GO TO 900
C
  111 ASSIGN 71 TO NCAL4
  112 CALL SETR (1, 0, Q0, 1, CONST(10))
  113 CALL SETR (1, FX, Q, 2, CONST(10))
  114 CALL SETR (1, FXP, QP, 3, CONST(10))
  115 CALL SETR (2, Q0, DQ0, 4, CONST(10))
  116 CALL CALXR (4, CONST(10))
  117 GO TO NCAL4, (71,161,196,213,544)
C
C              IF CUT CHOSEN AND OPT L.T.E. 1/4, USE 1/4 UNLESS TOO SMALL
C                 (IF TOO SMALL, QUIT)
C
  121 IF (0.25*FX - FXMIN) 122,131,131
  122 CALL MOVE (3, V(12), ACTION(5))
  123 IF (Q - Q0) 124,84,84
  124 FXOPT = FX
  125 GO TO 85
```

```
C
  131 ACTION(5) = V(12)
  132 FX = 0.25*FX
  133 ICON = 3
  134 GO TO 930
C
C           OPTIM O.K., QUIT
C
  141 CALL MOVE (2, V(15), ACTION(5))
  142 GO TO 900
C
C           ICON = 3
C           EXAMINE CUT POINT
C           IF NOK, CUT AGAIN IF POSSIBLE
C           IF OK, OPTIMIZE
C           IF OPTIM L.T.E. 1/4 FX, CUT AGAIN IF POSSIBLE
C           IF OPTIM G.T. 1/4 FX, TRY OPTIM (ICON = 4)
C
  151 IF (Q - Q0) 152,62,62
  152 IF (ICONX) 155,153,153
  153 ASSIGN 161 TO NCAL4
  154 GO TO 112
  155 ASSIGN 161 TO NCAL3P
  156 GO TO 411
C
  161 CALL OPTR (FXOPT, DELOPT, CONST(10))
  162 IF (-Q/XNV + (DELOPT - Q)) 163,163,601
  163 FXP = FX
  164 QP = Q
  165 IF (FXOPT - 0.25*FX) 166,166,167
  166 IF (ICONX) 131,121,121
  167 IF (FXOPT - FXP) 171,171,168
  168 CALL MOVE (3, V(17), ACTION(5))
 2168 IF (Q - QP) 3168,5168,5168
 3168 FXOPT = FX
 4168 GO TO 169
 5168 FXOPT = FXP
 6168 FX = FXOPT
  169 GO TO 142
C
  171 CALL MOVE (3, V(7), ACTION(5))
  172 FX = FXOPT
  173 ICON = 4
  174 GO TO 930
C
C           ICON = 4
C           EXAMINE TRIAL OPTIM AND CHOOSE, QUIT
C
  181 IF (Q - QP) 141,141,182
  182 CALL MOVE (3, V(23), ACTION(5))
  183 FXOPT = FXP
  184 CONTINUE
C
  191 IF (ICONX) 194,192,192
  192 ASSIGN 196 TO NCAL4
```

```
  193 GO TO 112
  194 ASSIGN 196 TO NCAL3P
  195 GO TO 411
  196 Q = QP
  197 FX = FXOPT
  198 GO TO 900
C
C              GROWTH CHOSEN
C              TRY 2*FX
C
  201 IF (ICONX) 206,202,202
  202 ASSIGN 204 TO NCAL3
  203 GO TO 63
  204 CALL OPTR (FXOPT, DELOPT, CONST(10))
  205 IF (-Q/XNV + (DELOPT - Q)) 2205,2205,601
 2205 IF (FXOPT - FX) 3205,206,206
 3205 FXP = FX
 4205 QP = Q
 5205 CALL MOVE (3, V(7), ACTION(5))
 6205 FX = FXOPT
 7205 ICON = 10
 8205 GO TO 930
C
  206 ICON = 5
  207 FXP = FX
  208 QP = Q
  209 CALL MOVE (2, V(26), ACTION(5))
 2209 FX = 2.0*FX
 3209 GO TO 930
C
C              ICON = 5
C              TEST FOR FURTHER GROWTH
C
  211 IF (ICONX) 212,2211,2211
 2211 ASSIGN 213 TO NCAL4
 3211 GO TO 112
  212 ASSIGN 213 TO NCAL3P
 2212 GO TO 411
  213 CALL OPTR (FXOPT, DELOPT, CONST(10))
  214 IF (FXOPT - FX) 221,215,215
  215 IF (Q - QP) 216,241,241
  216 IF (-Q/XNV + (DELOPT - Q)) 2216,2216,601
 2216 ICON = 7
  217 FXPP = FXP
  218 QPP = QP
  219 GO TO 207
C
C              RESTRAIN GROWTH
C
  221 IF (Q - QP) 226,222,222
  222 IF (-QP/XNV + (DELOPT - QP)) 231,231,224
  224 CALL MOVE (3, V(23), ACTION(5))
  225 GO TO 242
  226 IF (-Q/XNV + (DELOPT - Q)) 227,227,601
  227 FXP = FX
```

```
  228 QP = Q
C
  231 CALL MOVE (3, V(7), ACTION(5))
  232 ICON = 6
  233 FX = FXOPT
  234 GO TO 930
C
  241 CALL MOVE (3, V(20), ACTION(5))
  242 FXOPT = FXP
  243 CONTINUE
C
  251 FX = FXP
  252 Q = QP
  253 GO TO 900
C
C           ICON = 6
C           RETURN FROM ATTEMPT AT OPTIM ON GROWTH
C
  261 IF (Q - QP) 141,141,262
  262 CALL MOVE (3, V(28), ACTION(5))
  263 GO TO 242
C
C           ICON = 7
C           RETURN FROM FURTHER GROWTH
C
  271 CALL SETR (1, FX, Q, 1, CONST(10))
  272 CALL SETR (1, FXP, QP, 2, CONST(10))
  273 CALL SETR (1, FXPP, QPP, 3, CONST(10))
  274 CALL CALXR (3, CONST(10))
  275 CALL OPTR (FXOPT, DELOPT, CONST(10))
  276 GO TO 214
C
C           INITIAL CUT WITH NO DQO
C           TEST SIZE OF FX FOR POSSIBLE REVERSING
C
  401 FXP = FX
  402 QP = Q
  403 IF (0.25*FX - FXMIN) 404,404,131
  404 CALL MOVE (2, V(31), ACTION(5))
  405 FX = -AMIN1(FX,FXMIN)
  406 ICON = 8
  407 GO TO 930
C
C           3 PT FIT USING Q0
C
  411 CALL SETR (1, 0, Q0, 1, CONST(10))
  412 CALL SETR (1, FX, Q, 2, CONST(10))
  413 CALL SETR (1, FXP, QP, 3, CONST(10))
  414 CALL CALXR (3, CONST(10))
  415 GO TO NCAL3P,(161,196,213,503)
C
C           ICON = 8
C           RETURN FROM REVERSAL
C
  501 ASSIGN 503 TO NCAL3P
```

```
  502 GO TO 411
  503 IF (DERR(O,CONST(10))) 511,521,521
C
  511 CALL OPTR (FXOPT, DELOPT, CONST(10))
  512 CALL MOVE (3, V(7), ACTION(5))
  513 ICON = 9
  514 FX = FXOPT
  515 GO TO 930
C
  521 FX = 0.
  522 FXOPT = 0.
  523 ICON = -1
  524 Q = Q0
  525 CALL MOVE (2, V(33), ACTION(5))
  526 GO TO 930
C
C             ICON = 9
C             RETURN AFTER REVERSAL AND OPTIMIZATION
C
  531 IF (Q - Q0) 141,521,521
C
C             ICON = 10
C             RETURN AFTER ATTEMPTED IMMEDIATE OPTIMIZATION
C
  541 IF (Q - QP) 141,141,542
  542 ASSIGN 544 TO NCAL4
  543 GO TO 112
  544 CALL OPTR (FXOPT, DELOPT, CONST(10))
  545 IF (FXOPT - FXP) 224,224,546
  546 IF (-QP/XNV + (DELCPT - QP)) 551,551,224
C
  551 ICON = 5
  552 FX = 2.0*FXP
  553 CALL MOVE (3, V(35), ACTION(5))
  554 GO TO 930
C
C             TAKE EXIT
C
  601 FXOPT = FX
  602 ACTION(5) = V(17)
  603 GO TO 142
C
C             900 EXIT
C
  900 ICON = 0
  901 IF (ILIMX) 903,903,902
  902 IF (FX - FXMAX) 903,905,905
  903 ILIM = 0
  904 GO TO 999
  905 ILIM = ILIMX
  906 GO TO 999
C
  930 IF (ILIMX) 936,936,931
  931 IF (FX - 0.999*FXMAX) 936,936,932
```

```
  933 IF (ILIM) 934,934,900
  934 ILIM = ILIMX
  935 GO TO 999
  936 ILIM = 0
  937 GO TO 999
C
C           980 EXIT
C
  980 IF (ILIMX) 984,984,981
  981 IF (FX - FXMAX) 984,982,982
  982 ACTION(1) = 1.0
  983 CALL MOVE (4, V(38), ACTION(2))
      GO TO 930
  984 ACTION(1) = 1.0
  985 CALL MOVE (3, V(4), ACTION(2))
      GO TO 930
C
  999 CALL MOVE (9, ACONST, CONST)
      RETURN
C
       END
```

```
      1 SUBROUTINE SETR (K, X, Y, I, CONST)
C
      2 DIMENSION CONST(25), ACONST(25)
      4 DIMENSION A(4,4), B(4), C(4)
      5 EQUIVALENCE (ACONST(1),A), (ACONST(17),B),(ACONST(21),C),
     51  (ACONST(25),KR)
C
      6 CALL MOVE (25, CONST, ACONST)
     11 GO TO (21,31), K
C
C            K = 1
C            SET UP FOR POINT WITH FX = X AND Q = Y
C
     21 A(I,1) = 1.0
     22 A(I,2) = X
     23 A(I,3) = X**2
     24 A(I,4) = -X*Y
     25 B(I) = Y
     26 GO TO 999
C
C            K = 2
C
     31 A(I,1) = 0.
     32 A(I,2) = 1.0
     33 A(I,3) = 0.
     34 A(I,4) = -X
     35 B(I) = Y
     36 GO TO 999
C
    999 CALL MOVE (25, ACONST, CONST)
        RETURN
C
         END
```

```
  1 SUBROUTINE CALXR (NN, CONST)
C
  2 DIMENSION CONST(25), ACONST(25)
  3 DIMENSION W(4,4), WW(4,4), X(4), BB(4)
  4 DIMENSION A(4,4), B(4), C(4)
  5 EQUIVALENCE (ACONST(1),A), (ACONST(17),B),(ACONST(21),C),
 51   (ACONST(25),KR)
  6 CALL MOVE (25, CONST, ACONST)
C
    KR = 0
  7 N = NN
 11 CALL MATMPY (N, 1, N, -4, 1, 1, A, B, X)
 12 CALL MATMPY (N, N, N, -4, 4, 4, A, A, W)
 13  CALL DIAG(N,4,W,WW,BB,1.E-08)
 14 CALL MATMPY (N, 1, N, -4, 1, 1, WW, X, BB)
 15 DO 16 I = 1,N
 16 BB(I) = BB(I)/W(I,I)
 17 CALL MATMPY (N, 1, N, 4, 1, 1, WW, BB, C)
 18 IF (N - 3) 19,19,201
 19 C(4) = 0.
 20 GO TO 999
C
201 IF (KR) 202,202,999
202 IF (C(4)) 211,999,999
C
211 DO 222 I = 1,4
212 IF (A(I,1)) 221,213,221
213 A(I,3) = -2.0*A(I,4)
214 A(I,4) = 3.0*A(I,4)**2
215 GO TO 222
C
221 A(I,4) = A(I,2)*A(I,3)
222 CONTINUE
223 KR = 1
224 GO TO 11
C
999 CALL MOVE (25, ACONST, CONST)
    RETURN
C
     END
```

```
    1 SUBROUTINE OPTR (FXOPT, DELOPT, CONST)
C
    2 DIMENSION CONST(25), ACONST(25)
    3 DOUBLE PRECISION RR
    4 DIMENSION A(4,4), B(4), C(4)
    5 EQUIVALENCE (ACONST(1),A), (ACONST(17),B),(ACONST(21),C),
   51   (ACONST(25),KR)
    6 CALL MOVE (25, CONST, ACONST)
C
   10 IF (KR) 11,11,41
   11 D3 = C(2) - C(1)*C(4)
   12 IF (D3) 21,13,13
   13 FXOPT = -1.0
   14 DELOPT = 0.
   15 GO TO 999
C
   21 CONTINUE
   22 IF (C(3)) 23,23,25
   23 FXOPT = 1.0E20
 2023 DELOPT = -1.0E30
   24 GO TO 999
   25 CONTINUE
   26 GO TO 101
C
   31 FXOPT = AMAX1 (R1,R2)
   32 IF (FXOPT - 1.0E20) 33,2023,2023
   33 DELOPT = (D3 + C(3)*FXOPT) * FXOPT / (1.0 + C(4)*FXOPT)
   34 GO TO 999
C
   41 IF (C(4)) 23,11,42
   42 IF (C(2)) 44,43,43
   43 IF (C(3)) 51,13,13
   44 IF (C(3)) 51,45,51
   45 FXOPT = SQRT(-C(2)/3.0/C(4))
   46 GO TO 62
C
   51 DA = C(2)/C(3)
   52 D3 = C(3)/3.0/C(4)
   53 IF (DA - D3) 54,54,13
   54 RR = DSQRT (1.0D0 - DBLE(DA/D3))
   55 R1 = D3 * SNGL(-1.0D0 + RR)
   56 R2 = D3 * SNGL (-1.0D0 - RR)
C
   61 FXOPT = AMAX1(R1,R2)
   62 IF (FXOPT - 1.0E20) 63,2023,2023
   63 DELOPT = FXOPT * (C(2) + FXOPT * (C(3) + FXOPT*C(4))) + C(1)
   64 IF (DELOPT) 999,999,13
C
  101 IF (C(4)) 111,102,111
  102 R1 = -D3/2.0/C(3)
  103 R2 = R1
  104 GO TO 31
C
  111 RR = DSQRT (1.0D0 - DBLE(D3*C(4)/C(3)))
```

```
  112 R1 = SNGL (-1.0D0 + RR)/C(4)
  113 R2 = SNGL (-1.0D0 - RR)/C(4)
  114 GO TO 31
C
  999 CALL MOVE (25, ACONST, CONST)
      RETURN
C
      END
```

```
      1 FUNCTION DERR (FX, CONST)
C
      2 DIMENSION CONST(25), ACONST(25)
      4 DIMENSION A(4,4), B(4), C(4)
      5 EQUIVALENCE (ACONST(1),A), (ACONST(17),B),(ACONST(21),C),
     51   (ACONST(25),KR)
      6 CALL MOVE (25, CONST, ACONST)
C
     10 IF (KR) 11,11,13
     11 DERR = (C(2) - C(1)*C(4) + 2.0*C(3)*FX + C(3)*C(4)*FX**2) /
    111   (1.0 + C(4)*FX)**2
     12 GO TO 999
     13 DERR = C(2) + FX*(2.0*C(3) + 3.0*C(4)*FX)
     14 GO TO 999
C
    999 CALL MOVE (25, ACONST, CONST)
        RETURN
C
        END
```

```
      SUBROUTINE DIAG(N,NDIM,A,T,SCR,TFACT)
C
C     THIS SUBROUTINE DIAGONALIZES A SQUARE,SYMMETRIC,POSITIVE
C       SEMI-DEFINITE MATRIX, A, ACCORDING TO THE TRANSFORMATION
C
C
C          (T)T A T = DIAGONAL MATRIX
C     INPUT=
C       NDIM     = DIMENSION OF T,TT,AND A
C       A        = MATRIX TO BE DIAGONALIZED
C       TFACT    TOLERANCE FACTOR (1.E$08 RECOMMENDED)
C       N        = SIZE OF A (I.E.,A IS N X N)
C
C     OUTPUT=
C       T        = THE TRANSFORMATION MATRIX
C       A        = THE DIAGONALIZED VERSION OF A
C
C     NOTE=  SCR IS A SCRATCH ARRAY
C
      DIMENSION A(1),T(1),SCR(1)
C
      LOC(IX,JX) = (JX - 1) * NDIM + IX
      Z = 1./SQRT(2.)
      ABIG = 0.0
      DO 200 I=1,N
      DO 200 J=1,N
      IJ = LOC(I,J)
      IF(ABS(A(IJ )) .GT. ABIG) ABIG = ABS(A(IJ ))
  200 CONTINUE
      TF = ABIG * TFACT
C
      L = 1
      NN = N - 1
C
      CALL IDENT(N,T,NDIM)
C
    1 I = L + 1
   11 NSW = 0
      LLX= LOC(L,L)
      IF(ABS(A(LLX)) .LE. TF   ) GO TO 10
C              NON-ZERO DIAGONAL FOUND---NORMAL PROCEDURE
C                   USED TO ZERO OFF-DIAGONAL ELEMENTS
      DO 3 J=I,N
      LJ = LOC(L,J)
      FACT = -A(LJ ) / A(LLX)
      A(LJ ) = 0.0
      DO 14 LL = 1,N
      LLJ = LOC(LL,J)
      LLL = LOC(LL,L)
   14 T(LLJ ) = T(LLJ ) + FACT*T(LLL )
      DO 2 K=I,N
      KJ = LOC(K,J)
      KL = LOC(K,L)
    2 A(KJ ) = A(KJ ) + FACT * A(KL )
```

```
  3   CONTINUE
      DO 4 J=I,N
      JL = LOC(J,L)
  4   A(JL ) = 0.0

      IF(L .GE. NN) GO TO 8
      L = L + 1
      GO TO 1
 10   M   = I + NSW
      IF(M   .GT. N) GO TO 20
      MM = LOC(M,M)
      IF(ABS(A( MM  )) - TF   ) 12,12,13
 12   NSW = NSW + 1
      GO TO 10
                ZERO DIAGONAL FOUND UPSTREAM--NON-ZERO DIAGONAL FOUND DOWNSTREAM
                    SWAP OF ROW-COLUMN TO GET NON-ZERO DIAGONAL UPSTREAM
 13   DO 5 K=L,N
      KL = LOC(K,L)
      KM = LOC(K,M)
      SCR(K) = A(KL )
      A(KL )=A(KM )
  5   A(KM )=SCR(K)
      DO 49 K = 1,N
      KL = LOC(K,L)
      KM = LOC(K,M)
      SCR(K) = T(KL )
      T(KL ) = T(KM )
 49   T(KM ) = SCR(K)
      DO 6 K=L,N
      LK = LOC(L,K)
      MK = LOC(M,K)
      SCR(K) = A(LK )
      A(LK ) = A(MK )
  6   A(MK ) = SCR(K)
      GO TO 11
C
 20   II = I - 1
 25   II = II + 1
      IF ( II .LE. N ) GO TO 26
C                 ALL REMAINING DIAGONALS ARE ZERO
C                    ALSO ALL OFF DIAGONAL ELEMENTS OF ROW L ARE ZERO
C                    THEREFORE PUT ZERO ON DIAGONAL AND GO TO NEXT ROW
C                    TO SEARCH FOR NON-ZERO OFF-DIAGONAL ELEMENT
      L = L + 1
      I = L + 1
      IF ( I .GT. N ) GO TO 8
      GO TO 20
 26   LII = LOC(L,II)
      IF( ABS(A(LII )) .LT. TF ) GO TO 25
C
C                 ALL REMAINING DIAGONALS ARE ZERO BUT OFF DIAGONAL FOUND TO BE
C                    NON-ZERO----ROTATION PERFORMED
      DO 42 K = L,N
      KII = LOC(K,II)
      KL = LOC(K,L)
```

```
      SCR(K) = A(KII )
      A(KII ) = (A(KII ) + A(KL )) * Z
   42 A(KL ) = (-SCR(K) + A(KL ))*Z
      DO 46 K = L,N
      IIK = LOC(II,K)
      LK = LOC(L,K)
      SCR(K) = A(IIK )
      A(IIK ) = (A(IIK ) + A(LK ))*Z
   46 A(LK ) = (-SCR(K) + A(LK ))*Z
      DO 44 K = 1,N
      KII = LOC(K,II)
      KL = LOC(K,L)
      SCR(K) = T(KII )
      T(KII ) = (T(KII ) + T(KL ))*Z
   44 T(KL ) = (-SCR(K) + T(KL ))*Z
      GO TO 11

    8 RETURN
      END
```

```
C RHF FPRNDM    0778H
      SUBROUTINE FPRNDM (Z)
      IF (Z)2,1,2
    1 NS = 7777777
      NG = 2**17 + 3
      B = 1./34535973836 7.
    2 NS = NG*NS
      Z = NS
      Z = Z*B
      RETURN
      END
```