A NEW DIRECT SEARCH METHOD FOR UNCONSTRAINED FUNCTION OPTIMIZATION

By

FRED EARL WITZ

Bachelor of Science in Electrical Engineering
North Dakota State University
Fargo, North Dakota
1968

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1970

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
DOCTOR OF PHILOSOPHY
May, 1976

A NEW DIRECT SEARCH METHOD FOR

UNCONSTRAINED FUNCTION

OPTIMIZATION

Thesis Approved:

Charles M. Bacon
Thesis Adviser

Bennett Basore

Paul A. McCollum

J. P. Chandler

N. M. Durham
Dean of Graduate College

PREFACE

This research was concerned with the development of a new optimization method. Interest in optimization originated in a search for a method for parameter estimation in the doctoral research of my brother, John Witz.

The many hours of assistance of committee chairman, Dr. Charles Bacon, in preparing the final manuscript is greatfully acknowledged. Special thanks go to Dr. John Chandler for reviewing the technical aspects of the thesis and for calling attention to the problem of scaling discussed in Chapter IV. Gratitude is also expressed to the other committee members, Dr. Bennett Basore and Prof. Paul McCollom, for their time, and to all of the committee for their patience during periods of slow progress.

Sincere appreciation is expressed to Barbara Strategier for her excellent typing of both drafts of the thesis under the pressure of deadlines. Appreciation is also expressed to Charles Hanes and Trecia Markes for proofreading the thesis, to Jim Hysaw for preparing the figures and to Sandy Coughlin for keypunching the cards used in Table II.

The patience and support of my family are deeply appreciated. Special gratitude is expressed to Myron Paine for his counsel and support during difficult periods of the research. This thesis is dedicated to Susan and her friends, whose prayers and encouragement made it possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Motivation

This research develops a new direct search method for unconstrained smooth function optimization. The method applies to any such function which can be evaluated by the computer. It was designed for the case where only the value of the function is available (not derivatives) and evaluation is expensive. The resulting method has properties which could make it useful in other situations also.

A method which requires fewer function evaluations than other methods logically must retain more information or do more calculation. The only information available is the value of the function at a number of sample points. A good way to organize the information is to use a simple function to model the function. The usual model is a quadratic function. Some previous methods retain equivalent information, but it is not always organized as a model.

Any method using this approach must contend with several difficulties. First, for the case where only the value of the function is available, the derivatives of the function cannot be used to create the model. Second, the optimum of the model often lies outside the region where the model is valid. For this reason it is advantageous to use direction vectors and search in one direction at a time. As shown in

Chapter II, the eigenvectors of second partial matrix derivatives of the function are desirable directions for this purpose. The matrix of second partial derivatives will be referred to here as the curvature matrix. A third possible difficulty is the calculation of the location of the optimum of the model. The direct solution requires the inverse of the curvature matrix, which is usually a lengthy calculation. Finally, creating a complete model and calculating the location of the optimum before each movement toward the goal is usually highly efficient.

In the new method, the key to the solution of these difficulties is the Jacobi method for finding eigenvectors. The Jacobi method is an iterative method which works with two eigenvectors at a time. Using the same organization, the new method works in the plane of two direction vectors at a time, sampling the function, fitting part of the model, correcting the two direction vectors, and searching for the optimum within the plane. This procedure is repeated for all pairs of direction vectors. Eventually the entire model is fitted, the direction vectors converge toward eigenvectors, and the base location moves toward the optimum.

Advantages of the New Method

The organization of the new method reduces function evaluation by eliminating the requirement for univariate optimizations. Instead, only sufficient samples to fit the function are required. In addition, the samples used to fit the function can be directed toward the optimum. In this way they also serve as search points.

The organization also reduces calculations. The creation of both the model and the direction vectors is done a portion at a time. In

this way, the entire calculations are not done every iteration, but the information is updated each time it is needed. Also, the model is stored in such a way that the inverse of the curvature matrix is easily obtainable.

An important attribute of the new method is that, since it specifies only an organization, the actual implementation of several subtasks is not restricted. In this sense, the new method could be called a collection of methods. The analysis and testing of the new method includes a comparison of some of the alternatives. The range of alternatives prohibits an exhaustive analysis within the scope of this research. In addition, the best choice may depend on the problem. This means, however, that the method can be adapted to various types of problems.

Organization of Thesis

The remainder of this thesis describes the development of the new method. Chapter II is a review of previous methods for optimization. Also included is a brief summary of the Jacobi Method. Chapter III describes the method in detail. In Chapter IV, the mathematical analysis of the method is broken down into three divisions, the construction of the model, the determination of the eigenvectors, and the convergence of the optimization process. Chapter V describes the computer program used for testing. Chapter VI reports the results of testing the new method on various functions, along with comparisons to previous methods. Chapter VII gives conclusions and ideas for further research.

CHAPTER II

LITERATURE REVIEW

Introduction

The subject of optimization is well known. For a general background and bibliography see Kowalik and Osborne (1968), Box, Davies and Swann (1969), Polak (1971), Brent (1972), and Murray (1972). For comparisons of methods, see Fletcher (1965), Box (1966), and Himmelblau (1972).

The problem is to find values for a set of n variables

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix} , \qquad \text{(II.1)}$$

which minimize or maximize the scalar function f(x). Some methods consider only minimization. The problem of maximization of a function h(x) can be accomplished by minimizing

$$f(x) = -h(x) \quad . \qquad \text{(II.2)}$$

In practice, however, it is easy to arrange most algorithms to minimize or maximize on command. The test program used in this research includes this feature.

Many methods are designed to insure that the minimum of a quadratic function

$$f(x) = y_o + g_o^T (x - x_o) + (x - x_o)^T A_o (x - x_o) \qquad \text{(II.3)}$$

will be found in a finite number of steps (ignoring computational error). This property is often called "quadratic convergence." Quadratic convergence is also used to describe another property, the reduction of error to a multiple of the square of the previous error. To avoid confusion the term "finite convergence" is used for the first property.

The topic of this research is direct search optimization. Direct search methods are those which require only the value of the function (not the derivatives). Some other types of methods are sufficiently related to be reviewed in this chapter. Included are types which require values for the gradient or the matrix of second partial derivatives, called here the curvature matrix. Finally, due to its importance in the new method, the background of the Jacobi Method for finding eigenvalues is discussed.

Direct Search Optimization

Direct search methods can be categorized into methods which are based on direction vectors and those which are not. Direction vector methods include Rosenbrock (1960), Powell (1964), and Davies, Swann and Campey (DSC) reported by Swann (1964, 1969). Non-direction vector methods include the simplex method of Nelder and Mead (1965) and the

pattern search method of Hooke and Jeeves (1961). Although non-direction vector methods are sometimes more efficient on small problems, the direction vector methods have generally been found to be more efficient and more reliable in practical comparisons such as Box (1960), and Himmelblau (1972). To this author, Powell's method appears to be the best method for a wide range of search problems.

The Powell and DSC methods are based on moving to the optimum along a line in each direction in turn. Finite convergence is then assured if the direction vectors are conjugate with respect to the curvature matrix. Vectors s_i and s_j are conjugate with respect to A if

$$s_i^T A s_j = 0, \; i \neq j \quad . \tag{II.4}$$

The effect is that (for a quadratic function) movement in one direction is independent of movement in the other directions.

One problem with some versions of Powell's method is that the direction vectors can become linearly dependent (Zangwill, 1967; Brent, 1972). This slows or eliminates movement in some directions. Complete freedom of movement is allowed if the direction vectors are orthogonal. Vectors which are both orthogonal and conjugate with respect to a matrix are by definition the eigenvectors of the matrix. For this reason the eigenvectors of the curvature matrix are desirable as direction vectors.

Another problem with some methods is that their calculations depend on moving to the optimum in each direction. This involves at least as many directions as there are variables. Finding these optima can include a large number of trials which do not involve movement toward the overall goal. Fletcher (1965) points out the further disadvantage that the optimum along a line may not exist.

Stewart (1967) proposes using gradient methods when the gradient is not available by using an approximate gradient based on perturbations. Other authors advise against this approach (Swann, 1969; Brent, 1972). The approach has two disadvantages. First, it is difficult to choose a suitable step size since reducing the step length soon introduces round-off error and increasing the step increases the error of approximation. The errors which do result can be important since the gradient methods depend on the gradient for much of their information. Secondly, the steps used in finding the gradient may be wasted in terms of actual movement toward the goal. For further discussion and references, see Sargent and Sebastian (1972).

Other Types of Optimization

The situation associated with gradient methods yields to analysis somewhat more easily than that of the non-gradient types. The gradient provides very important information about the direction and distance to the optimum (assuming curvature information can be obtained) and indicates when the optimum has been found. Most methods determine a corrected direction vector based on the gradient and move to the optimum in that direction. These methods include the conjugate gradient method of Fletcher and Reeves (1964) and the variable metric or DFP method originally due to Davidon (1959) as simplified by Fletcher and Powell (1963). A generalized variable metric method given by Huang (1970) includes the conjugate gradient and DFP methods as special cases.

Huang and Levi (1970) show that, applied to a quadratic function, the conjugate gradient and DFP methods evaluate the same points. On non-quadratic functions the DFP method is more efficient (Huang and

Levi, 1970, Himmelblau, 1972, Sargent and Sebastian, 1972). This is to be expected since it retains more information and performs more calculation. The conjugate gradient method, however, has the advantages of simplicity and small storage requirements.

Most variable metric type methods base their operation on successive line searches. Rather than keeping a set of directions, however, the algorithm generates a new direction each iteration. The methods are usually designed to insure that, for a quadratic function, a sequence of n conjugate direction vectors is generated. Thus, the methods have finite convergence.

More recently, Fletcher (1970) describes another variable metric method. Based on testing by Himmelblau (1972), the method appears to be more efficient than other methods. This efficiency is in spite of the fact that the method does not have the property of finite convergence. One factor which does contribute to the efficiency is that the method does not depend on moving to the optimum along lines.

In some problems the curvature matrix can be evaluated or approximated. An important special case is minimization of a function consisting of a sum of squared terms. For a sum of squares, the curvature matrix can be approximated using the values and gradients of the individual terms. Methods which make use of the curvature matrix (or equivalent information) may require even fewer function evaluations. For a review and comparison of methods see Bard (1970, 1974), and Sargent and Sebastian (1972).

The Jacobi Method

For a real symmetric matrix A, the eigenvectors can be defined as

s_i, the columns of S, such that S is orthogonal and

$$D = S^T A S \tag{II.5}$$

is diagonal. Each d_{ii} is then the eigenvalue of A corresponding to s_i. An iterative method for finding the eigenvectors and eigenvalues was originally described by Jacobi (1846). For further background of the eigenvalue-eigenvector problem and the Jacobi method see Wilkinson (1965) and Hammerling (1970).

The Jacobi method begins with a matrix A^o, equal to the original matrix A, and a second matrix S^k, initially equal to the identity matrix. At each iteration, a super-diagonal element a^k_{ij} is chosen. A plane rotation matrix is used to transform A^k so that the chosen element is reduced to zero. The matrix A^o is symmetric, and the rotations preserve symmetry, so a^k_{ji} is always equal to a^k_{ij} and becomes zero also. During this process, the matrix S^k is used to record the rotations, maintaining the relation

$$A^k = (S^k)^T A S^k \quad . \tag{II.6}$$

Since the elements which have become zero are affected by later rotations, the matrix A^k does not become diagonal after each element has been chosen one time. However, under proper conditions the off-diagonal elements converge to zero. Therefore, S^k converges to the matrix of eigenvectors.

The original (and fastest) version of the Jacobi method chooses the various elements of A^k in order of decreasing magnitude. When incorporated in the new optimization method, only the current off-diagonal element will be known. Therefore, an arbitrary ordering must be used.

This corresponds to the "cyclic Jacobi method" (Forsythe and Henrici, 1960), also known as the "serial Jacobi method" (Wilkinson, 1965). In this case, a complete cycle, in which every super-diagonal element of A^k is used exactly once, is called a sweep.

The cyclic Jacobi method is as follows.

1. Set

$$k = 0 \quad ,$$

$$A^0 = A \quad , \tag{II.7}$$

and

$$S^0 = I \quad . \tag{II.8}$$

2. According to some ordering, choose each pair of indices,

$$(i, j) = (i_k, j_k),\ 1 \leq i < j \leq n \tag{II.9}$$

For each pair perform the following

a. Calculate the matrix U^k of the form:

$$\left.\begin{aligned} u_{ii} &= \cos\phi & u_{ij} &= \sin\phi \\ u_{ji} &= -\sin\phi & u_{jj} &= \cos\phi \\ u_{pp} &= 1 \text{ for all } p \neq i,\ p \neq j \\ \text{all other } u_{pq} &= 0 \end{aligned}\right\} \tag{II.10}$$

where

$$\phi = \tfrac{1}{2}\ \text{Arctan}\ \left(\frac{2a_{ij}^k}{a_{jj}^k - a_{ii}^k}\right) . \tag{II.11}$$

b. Set

$$A^{k+1} = (U^k)^T A^k U^k \tag{II.11}$$

and

$$S^{k+1} = S^k U^k \quad . \tag{II.13}$$

c. Replace k by k+1 .

3. Return to step 2, unless convergence is indicated

Without specifying the ordering, the convergence of the cyclic Jacobi method is difficult to analyze. Convergence is measured by

$$E^k = \sum_{p \neq q} (a_{pq}^k)^2 \quad . \tag{II.14}$$

For A symmetric, the same value can be found from

$$E^k = \sum_{p<q} (a_{pq}^k)^2 \quad . \tag{II.15}$$

Henrici (1958) and Schonhage (1961) prove that <u>if</u> the method does converge and if the matrix has distinct eigenvalues, then the error eventually converges to zero quadratically. It has not been proved that the general cyclic Jacobi method is convergent. (See, for example, Wilkinson, 1965.) In fact, Hansen (1963) shows a matrix for which a certain order fails to produce convergence, although the same paper gives results of testing several orderings on random matrices with no indication of failure.

As a result of the difficulty of analysis, almost all literature on the cyclic method uses the row ordering, or the column ordering. An example of each of these orderings is symbolized in Figure 1. When the row or column ordering is used, the method is known as the special cyclic (or special serial) Jacobi method. Hansen (1963) shows that the two orderings produce identical results at the end of each sweep (except for computational error).

	1	2	3	4	5
		6	7	8	9
			10	11	12
				13	14
					15

a. Row Ordering

	1	2	4	7	11
		3	5	8	12
			6	9	13
				10	14
					15

b. Column Ordering

Figure 1. Two Examples of Order of Choosing Super-Diagonal Elements

Forsythe and Hencici (1960) prove that the special cyclic Jacobi method as described here converges in the sense that

$$\lim_{k\to\infty} A^k = D \quad . \tag{II.16}$$

Wilkinson (1962) gives a proof of quadratic convergence with a better rate constant than Henrici (1958). On the other hand, Hansen (1963) presents arguments and test results which favor other orderings, such as a diagonal ordering symbolized in Figure 2. However, his counter-example to convergence can be generalized to this ordering, and for all orderings tried, the average time to converge to a standard accuracy varied only from 4.1 to 4.9 sweeps. Wilkinson (1965, p. 271) states that in practice five to six sweeps reduces the off-diagonal elements to zero to an accuracy of 10 to 15 places. Gregory (1953) gives test results which substantiate that claim. Thus, both analysis and experience indicate that one of the special cyclic orderings is preferable to other orderings.

	1	6	10	13	15
		2	7	11	14
			3	8	12
				4	9
					5

Figure 2. Example of Diagonal Ordering

CHAPTER III

PROPOSED METHOD

Introduction

This chapter describes the new method for unconstrained function optimization. The operation of the method is discussed in terms of the underlying processes. Various possibilities for implementation are considered and the algorithm used for testing is given. Some of the processes are analyzed in more detail in Chapter IV. The test program and results are given in Chapters V and VI.

The New Method

The new optimization method is based on a quadratic model,

$$u(x) = y_o + g^T(x - x_o) + \tfrac{1}{2}(x - x_o)^T A(x - x_o) \quad , \qquad \text{(III.1)}$$

to approximate the function $f(x)$ near x_o. The model parameters are the scalar y_o, the vector g, and the matrix A. A is referred to as the model curvature matrix. Taylor's theorem implies that

1. y_o approximates $f(x_o)$,
2. g approximates the gradient of $f(x)$ at x_o,
3. A approximates the matrix of second partial derivatives of $f(x)$ at x_o.

It is assumed that the function is well behaved, so the function curvature is symmetric. The model curvature matrix can always be assumed to be symmetric, since in the form S^TAS there is no way to distinguish between the contribution of $S_i a_{ij} S_j$ and $S_j a_{ji} S_i$.

In addition to the model, the new method uses a set of orthogonal direction vectors s_i, the columns of S. S^T is used as a linear transformation of the parameter space giving

$$z = S^T(x - x_o) \quad , \tag{III.2}$$

$$b = S^T g \quad , \tag{III.3}$$

$$C = S^T A S \quad . \tag{III.4}$$

Then S and C have the proper relation to A for the Jacobi method. Note that $x = x_o + Sz$. Since SS^T is the identity matrix, the model can be written

$$u(x) = y_o + g^T SS^T(x - x_o) + \tfrac{1}{2}(x - x_o)^T SS^T ASS^T(x - x_o) \tag{III.5}$$

or

$$u(x_o + Sz) = y_o + b^T z + \tfrac{1}{2} z^T C_z \quad . \tag{III.6}$$

Each iteration of the Jacobi method requires only the values of c_{ii}, c_{jj} and c_{ij}. If only z_i and z_j are non-zero, the model reduces to

$$u(x_o + z_i s_i + z_j s_j) = y_o + b_i z_i + b_j z_j + \tfrac{1}{2} c_{ii}(z_i)^2$$

$$+ \tfrac{1}{2} c_{jj}(z_j)^2 + \tfrac{1}{2} c_{ij} z_i z_j + \tfrac{1}{2} c_{ji} z_j z_i \tag{III.7}$$

Due to symmetry

$$\tfrac{1}{2}c_{ij}z_iz_j + \tfrac{1}{2}c_{ji}z_jz_i = c_{ij}z_iz_j \quad , \tag{III.8}$$

so

$$u\, x_o + z_is_i + z_js_j) = y_o + b_iz_i + b_jz_j$$

$$+ \tfrac{1}{2}c_{ii}(z_i)^2 + \tfrac{1}{2}c_{jj}(z_j)^2 + c_{ij}z_iz_j \quad . \tag{III.9}$$

This submodel can be fit to the actual function in the plane of $x_o + z_is_i + z_js_j$, resulting in c_{ii}, c_{jj} and c_{ij} as required.

The basic method is as follows.

1. Assume some initial base location, x_o, and an initial set of orthogonal direction vectors, S. For a two-dimensional example, see Figure 3.
2. According to some ordering, choose each possible pair

$$(i, j) = (i_k, j_k),\ 1 \leq i < j \leq n \quad . \tag{III.10}$$

For each pair, operate in the plane of the two direction vectors s_i and s_j as follows.

a. Sample the function in the plane near the base location, x_o, and calculate b_i, b_j, c_{ii}, c_{jj} and c_{ij} in the bivariate quadratic submodel of the function, Eq. III.9, to make the model fit the function at the sample points. See Figure 4.

b. Calculate the plane rotation matrix R of the form of U^k in Equation II.10.

c. Replace b by R^TB, C by R^TCR and S by SR. See Figure 5.

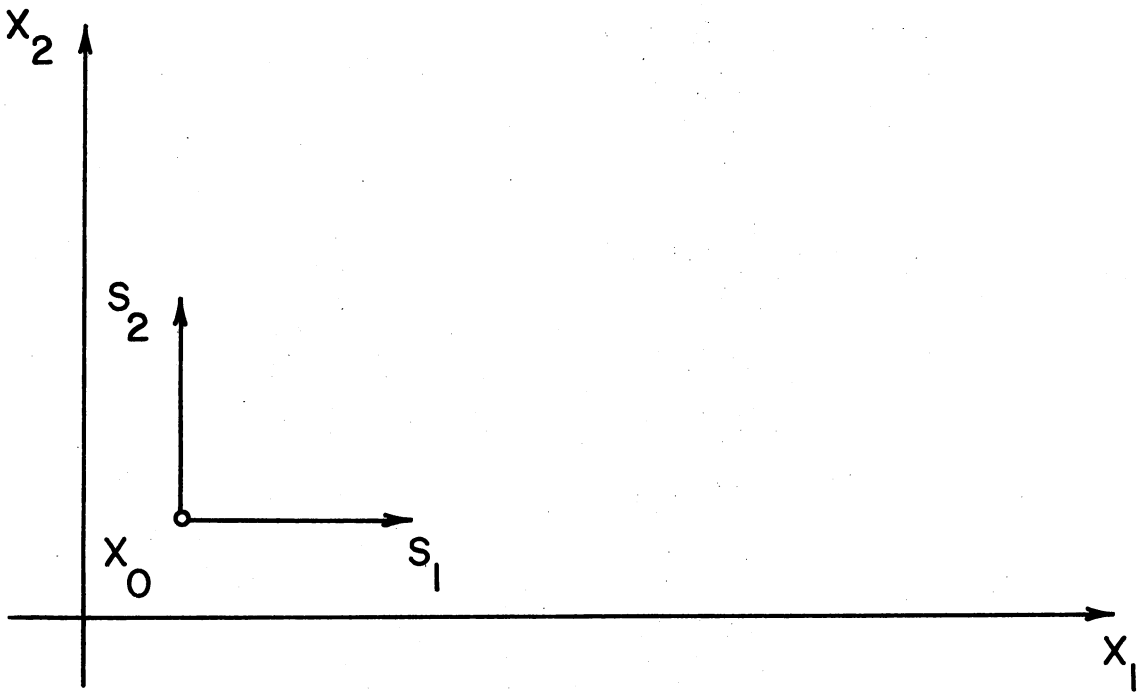

Figure 3. Base Location and Direction Vectors

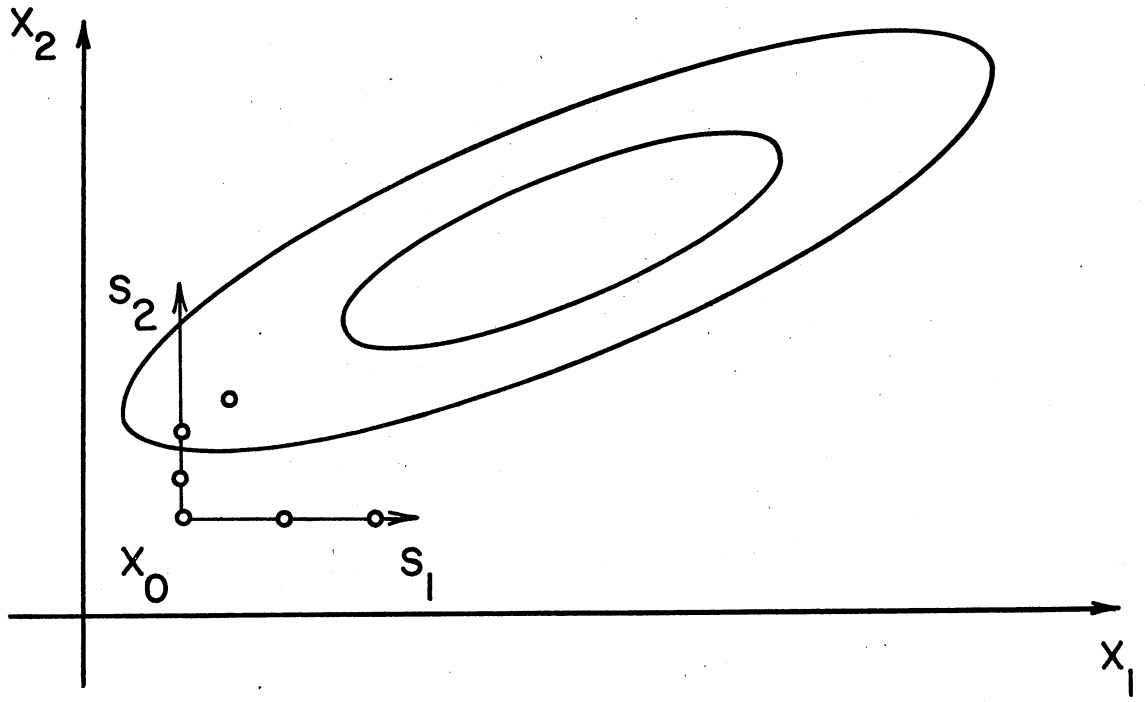

Figure 4. Sample Points and Model (Represented by Isometric Curves)

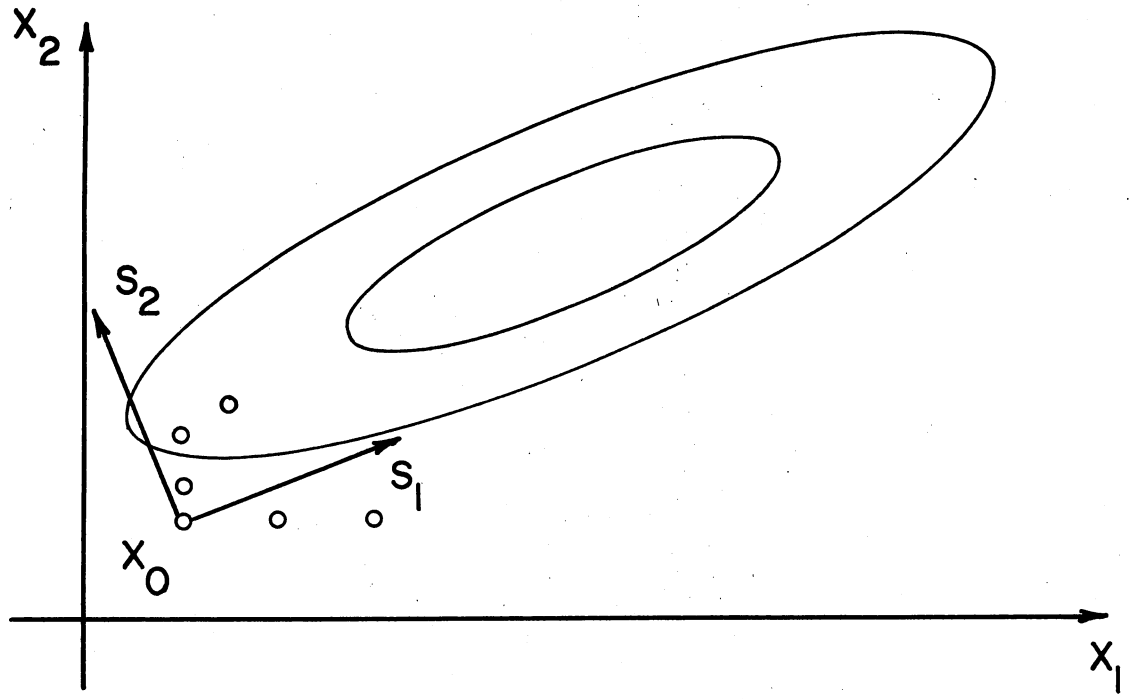

Figure 5. Rotated Direction Vectors

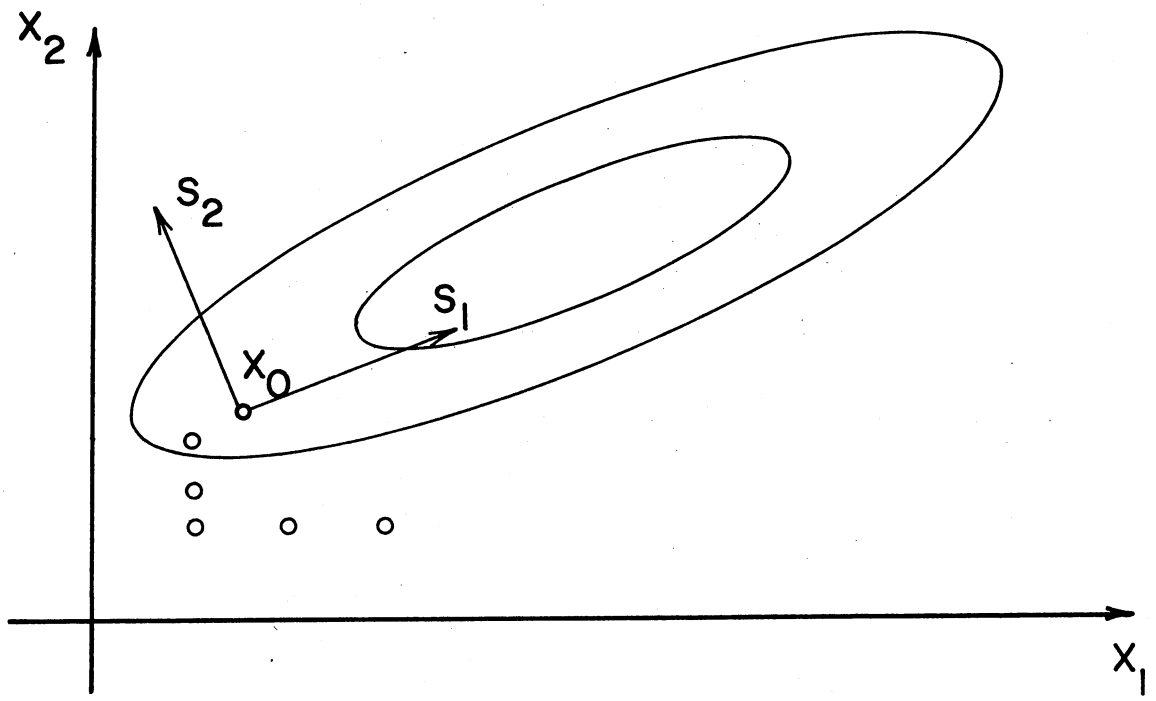

Figure 6. New Base Location

d. Using the direction vectors, search in the plane for a location to improve the value of the function and move the base location, x_o, to the best point found. See Figure 6.

e. Replace k by k + 1.

3. Return to step 2, unless convergence to the optimum is indicated.

As described, the algorithm involves three matrices. However, each time an off-diagonal element, c_{ij}, is found it is immediately changed to zero. Therefore, only the diagonal of C needs to be stored. The rotation matrix R is a special form which requires only two values, the sine and cosine. Thus, only one matrix, S, is required.

The amount of calculation also appears large. Each sweep involves on the order of n^2 rotations and each rotation involves a number of vector and matrix operations. However, the rotation (step 2c) involves only two elements of b, two diagonal elements of C and two columns of S. Thus, the number of calculations is on the order of n per rotation, i.e., n per line search. The total amount of computation for a complete sweep is on the order of n^3 but involves on the order of n^2 line searches.

Operation of the Method

Three simultaneous processes are involved in the method, each updating a separate set of information.

1. A curve fitting process is creating a model of the function in the region of the base location.
2. An eigenvector process is finding the eigenvectors of the model curvature matrix.

3. A search process is moving the base location.

For each pair of direction vectors, each process operates in turn, updating a portion of its information. Each process, however, uses information from the other processes.

The interaction of the three processes is shown by the diagram, Figure 7. For a given pair of vectors, the curve fitting process creates the submodel for the plane (path a) using the current direction vectors (path e) and the base location (path f) to define the plane. The eigenvector process rotates the two direction vectors (path b) based on the second derivative information of the model (path g). The search process moves the base location (path c) based on the calculated optimum from the model (path i) along direction vectors (path j).

As a result of the interaction, the convergence of each process is affected by the operation of the other processes. Although this makes analysis of convergence difficult, the frequent correction of the model and direction vectors is a great advantage to the search process.

Implementation of the Method

The basic method allows flexibility in accomplishing the following operations.

1. The order of choosing pairs of direction vectors in step 2,
2. The arrangement of sample points in step 2a,
3. The strategy for searching to improve the base location in step 2d,
4. The criterion for convergence in step 3.

Thus, the basic method could be called a collection of methods. The choice of an algorithm for each operation results in a specific method.

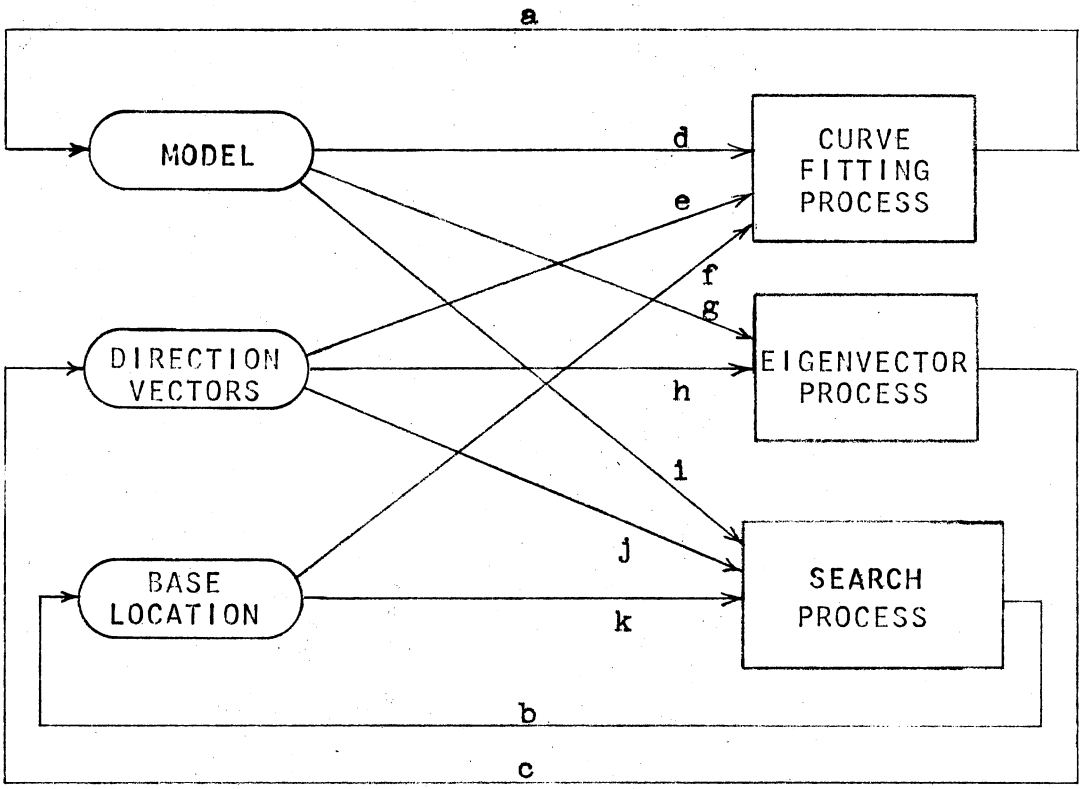

Figure 7. System Diagram of the Information and Processes in the New Method

Some of the possible algorithms are given in the following discussion of the four operations.

The order of choosing pairs affects two things:

1. The pattern of searching in the various directions,
2. The convergence of the Jacobi method.

The pattern of searches suggests an ordering to try to isolate one occurrence of each index. (The indices correspond to direction vectors.) For n equal to six this could be

$$(i_k,j_k) = \{(1,2),(3,4),(5,6),(2,3),(4,5),(6,1),(1,3),(2,4),$$
$$(3,5),(4,6),(5,1),(6,2),(1,4),(2,5),(3,6)\} . \qquad \text{(III.11)}$$

Another possibility is to favor two sequential occurrences of each index, such as

$$(i_k,j_k) = \{(1,2),(2,3),(3,4),(4,5),(5,6),(6,1),(1,3),(3,5),$$
$$(5,1),(2,4),(4,6),(6,2),(1,4),(2,5),(3,6)\} \qquad \text{(III.12)}$$

These orderings are both organized along diagonals of the curvature matrix as symbolized in Figure 8. The convergence of the Jacobi method, however, is improved by the orderings along rows or columns symbolized in Figure 1. The effect of ordering on the Jacobi method is discussed in Chapters II and IV. Two orderings are tested in Chapter VI.

The sample points used for fitting the model could be arranged in a fixed pattern. The calculations could then be further simplified. On the other hand, the sample points used for fitting the model could also serve as search points. The search process (step 2d) could then consist solely of moving the base location to the best point found.

	1	7	13	11	6
		4	8	14	12
			2	9	15
				5	10
					3

a. Ordering to isolate one occurence of each index

	1	7	13	9	6
		2	10	14	12
			3	8	15
				4	11
					5

b. Ordering to favor two sequential occurences of each index

Figure 8. Orderings Based on the Pattern of Searches

This combination of processes is limited when the desired sample point with regard to the search produces a step size which is too small for the calculation of the model parameters.

A number of search strategies could be used in the new method. Possibilities include line searches along both direction vectors or a line search in the direction of the optimum. The algorithm used for testing uses the search points for fitting the model as just described. Additional points are added to the curve fitting points to insure, if possible, that the optimum is bracketed and a better point is found. At the end of a complete sweep an additional sample is taken at the optimum of the entire model. During the process the base point is moved to the best point found when the move will not disturb the calculations.

The choice of a convergence criterion is difficult for all optimization methods. It is commonly considered to be a separate problem from the method. For comparison to other methods, a standard termination scheme, such as Himmelblau (1972), could be applied. In the testing termination was not used for comparison. To prevent useless execution, the program terminates when a complete sweep produces no improvement.

The resulting algorithm used in the trials is as follows.

1. Set some initial x_o, and set S to the identity matrix.
2. According to some ordering, choose each possible pair of indices.

$$(i, j) = (i_k, j_k),\ 1 \leq i < j \leq n\ . \qquad \text{(III.13)}$$

For each pair, perform the following.

a. Sample the function at the predicted optimum in direction s_i.

b. Calculate a corrected value for b_i.

c. Sample at the corrected optimum in direction s_i.

d. Calculate b_i and c_{ii}.

e. Move the base location, x_o, to the best point found thus far.

f. Similarly, sample twice in direction s_j, resulting in b_j and c_{jj}.

g. Sample at the predicted optimum in the plane of s_i and s_j.

h. Calculate c_{ij}.

i. Move to the best point found.

j. Calculate the rotation matrix R.

k. Replace b by $R^T b$, C by $R^T CR$, and S by SR.

l. Replace k by k+1.

3. Sample the function at the predicted multivariate optimum and move the base location if better.

4. Return to step 2, unless no progress has been made during the sweep.

The optima are based on the previous model limited to reasonable values.

Summary

A new method for optimization is described in this chapter, based on the Jacobi method for finding eigenvectors. In the new method, a quadratic model of the function is created, and search directions which approximate the eigenvectors of the curvature matrix are used. The

nature of the Jacobi method allows the construction of the model, the determination of the search directions, and the search itself to proceed simultaneously. The basic method does not specify the procedure for accomplishing some operations, so there are many ways to implement the method.

CHAPTER IV

THEORETICAL ANALYSIS

Introduction

This chapter collects the analysis of processes related to the new optimization method. First, the effects of scaling the problem variables are discussed. Next, the curve fitting process is examined and restrictions on the step size are developed. The effect of ordering on the Jacobi method is then analyzed. A modified ordering is given along with a proof regarding convergence. Finally, the convergence of the base location to the optimum is considered, and the rate of convergence for the case of a quadratic function is found.

Scaling

For a given function, a change of scale is the replacement of any variable x_i by a constant multiple px_i. If the values of the variable are divided by the same constant, then the function values will remain the same. Thus, a change of scale does not change the inherent properties of the function. An example of change of scale is a change in the units of measure, say from meters to kilometers. If the function is rewritten in kilometers and all "data" values are converted to kilometers the function will behave the same. This may not affect all variables, because various variables may have entirely different dimensions (e.g. time).

The appearance of a function as an optimization problem is changed by scaling. To illustrate, consider minimizing the function

$$f(x) = .01 \frac{x_1^2}{2} + \frac{x_2^2}{2} \quad . \tag{IV.1}$$

The contour diagram in Figure 9 shows that the function appears to be a long valley. If, however, x is "de-scaled" with

$$\tilde{x}_1 = .1\, x_1 \ , \tag{IV.2}$$

$$\tilde{x}_2 = x_2 \ , \tag{IV.3}$$

the function becomes

$$f(\tilde{x}) = \frac{\tilde{x}_1^2}{2} + \frac{\tilde{x}_2^2}{2} \ , \tag{IV.4}$$

which has circular contours. The de-scaling of IV.2 and IV.3 is given by

$$\tilde{x}_i = \sqrt{a_{ii}}\ x_i \ , \tag{IV.5}$$

where the a_{ii} are the diagonal elements of the curvature matrix. The curvature matrix of the original function

$$A = \begin{pmatrix} .01 & 0 \\ 0 & 1 \end{pmatrix} \ , \tag{IV.6}$$

becomes, for the de-scaled function,

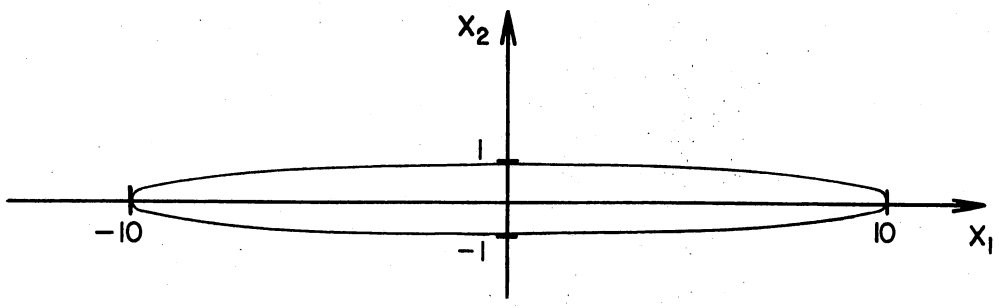

Figure 9. Contour Diagram f(x) = 0.5 for Equation IV.1

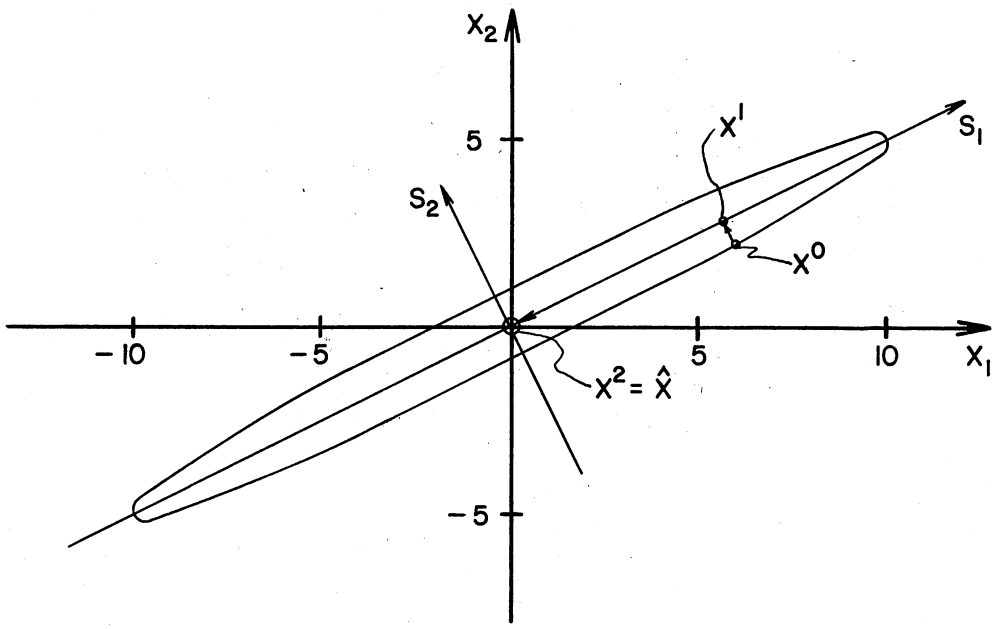

Figure 10. f(x) = 0.5 for Equation IV.8

$$\tilde{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} . \qquad \text{(IV.7)}$$

The de-scaling of Equation IV.5 always results in a curvature matrix with ones on the diagonal. The valley is controlled entirely by the scaling, and there is no absolute reference for the scale. Therefore, it can be argued that it is meaningless to talk about a valley in this problem.

On the other hand, the function

$$f(x) = .01 \frac{(0.5\ x_1 - x_2)^2}{2} + \frac{(0.5\ x_1 + x_2)^2}{2} , \qquad \text{(IV.8)}$$

diagrammed in Figure 10, has a valley which may be modified, but not completely removed by de-scaling x. (To remove it completely requires a general linear transformation or a rotation followed by de-scaling.) When de-scaled with

$$\tilde{x}_1 = 0.5\ x_1 , \qquad \text{(IV.9)}$$

$$\tilde{x}_2 = x_2 , \qquad \text{(IV.10)}$$

$$f(\tilde{x}) = .01 \frac{(\tilde{x}_1 - \tilde{x}_2)^2}{2} + \frac{(\tilde{x}_1 + \tilde{x}_2)^2}{2} . \qquad \text{(IV.11)}$$

The contours shown in Figure 11 illustrate a remaining valley. The "narrowness" of the valley, i.e., the ratio of the eigenvalues (curvature), is reduced, however. Non-quadratic functions can also create

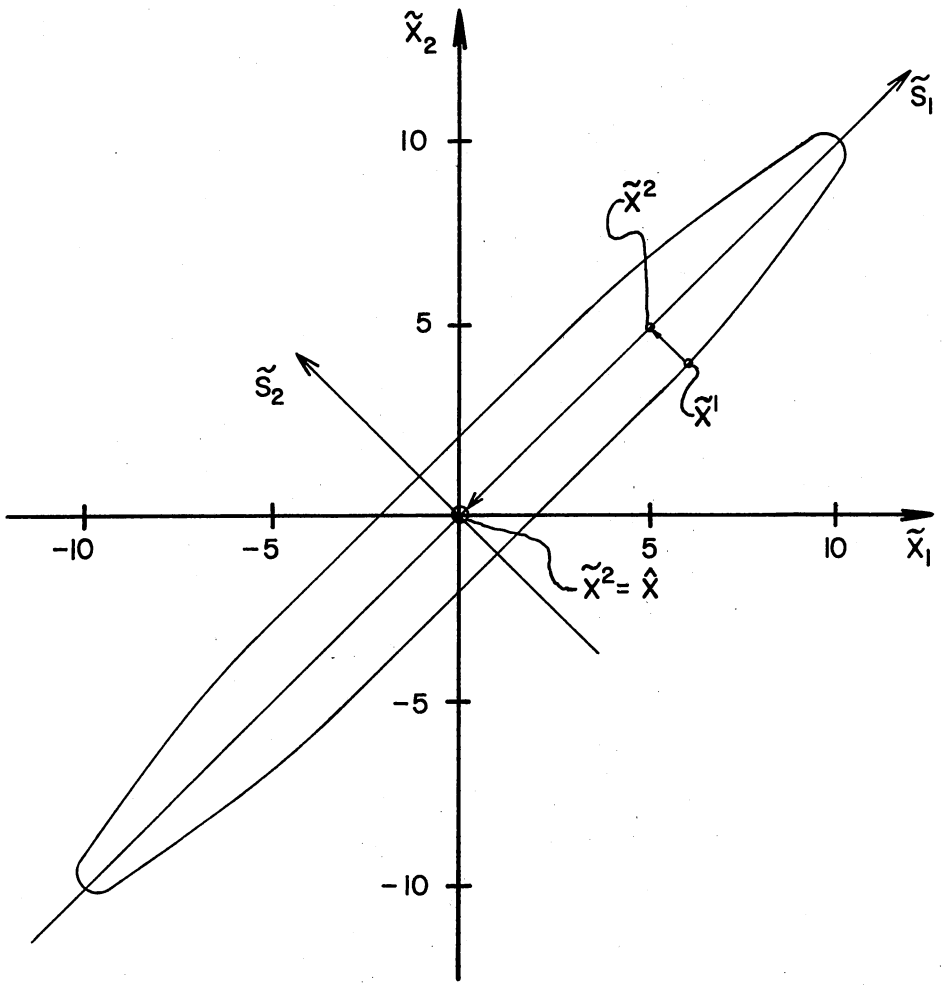

Figure 11. f(x) = 0.5 for Equation IV.11

conditions where a valley cannot be removed by scaling or even by a general linear transformation.

The conclusion is that "poor" scaling can increase the apparent difficulty of a problem. One solution is to reformulate the problem to improve scaling. However, the function is not always sufficiently well known to de-scale manually. Also, for non-linear problems the scale may change with location. Therefore, it is desirable that optimization methods be insensitive to poor scaling. When properly programmed, some methods are scale-invariant, that is, when the scale of a problem is changed (including the initial location and step sizes) the method samples the scaled function at exactly the points corresponding to those used before scaling.

The new method is not scale-invariant due to the properties of eigenvectors. When a function such as IV.8 is scaled to IV.11 the eigenvectors of the scaled problem are not equal to the scaled eigenvectors of the original problem. To illustrate the effect on the method, assume in each case the method starts at an initial location, x_o, finds the eigenvectors exactly, and then performs a linear search in direction s_2 and then s_1. Referring to Figure 10, the movement for the original function is to point x^1 and then to $s^2 = x$, the optimum. In the scaled problem of Figure 11, the movement is to $\tilde{x}^1$ and then to $\tilde{x}^2 = \hat{x}$. Since direction s_2 does not scale to $\tilde{s}_2$ the point $\tilde{x}^1$ does not correspond to x^1. Therefore, the method applied to a scaled problem does not sample the scaled version of the original points.

In the new method, the A matrix can be found from the model, so it would seem easy to automatically de-scale the variables. Several problems arise, however. The greatest problem is that if the old

direction vectors actually are the correct eigenvectors, and the scale is changed, then the direction vectors are no longer the eigenvectors The calculation of the de-scaled eigenvectors requires the complete calculation of the eigenvectors of a matrix. Even calculation of new curvature values (c_{ii}) along the old direction vectors is a lengthy calculation (n^3 operations). Fortunately, the method can be made "scale-invariant" along the direction vectors by basing every step size calculation on scale-invariant properties of the model or on the previous step size. (This was done in the test program.) For this reason, once the direction vectors are aligned with the "valley" the functions of Figures 10 and 11 both appear as circular contours. Thus, the new method should work "well," though differently, when the scale is changed.

Another effect occurs when the scale is changed. If the function of IV.11 is further scaled with

$$\tilde{\tilde{x}}_1 = \tilde{x}_1 \quad , \tag{IV.12}$$

$$\tilde{\tilde{x}}_2 = 2\tilde{x}_2 \quad , \tag{IV.13}$$

the contours becomes those of Figure 12. The rotation of the direction vectors always chooses the angle less than 45^o, so direction s_2 becomes "down the valley" and s_1 becomes "across the valley." For cases where there is some reason to distinguish between "across the valley" and "down the valley" (e.g., nonlinear problems), the meaning of s_1 and s_2 are reversed. If the direction vectors are chosen in order of curvature, c_{ii}, then the first (last) direction vector will represent, say, the greatest (least) curvature which corresponds to "across the valley"

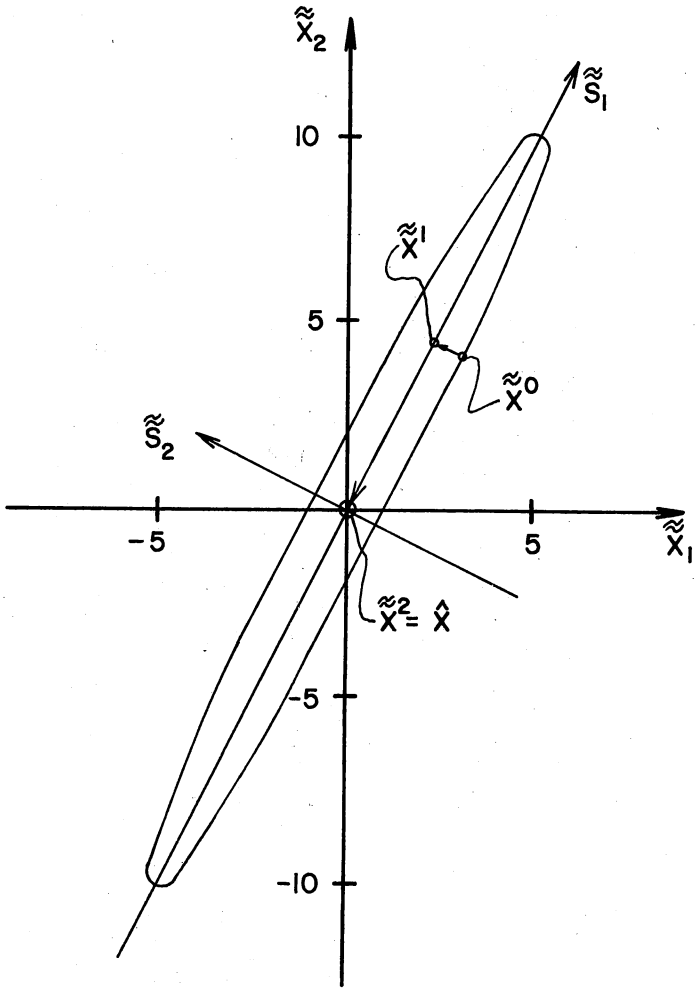

Figure 12. f(x) = 0.5 for Function of Figure 11 with Scale Changes of Equations IV.12 and IV.13

("down the valley"). Thus, the "pattern" of searching will remain similar even though the scale is changed. Sorting the diagonal of the C matrix, discussed later in this chapter, causes the direction vectors to be chosen in a specified order according to curvature. In that case the new method is somewhat insensitive to change of scale.

Fitting the Model

In the basic method given in Chapter III, one task is to sample the function f(x) at a sufficient number of points to fit the model expressed by Equation III.9,

$$u(x_o + z_i s_i + z_j s_j) = y_o + b_i z_i + b_j z_j$$

$$+ \tfrac{1}{2} c_{ii} z_i^2 + \tfrac{1}{2} z_{jj} c_j^2 + c_{ij} z_i z_j \ . \qquad \text{(IV.14)}$$

Normally x_o will be changed only to a point which has already been evaluated. As a result,

$$y_o = u(x_o) = f(x_o) \qquad \text{(IV.15)}$$

is assumed to be known. Five additional samples are usually sufficient to determine the remaining five unknowns, b_i, b_j, c_{ii}, c_{jj}, and c_{ij}.

The organization of the sample points can isolate the effect of z_i and z_j in the following way. If z_j is zero, the model becomes

$$u(x_o + z_i s_i) = y_o + b_i z_i + \tfrac{1}{2} c_{ii} z_i^2 \ . \qquad \text{(IV.16)}$$

Thus, two points on the line $x_o + z_i s_i$ (parametric in z_i) are usually sufficient to find b_i and c_{ii}. Similarly, two points on the line

$x_o + z_j s_j$ give b_j and c_{jj}. For this reason and since optimization along a line is of interest, the model along a line is considered next.

Univariate Model Calculations

For this section, the subscript i is temporarily eliminated giving the model

$$u(x_o + zs) = y_o + bz + \tfrac{1}{2}cz^2 \qquad \text{(IV.17)}$$

for the function $f(x_o + zs)$, where s becomes a single direction vector, and z, b and c are scalars. As noted before, y_o is assumed to be available. The task is to chose two additional points, z_1 and z_2, sample the function at the points, giving

$$y_1 = f(x_o + z_1 s) \ , \qquad \text{(IV.18)}$$

$$y_2 = f(x_o + z_2 s) \ , \qquad \text{(IV.18)}$$

and find b and c which satisfy the relations

$$u(x_o + z_1 s) = y_o + bz_1 + \tfrac{1}{2}cz_1^2 = y_1 \ , \qquad \text{(IV.20)}$$

$$u(x_o + z_2 s) = y_o + bz_2 + \tfrac{1}{2}cz_2^2 = y_2 \ . \qquad \text{(IV.21)}$$

One important use of the model is to give an improved estimate $\hat{z}$ of the optimum along the line by solving

$$\frac{du}{dz} = b + cz = 0 \ , \qquad \text{(IV.22)}$$

Resulting in

$$\hat{z} = -b/c \quad . \tag{IV.23}$$

The direct solution of the original Equations IV.20 and IV.21 gives

$$b = \frac{z_1^2 (y_2 - y_o) - z_2^2 (y_1 - y_o)}{z_1 z_2 (z_2 - z_1)} \quad , \tag{IV.24}$$

$$c = 2 \left[\frac{z_1 (y_2 - y_o) - z_2 (y_1 - y_o)}{z_1 z_2 (z_2 - z_1)} \right] \quad . \tag{IV.25}$$

When using a computer to evaluate b and c, roundoff (or truncation) errors can become significant. As a result the order of calculation is important. The algorithm used in the testing is as follows:

$$d_1 = y_1 - y_o \quad , \tag{IV.26}$$

$$d_2 = y_2 - y_o \quad , \tag{IV.27}$$

$$b_1 = d_1/z_1 \quad , \tag{IV.28}$$

$$b_2 = d_2/z_2 \quad , \tag{IV.29}$$

$$c = 2(b_2 - b_1)/(z_2 - z_1) \quad , \tag{IV.30}$$

$$b = b_2 - cz_2/2 \quad , \tag{IV.31}$$

$$\hat{z} = -b/c \quad . \tag{IV.32}$$

Bivariate Model Calculations

Fitting the univariate model in directions s_i and s_j gives b_i, b_j,

c_{ii} and c_{jj}, leaving c_{ij} to be found. At least one additional point with both z_i and z_j nonzero is required. If the function is sampled giving

$$y_5 = f(x_o + z_i s_i + z_j s_j) \quad , \tag{IV.33}$$

then c_{ij} can be determined from the relation

$$u(x_o + z_i s_i + z_j s_j) = y_5 \quad . \tag{IV.34}$$

The direct solution is

$$c_{ij} = \frac{y_5 - (y_o + b_i z_i + b_j z_j + \tfrac{1}{2}c_{ii} z_i^2 + \tfrac{1}{2}c_{jj} z_j^2)}{z_i z_j} \quad . \tag{IV.35}$$

The algorithm used in the testing is

$$d_5 = y_5 - y_o \quad , \tag{IV.36}$$

$$c_{ij} = \frac{d_5 - (b_i + c_{ii} z_i/2) z_i - (b_j + c_{jj} z_j/2) z_j}{z_i z_j} \quad . \tag{IV.37}$$

If Equation IV.37 does not give sufficient accuracy, the value of c_{ij} can be found more directly from a univariate model along a line

$$x = x_o + z(u_i s_i + u_j s_j) \quad , \tag{IV.38}$$

where u_i and j_u are arbitrary nonzero constants. For convenience assume

$$u_i^2 + u_j^2 = 1 \quad . \tag{IV.39}$$

The resulting univariate model is denoted

$$u(x_o + z(u_i s_i + u_j s_j) = y_o + b_5 z + \tfrac{1}{2} c_5 z^2 \quad . \qquad \text{(IV.40)}$$

Equation IV.14 gives the corresponding form,

$$u(x_o + z u_i s_i + z u_j s_j) = y_o + b_i u_i z + b_j u_j z$$

$$+ \tfrac{1}{2} c_{ii} u_i^2 z^2 + \tfrac{1}{2} c_{jj} u_j^2 z^2 + c_{ij} u_i u_j z^2 \quad . \qquad \text{(IV.41)}$$

Equating the coefficients of the z^2 terms in IV.40 and IV.41,

$$c_{ij} = \frac{c_5 - c_{ii} u_i^2 - c_{jj} u_j^2}{2 u_i u_j} \quad . \qquad \text{(IV.42)}$$

The extra information provided by b_5 (because an extra point has been sampled) could then be used to test the accuracy and validity of the quadratic model by comparison to $(u_i b_i + u_j b_j)$. This option was not tested.

Step Size Restrictions

The calculations of both the univariate and bivariate model involve differences between function values at two sample points. As the step size becomes small the difference in function values becomes small. The errors in the function values then appear large relative to the difference. The relative error is then passed on to subsequent operations. As a result, a lower limit must be placed on the step size.

Let e(u) represent the absolute error in u. In moder floating point computers the error due to truncation is bounded by

$$e(u) \leq \varepsilon |u| \quad , \qquad \text{(IV.43)}$$

where ε is the machine precision. For example

$$\varepsilon = 10^{-6} \tag{IV.44}$$

for 7 digit storage.

The errors in the y values which can be estimated are due to truncation in storing x and y. Using a first order approximation to the function near x,

$$f(x + E(x)) = f(x) + g(x)^T E(x) \quad , \tag{IV.45}$$

where g(x) is the gradient of f at x. The error is thus

$$f(x + E(x)) - f(x) = g(x)^T E(x) \quad . \tag{IV.46}$$

Using IV.43 for each element of x,

$$|f(x + E(x)) - f(x)| \leq \varepsilon |g(x)|^T |x| \quad , \tag{IV.47}$$

where the absolute value of the vectors is performed on each element. The bound on the error in y due to both x and y is then approximated by

$$E(y) \leq \varepsilon \ |y| + \varepsilon |g(x)|^T |x| \quad . \tag{IV.48}$$

The errors in all the y values are estimated by the error in y_o, with

$$E(y_o) \leq \varepsilon |y_o| + \varepsilon |g(x_o)|^T |x| \quad . \tag{IV.49}$$

The gradient $g(x_o)$ can then be estimated from the model using III.3.

Next the value of d_1 is estimated under two conditions for $\hat{z}$. First, if

$$\hat{z} = -b/c \tag{IV.50}$$

is sufficiently large, then it can be used for z_1. In that case.

$$b = -cz_1 \quad , \tag{IV.51}$$

so IV.20 becomes

$$y_1 = y_o - cz_1^2 + \tfrac{1}{2}cz_1^2 - \tfrac{1}{2}cz_1^2 \quad , \tag{IV.52}$$

and IV.26 gives

$$d_1 = y_1 - y_o = -\tfrac{1}{2}cz_1^2 \ . \tag{IV.53}$$

On the other hand, if $\hat{z}$ becomes small, then z_1 must be limited. Assuming

$$|\hat{z}| \ll |z_1| \quad , \tag{IV.54}$$

then

$$|c\hat{z}| \ll |cz_1| \quad . \tag{IV.55}$$

Using IV.50 and IV.55

$$|b| = |c\hat{z}| \ll |cz_1| \quad , \tag{IV.56}$$

so

$$|bz_1| \ll |cz_1^2| \quad . \tag{IV.57}$$

Then the second term of IV.20 can be neglected, giving

$$y_1 \tilde{=} y_o + \tfrac{1}{2}cz_1^2 \tag{IV.58}$$

and

$$d_1 = y_1 - y_o \tilde{=} \tfrac{1}{2}cz_1^2 \quad . \tag{IV.59}$$

In either case, $\tfrac{1}{2}cz_1^2$ is a good estimate for d_1. To keep the relative error small, it is desired to insure

$$\frac{|E(d_1)|}{|d_1|} < t \quad , \tag{IV.60}$$

where t is some large number. Thus the limit on d_1 is

$$|d_1| \leq t|E(d_1)| \quad . \tag{IV.61}$$

Any error in y_1 or y_o causes an identical error in the magnitude of d_1 so the error in d_1 due to the errors in the y is bounded by

$$|E(d_i)| \leq |E(y_1)| + |E(y_o)| \quad . \tag{IV.62}$$

Using the error estimate of IV.49 for both $E(y_1)$ and $E(y_o)$

$$|E(d_1)| \leq 2E(y_o) \leq 2(\varepsilon|y_o| + \varepsilon|g|^T |x_o|) \quad . \tag{IV.63}$$

Therefore, if $|d_1|$ is limited by

$$|d_1| \geq 2t(\varepsilon|y_o| + \varepsilon|g|^T |x_o|) \geq t\, E(d_1) \quad , \tag{IV.64}$$

then IV.61 is ensured. Using $\tfrac{1}{2}cz_1^2$ to estimate d_1 the restriction becomes

$$|\tfrac{1}{2}cz_1^2| \geq 2t(\varepsilon|y_o| + \varepsilon|g|^T|x_o|) \quad . \tag{IV.65}$$

Solving for z_1 gives

$$|z_1| \geq \sqrt{\frac{4t\varepsilon|y_o| + 4t\varepsilon|g|^T|x_o|}{|c|}} \quad . \tag{IV.66}$$

The value of t is arbitrary so it can include 4ε by letting

$$t' = 4t\varepsilon \quad . \tag{IV.67}$$

In the program used for testing, two parameters t_y and t_x are used for the errors due to storing y and x respectively. The limit then becomes

$$|z_1| \geq \sqrt{\frac{t_y|y_o| + t_x|g|^T|x_o|}{|c|}} \quad . \tag{IV.68}$$

To maintain sufficient distance between all three sample locations the same limit is applied to z_2 and the difference $z_2 - z_1$.

To maintain accuracy in the bivariate fit, the steps in the two directions must be of similar magnitude. The comparison of step sizes in the two directions, however, must account for possible differences in scaling. This can be done by comparing the change in the function value caused by the steps. Again using $\frac{1}{2}cz^2$ to estimate the change $y - y_o$, the ratio of the changes for the two directions is limited by the relation

$$t_c \leq \frac{|c_{ii}z_i^2|}{|c_{jj}z_j^2|} \leq \frac{1}{t_c} \tag{IV.69}$$

with $t_c < 1$. The lower limit gives

$$|c_{ii}z_i^2| \geq t_c\,|c_{jj}z_j^2| \quad . \tag{IV.70}$$

Solving for z_i,

$$|z_i| \geq \sqrt{\frac{t_c|c_{jj}z_j^2|}{|c_{ii}|}} \quad . \tag{IV.71}$$

Similarly, the upper limit of IV.69 gives

$$|c_{jj}z_j^2| \geq t_c|c_{ii}z_i^2| \quad , \tag{IV.72}$$

which becomes

$$|z_j| \geq \sqrt{\frac{t_c|c_{ii}z_i^2|}{|c_{jj}|}} \quad . \tag{IV.73}$$

All step sizes in a given direction should be of similar magnitude, so all limits should be computed before the linear fitting. To keep the restriction on a given z_i from changing due to pairing with various z_j, the worst case

$$|z_i| \geq \sqrt{\frac{t_c \max\limits_{1\leq p\leq n} |c_{pp}z_p^2|}{|c_{ii}|}} \quad , \tag{IV.74}$$

should be used every iteration. Finally, the limits of Equations IV.68 and IV.74 ca be combined into

$$|z_i| \geq \sqrt{\frac{y_L}{|c_{ii}|}} \quad , \tag{IV.75}$$

where

$$y_L = t_y|y_o| + t_x|g|^T|x_o| + t_c \max_{1\le p\le n} |c_{pp}z_p^2| \quad . \qquad \text{(IV.76)}$$

The same value of y_L can then be used for the corresponding limit on z_j. Before fitting the univariate models, the test program calculates y_L based on the previous model. The limit of IV.75 is calculated as soon as a reliable estimate for c_{ii} is available. The limit is then enforced on the step sizes z_1 and z_2 and the difference $z_2 - z_1$. Whenever the optimum of the model produces a step size $\hat{z}$ which does not satisfy the limit, the step size for the sample is enlarged to equal the limiting value.

The Effect of Ordering on the Jacobi Method

This section uses the notation of Chapter II for the Jacobi method. In particular, the current matrix is

$$A^k = (S^k)^T A S^k \quad . \qquad \text{(IV.77)}$$

The convergence of A^k to a diagonal matrix is measured by the error at iteration k (beginning at zero),

$$E^k = \sum_{p\neq q} (a_{pq}^k)^2 \quad . \qquad \text{(IV.78)}$$

The object is to determine the ordering which will mini ize the error at the end of the sweep. A sweep involves N pairs of indices (p,q) where

$$N = \tfrac{1}{2}\, n(n-1) \quad . \qquad \text{(IV.79)}$$

Assume an arbitrary cyclic ordering and consider one sweep. Let k_o be

the first iteration of the sweep and let

$$I(p,q) = k \quad . \tag{IV.80}$$

describe the first iteration on or after k_o which rotates a_{pq}, that is,

$$(i_k, j_k) = (p, q) \tag{IV.81}$$

Each rotation causes the chosen element to become zero, but later rotations change the value. To investigate the simplest occurrence of this effect, define a triplet to be three elements, a_{pq}, a_{pr}, and a_{qr}, where p, q, and r are distinct indices (not necessarily ordered) in the interval (1, n). The position of the elements of a typical triplet is shown in Figure 13. Actually, if the indices are not ordered, some of the elements will not be super-diagonal. However, A is symmetric, and the rotation affects, for example, a_{pq} and a_{qp} identically. Therefore, all references to a_{pq} apply to either a_{pq} or a_{qp}. Let the interations which rotate the three elements be

$$k_1 = I(p, q) \quad , \tag{IV.82}$$

$$k_2 = I(p, r) \quad , \tag{IV.83}$$

$$k_3 = I(q, r) \quad , \tag{IV.84}$$

all within $(k_o, k_o + N-1)$. Assume

$$k_1 < k_2 < k_3 \quad , \tag{IV.85}$$

that is, the elements are chosen in the order a_{pq}, a_{pr}, a_{qr}.

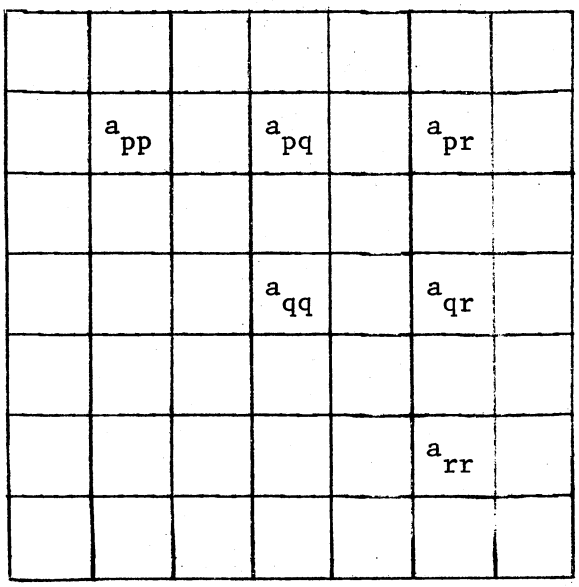

Figure 13. An Example of a Triplet and Diagonal Element

To discuss the accumulation of the changes in the elements which result in the error at the end of the sweep, let

$$T^k = \sum_{\substack{I(p,q)<k \\ p\neq q}} (a_{pq}^k)^2 \quad , \tag{IV.86}$$

that is, all elements which have been rotated during the sweep previous to k. Note that

$$T^{k_o} = 0 \quad , \tag{IV.87}$$

because no elements have been chosen, and

$$T^{k_o+N} = E^{k_o+N} \quad , \tag{IV.88}$$

because all elements have been chosen.

Previous to k_1 none of the elements of the triplet contribute to T^k. After k_1, T^k includes a_{pq}. From k_1 on, a_{pq} begins at zero and may be increased by the action of elements in other triplets of which it is a member until time k_2. Then a_{pr} becomes zero and is included in T^k. At the same time a_{pq} is affected according to Equation II.13, resulting in

$$a_{pq}^{k_2+1} = a_{pq}^{k_2} \cos \phi + a_{qr}^{k_2} \sin \phi \quad . \tag{IV.89}$$

Again, a_{pq} and now a_{pr} may be increased by other triplets, until k_3 when a_{qr} is included in T^k with a value of zero. At k_3, a_{pq} and a_{pr} are rotated together according to

$$a_{pq}^{k_3+1} = a_{pq}^{k_3} \cos \phi + a_{pr}^{k_3} \sin \phi \quad , \tag{IV.90}$$

$$a_{pr}^{k_3+1} = a_{pr}^{k_3} \cos \phi - a_{pq}^{k_3} \sin \phi \quad . \tag{IV.91}$$

However, rotation preserves the magnitude, so

$$(a_{pq}^{k_3+1})^2 + (a_{pr}^{k_3+1})^2 = (a_{pq}^{k_3})^2 + (a_{pr}^{k_3})^2 \, , \tag{IV.92}$$

and the contribution to T^{k_3} is not changed. From k_3 to the end of the sweep the elements are changed due to other triplets. Thus, the total effect on T^N due to this triplet is the change in a_{pq}^{k} at iteration k_2. To generalize, the rotation of the middle element of a triplet is the critical one.

The new a_{pq}^{k+1} given by Equation IV.87, is bounded by the relation

$$\left|a_{pq}^{k_2+1}\right| \leq \left|a_{pq}^{k_2}\right| + \left|a_{qr}^{k_2}\right| \, |\sin \phi| \quad , \tag{IV.93}$$

where ϕ is given by

$$\phi = \tfrac{1}{2} \arctan \left(\frac{2 \, a_{pr}^{k_2}}{a_{pp}^{k_2} - a_{rr}^{k_2}}\right), \; -\pi/4 \leq \phi \leq \pi/4 \quad . \tag{IV.94}$$

Note than

$$|\sin \phi| \leq \tfrac{1}{2}|2\phi| \leq \tfrac{1}{2}|\tan 2 \, \phi| = \frac{|a_{pr}^{k_2}|}{|a_{pp}^{k_2} - a_{rr}^{k_2}|} \, , \tag{IV.95}$$

so IV.93 implies

$$\left|a_{pq}^{k_2+1}\right| \leq \left|a_{pq}^{k_2}\right| + \frac{|a_{qr}^{k_2}| \; |a_{pr}^{k_2}|}{|a_{pp}^{k_2} - a_{rr}^{k_2}|} \quad . \tag{IV.96}$$

The total error at the end of the sweep is thus bounded by the sum of terms like last term in IV.96 for all triplets. To minimize $|a_{pq}^{k_2+1}|$ and therefore E^{k_o+N}, the ordering should try to promote two propositions

1. minimize $|a_{pr}^{k_2}|$ and $|a_{qr}^{k_2}|$, (IV.97)

2. maximize $|a_{pp}^{k_2} - a_{rr}^{k_2}|$. (IV.98)

For proposition 1, the only effect of the ordering is which of the elements of the triplet will become a_{pr} and a_{qr}. The decision must be made at time k_1 when a_{pq} is chosen. Therefore, proposition 1 is equivalent to

1b. maximize $|a_{pq}^{k_1}|$. (IV.99)

Proposition 1 is enforced directly in the original method of Jacobi. Hansen (1963) attempts to promote the same effect by indirect means. As noted in Chapter II, the values of the off-diagonal elements are not known in the optimization method.

The second proposition is useful because it concerns only the diagonal elements. Note that both the row and column orderings of Figure 1 choose the elements of every triplet in the order a_{pq}, a_{pr}, a_{qr} with $p < q < r$. Obviously,

$$|a_{pp} - a_{rr}| \geq |a_{pp} - a_{qq}| \tag{IV.100}$$

and

$$|a_{pp} - a_{rr}| \geq |a_{qq} - a_{rr}| \tag{IV.101}$$

together are equivalent to either

$$a_{pp} \leq a_{qq} \leq a_{rr} \tag{IV.102}$$

or

$$a_{pp} \geq a_{qq} \geq a_{rr} \ . \tag{IV.103}$$

Therefore, if the diagonal is sorted throughout the sweep, the row or column ordering will consistently satisfy Proposition 2.

It is interesting to note that the diagonal ordering of Figure 2 consistently violates the order a_{pq}, a_{pr}, a_{qr}. This property is not a direct contradiction to the testing of Hansen (1963) because the diagonals in his examples were not ordered and therefore could be considered random.

To use the previous results, a modification is proposed in which the entire diagonal is sorted in, say, decreasing order before each sweep. The sorting can be accomplished without disturbing the algorithm or results (except the order of the results) by a succession of exchanges of two rols and the corresponding two columns of A^k. For the optimization method, the corresponding two columns of the eigenvector matrix, S, and the two elements of b, z, etc., would also have to be exchanged. In the actual computational algorithm, the same effect is achieved at less expense by using a permutation or pointer vector to record the exchanges and changing the ordering. The remainder of this discussion, however, considers the modification matrix with sorted diagonal and the original ordering for the selection of pairs. The following theorem shows that sorting the diagonal elements of A does not invalidate the

theorems on convergence cited in Chapter II.

Theorem 1. If the special cyclic Jacobi method is modified before each sweep by permuting the rows and columns of A alike, such that a_{pp} is a monotonic sequence in p, then there exists an iteration k_1 after which the modification causes no further change to the matrix.

Corollary. Theorem 1 holds with the special cyclic ordering of pairs replaced by any ordering for which E^k converges to zero.

Proof. The exchange of two rows and the corresponding columns interchanges diagonal elements with diagonal elements and off-diagonal elements with off-diagonal elements. Therefore, the value of E^k of IV.76 is not changed by the permutations. Also, the modification changes the matrix only between sweeps. Therefore, the properties of a single rotation or of rotations within one sweep are not changed by the modification.

Forsythe and Henrici (1960) prove that

$$\lim_{k\to\infty} E^k = 0 \qquad \text{(IV.104)}$$

for the unmodified special cyclic Jacobi method. The proof depends only on properties of a single sweep (Lemmas 1, 2 and 3) and rotations within one sweep (Lemma 4 and the section labelled "proof of (11)"). Therefore, the proof applies to the modified method. Based only on the convergence of E^k to zero and the properties of matrices, Forsythe and Henrici prove that there exists a k_1 after which the permutations required to put the matrix diagonal in monotonic order does not change (Lemma 6). The modified method causes E^k to converge to zero, so the proof applies to the new method, proving Theorem 1. The corollary

follows directly from Lemma 6 of Forsythe and Henrici.

Theorem 2. The special cyclic Jacobi method modified as in Theorem 1 converges in the sense that

$$\lim_{k\to\infty} A^k = D \quad , \tag{IV.105}$$

where D is a diagonal matrix.

Proof. The proof of Theorem 1 includes the proof that E^k converges to zero. The remainder of the Forsythe and Henrici proof that the special cyclic Jacobi method converges in the sense of IV.105 applies to the modified method.

Alternately, Theorem 2 follows from Theorem 1 by applying the entire proof of Forsythe and Henrici to the iterations after the modification causes no changes. In the same way, the other relevant theorems cited in Chapter II apply to the modification method. In particular, the error E^k converges to zero quadratically. The quadratic convergence of the Jacobi method, with or without the modification, is used in the following discussion of the convergence of the whole optimization method.

Convergence of the Optimization Method

When the new method is applied to a general function bounded below, the sequence of function values cannot diverge as long as the algorithm never moves to a point unless the function value is improved. Therefore, the sequence is monotonic. Convergence to a point short of the optimum is a possibility, but several facts indicate that it is unlikely. First of all, the direction vectors in the method are always kept orthogonal, eliminating one of the problems of some other methods.

In addition, it has been noticed that even when the direction vectors change "randomly" the optimization progresses at a reduced rate. Finally, as the method converges, the direction vectors are still being corrected by small amounts, and this tends to prevent "stagnation."

For the minimization of a quadratic function, the new method does not appear to have finite convergence for n greater than two. Even if linear optimizations are used, the Jacobi method does not produce finite convergence of the direction vectors to the eigenvectors of the curvature matrix. The following analysis, however, shows that the distance to the optimum is bounded by a form which appears linear but depends on C. Under the conditions where the Jacobi process converges quadratically, the distance to the optimum converges to zero quadratically.

Consider the new method applied to the minimization of a quadratic form

$$f(x) = x^T A x \tag{IV.106}$$

with A symmetric and positive definite. Since the optimum is

$$\hat{x} = 0 \quad , \tag{IV.107}$$

the distance to the optimum is

$$|| x ||^2 = \sum_{p=1}^{n} (x_p)^2 \quad . \tag{IV.108}$$

In the notation of Chapter III, the current location x is the base location, x_o. Recall that the new method uses

$$C = S^T A S \tag{IV.109}$$

as in III.4, which corresponds to A^k in the discussion of the Jacobi method. Define z to be the error in the transformed space,

$$z = S^T x \quad , \tag{IV.110}$$

so that

$$f(Sz) = z^T C z \quad . \tag{IV.111}$$

Note that the definition of z used here is slightly different than that of III.2. The transformation S is orthogonal and normalized, so

$$|| z || = || x || \ .$$

For an ideal quadratic function like IV.106, the fitting process will produce the correct values for the elements of C regardless of position. For A positive definite,

$$c_{pp} = s_p^T A \, s_p > 0 \quad . \tag{IV.112}$$

Assume that all movement to the optimum is accomplished by an exact linear optimization each time a linear search is called for. Let the superscript k indicate values for the variables before the k-th linear optimization (counting from zero). Two linear optimizations are performed for each rotation of the direction vectors, so the index k here advances twice as fast as in the discussion of the Jacobi method. Therefore, a sweep consists of n(n-1) optimization iterations, rather than the $\frac{1}{2}n(n-1)$ rotation iterations for the Jacobi method.

Each optimization in direction s_p^k corresponds to solving equation p of

$$C^k z^k = 0 \qquad \text{(IV.113)}$$

for z_p^k. The solution is

$$z_p^{k+1} = \frac{-1}{c_{pp}^k} \sum_{\substack{q=1 \\ q \neq p}}^{n} c_{pq}^k \, z_q^k \qquad \text{(IV.114)}$$

When used for solving a system of linear equations, this repeated solution for each variable is known as relaxation. In relaxation, the indices p are usually chosen in consecutive order, giving the Gauss-Seidel method (see Schwarz, 1973). In the optimization method, p is chosen as i and then j of each pair (i,j), which usually does not result in consecutive order. As shown by the following three theorems, however, the order of choosing p does not eliminate the usual linear convergence characteristic of relaxation. Convergence is only slowed to the extent that more iterations are required before all z_p are reduced.

The three theorems each assume a k_o and establish a bound of the form

$$|| z^{k_1} ||^2 \leq \bar{r} \; || z^{k_o} ||^2 \qquad \text{(IV.115)}$$

for some k_1 after k_o. Theorem 3 considers the general case of relaxation with p chosen in any order and rotation of s_i and s_j (i.e., c_{ij}) after choosing p=i and p=j. The resulting bound applies for any interval (k_o, k_1-1) during which all p are chosen. Theorem 4 considers the conditions of the optimization method and relates the reduction of error over one sweep to the error measure of the Jacobi method.

$$E^k = \sum_{p=1}^{n} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2 \quad . \tag{IV.116}$$

Theorem 5 gives a better bound for the case of relaxation with constant c^k.

Theorem 3. Consider relaxation as in Equation IV.114 with p chosen in any order and

$$C^{k+1} = C^k \tag{IV.117}$$

except for rotations of c_{ij} (i.e., s_i and s_j) after choosing p=i and p=j. Let

$$r^k = \frac{1}{(c_{pp})^2} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2, \; p=p^k \quad , \tag{IV.118}$$

and

$$\bar{r} = \sum_{k=k_o}^{k_1-1} 2r^k \quad . \tag{IV.119}$$

For any interval (k_o, k_1-1) such that every p is chosen at least once, and such that

$$\bar{r} \leq 1 \quad , \tag{IV.120}$$

the relation

$$|| z^{k_1} ||^2 \leq \bar{r} \, || z^{k_o} ||^2 \tag{IV.121}$$

holds.

Proof. At any given iteration, Equation IV.114 may result in an increase in the value of z_p. The individual elements, however, are bounded by the norm of the vector. In order to isolate the new elements let e_p^k equal z_p^k for all p which have been chosen in the interval $(k_o, k-1)$ and e_p^k equal zero otherwise. At $k=k_o$, no p has been chosen, so

$$|| e^{k_o} ||^2 = 0 \ . \qquad \text{(IV.122)}$$

At $k=k_1$ all p have been chosen, so

$$|| z^{k_1} || = || e^{k_1} || \ . \qquad \text{(IV.123)}$$

From Equation IV.114,

$$e_p^{k+1} = \frac{-1}{c_{pp}^k} \sum_{\substack{q=1 \\ q\neq p}}^{n} c_{pq}^k \ z_q^k \ . \qquad \text{(IV.124)}$$

From the triangle inequality,

$$(e_p^{k+1})^2 \leq \frac{1}{(c_{pp}^k)^2} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2 \ || z^k ||^2 \ , \qquad \text{(IV.125)}$$

so from IV.118

$$(e_p^{k+1})^2 \leq r^k \ || z^k ||^2 \ . \qquad \text{(IV.126)}$$

At this point, it is necessary to use induction on k to prove

$$||e^k|| \leq \sum_{\ell_2=k_o}^{k-1} 2r^{\ell_2} ||z^{k_o}||^2 \quad . \tag{IV.127}$$

Assuming IV.127 holds for k, IV.120 implies

$$||e^k||^2 \leq ||z^{k_o}||^2 \quad . \tag{IV.128}$$

Each z_p^k equals either $z_p^{k_o}$ or e_p^k, so

$$||z^k||^2 \leq ||z^{k_o}|| + ||e^k||^2 \quad . \tag{IV.129}$$

Using IV.128

$$||z^k||^2 < 2||z^{k_o}||^2 \quad , \tag{IV.130}$$

and IV.126 becomes

$$(e_p^{k+1})^2 \leq 2r^k ||z^{k_o}||^2 \quad . \tag{IV.131}$$

The other elements of e^k are unchanged, so

$$||e^{k+1}||^2 \leq ||e^k||^2 + (e_p^{k+1})^2 \quad . \tag{IV.132}$$

Using IV.127 and IV.131

$$||e^{k+1}||^2 \leq \sum_{k_2=k_o}^{k-1} 2r^{k_2} ||z^{k_o}||^2 + 2r^k ||z^{k_o}||^2 \quad , \tag{IV.133}$$

so

$$||e^{k+1}||^2 \leq \sum_{k_2=k_o}^{k} 2r^{k_2} ||z^{k_o}||^2 \qquad \text{(IV.134)}$$

before rotation.

If a rotation affects only previously chosen elements, the rotated elements of z are included in e^{k+1}. A plane rotation does not change the Euclidean norm so $||z^{k+1}||^2$ and $||e^{k+1}||^2$ remain unchanged. Thus Relation IV.127 holds for k+1, after rotation. Using IV.122, relation IV.127 is true for $k-k_o$, and therefore, by induction, it is true for all $k \geq k_o$. Returning to the proof of Theorem 3, using IV.119 and IV.123, the relation IV.127 with $k=k_1$ implies IV.121 which proves the theorem.

Theorem 4. Under the conditions of Theorem 3, if p is chosen as i and then j of each pair (i,j) of the optimization method, and the interval (k_o, k_2-1) contains the iterations of one sweep, then Theorem 3 holds with $\bar{r}$ replaced by

$$\bar{r} = \frac{2n(n-1)\ E^k}{\min\limits_{1\leq q\leq n} \lambda_q^2} \ , \qquad \text{(IV.135)}$$

where λ_q is the q-th eigenvalue of A, and E^k is defined by IV.116.

Proof. From the properties of matrices, the eigenvalues of C are those of A, and

$$(c_{pp})^2 \geq \min_{1\leq q\leq n} \lambda_q^2 \ . \qquad \text{(IV.136)}$$

For A positive definite

$$\lambda_q > 0 \quad . \tag{IV.137}$$

Thus, IV.118 gives

$$r^k \leq \frac{1}{\min\limits_{1\leq q\leq n} \lambda_q^2} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2 \quad . \tag{IV.138}$$

Certainly,

$$\sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2 \leq \sum_{p=1}^{n} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2 = E^k \tag{IV.139}$$

with E^k of IV.116. For the Jacobi method,

$$E^k \leq E^{k_o} \quad , \tag{IV.140}$$

so IV.138 becomes

$$r^k \leq \frac{E^{k_o}}{\min\limits_{1\leq q\leq n} \lambda_q^2} \quad . \tag{IV.141}$$

Every p is chosen in a sweep which consists of n(n-1) iterations, so, since

$$k_1 = k_o + n(n-1) \quad , \tag{IV.142}$$

$$\sum_{\ell=k_o}^{k-1} r^{\ell} \leq \sum_{\ell=k_o}^{k_1-1} r^{\ell} \leq n(n-1) \frac{E^{k_o}}{\min\limits_{1\leq q\leq n} \lambda_q^2} . \qquad \text{(IV.143)}$$

Relations IV. 20 and IV.135 imply

$$2n(n-1) \frac{E^{k_o}}{\min\limits_{1\leq q\leq n} \lambda_q^2} \leq 1 , \qquad \text{(IV.144)}$$

so

$$\sum_{\ell=k_o}^{k-1} 2r^{\ell} \leq 1 . \qquad \text{(IV.145)}$$

Relation IV.145 can be used in place of IV.120 to prove that IV.127 implies IV.128. The remainder of the proof of Theorem 4 follows the proof of Theorem 3.

Theorem 5. If C^k is constant then the theorem is true with $\bar{r}$ replaced by

$$\bar{r} = \frac{2E^k}{\min\limits_{1\leq q\leq n} (c_{qq}^k)^2} . \qquad \text{(IV.146)}$$

Proof. Define

$$r_p = \frac{1}{\min\limits_{1\leq q\leq n} (c_{qq})^2} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq})^2 . \qquad \text{(IV.147)}$$

Note that the definition of IV.146 gives

$$\sum_{p=1}^{n} 2r_p = \frac{1}{\min\limits_{1\le q\le n} (c_{qq}^k)^2} \sum_{p=1}^{n} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq})^2 = \frac{2E^k}{\min\limits_{1\le q\le n} (c_{qq}^k)^2} = \bar{r}, \quad \text{(IV.148)}$$

so condition IV.120 is equivalent to

$$\sum_{p=1}^{n} 2r_p \le 1 \quad . \quad \text{(IV.149)}$$

If iteration k chooses p, then comparing IV.118 and IV.147

$$r^k = \frac{1}{(c_{pp}^k)^2} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2 \le \frac{1}{\min\limits_{1\le q\le n} (c_{qq}^k)^2} \sum_{\substack{q=1 \\ q\neq p}}^{n} (c_{pq}^k)^2 = r_p \quad . \quad \text{(IV.150)}$$

Thus, relation IV.126 becomes

$$(e_p^{k+1})^2 \le r_p \, ||z^k||^2 \quad . \quad \text{(IV.151)}$$

Again, induction on k is required to prove that for all p

$$(c_p^k)^2 \le 2r_p \, ||z^{k_o}||^2 \quad . \quad \text{(IV.152)}$$

Assuming IV.152 holds for k, summing over p gives

$$||e_p^k||^2 \le \sum_{p=1}^{n} 2r_p \, ||z^{k_o}||^2 \quad , \quad \text{(IV.153)}$$

and using IV.149,

$$||e_p^k||^2 \leq ||z^{k_o}||^2 \quad . \tag{IV.154}$$

Noting that IV.154 is identical to IV.128, IV.130 is assured. Using relation IV.130, IV.151 becomes

$$(e_p^{k+1})^2 \leq 2r_p \; ||z^{k_o}||^2 \quad . \tag{IV.155}$$

and IV.152 is shown for k+1. By definition,

$$e^{k_o} = 0 \quad , \tag{IV.156}$$

so IV.152 is true for k_o. By induction, IV.152 is true for all k. Using IV.148, relation IV.153 becomes

$$||e^k||^2 \leq \bar{r} \; ||z^{k_o}||^2 \quad . \tag{IV.157}$$

Recalling IV.123, relation IV.157 for $k=k_1$ becomes,

$$||z^{k_1}||^2 = ||e^{k_1}||^2 \leq \bar{r} \; ||z^{k_o}||^2 \quad . \tag{IV.158}$$

Since this is identical to IV.121, Theorem 5 is proved.

To summarize the results for the new method, Theorem 3 states that once

$$E^{k_o} \leq \frac{1}{n(n-1)} \min_{1 \leq q \leq n} \lambda_q^2 \quad , \tag{IV.159}$$

the optimization method produces

$$||z^{k_o+n(n-1)}||^2 \leq \frac{n(n-1)E^{k_o}}{\min\limits_{1\leq q\leq n} \lambda_q^2} ||z^{k_o}||^2 . \qquad \text{(IV.160)}$$

In view of IV.156,

$$||z^{k_o+n(n-1)}|| \leq ||z^{k_o}|| \qquad \text{(IV.161)}$$

so,

$$||z^{k_o}|| \leq ||z^o|| . \qquad \text{(IV.162)}$$

Therefore, IV.160 becomes

$$||z^{k_o+n(n-1)}||^2 \leq \frac{n(n-1)E^{k_o+N}}{\min\limits_{1\leq q\leq n} \lambda_q^2} ||z^o||^2 . \qquad \text{(IV.163)}$$

With $||z^o||^2$ constant, the error at the end of each sweep is bounded by a multiple of E^{k_o}. If E^{k_o} converges quadratically to zero, then $||z^{k_o}||^2$ converges to zero at least quadratically.

For distinct eigenvalues the convergence of the Jacobi method insures relation IV.159 will eventually be met. Relation IV.158 as well as the rate constant of relation IV.160 appear discouraging due to the n^2 factor. In practice, however, the Jacobi method converges fairly rapidly resulting in C being approximately constant. Thus, the rate will be closer to that of Theorem 5.

Summary

The analysis in this chapter provides several results. The calcu-

lation of model parameters must consider errors involved. A simple analysis of the errors indicates that the step size z should be limited in such a way as to keep cz^2 sufficiently large relative to the estimated error in the function value and relative to other similar terms.

The order of choosing pairs of indices affects the convergence of the Jacobi method. The analysis suggests an ordering based on sorting the diagonal. Theorem 1 shows that previous proofs regarding convergence apply when the new ordering is used.

Convergence of the new method to the optimum of a quadratic function is analogous to that of iterative methods, such as Gauss-Siedel, for solving the system of linear equations

$$C^k z^k = 0 \quad . \tag{IV.164}$$

The linear equation methods are known to converge linearly. In the new method, however, the matrix C changes. Theorem 4 shows that the rate of linear convergence includes the function of C used as an error measure for the Jacobi method. Under conditions where the Jacobi error measure converges to zero quadratically, the optimization method converges quadratically.

CHAPTER V

TEST PROGRAM

Introduction

This chapter describes the FORTRAN program used for testing the new optimization method. The program listing is given in Appendix A. The algorithm of Chapter III is used with the step size limits of Chapter IV. Also, the modified ordering discussed in Chapter IV, with the diagonal sorted, is available as an option. The test results are reported in Chapter VI.

The program is segmented to allow testing of various alternatives for some sections. The routines comprising the program are listed in Table I, along with the purpose and subroutines called. Only one of the ordering subroutines SWEP1, SWEP2, or SWEP3, is used for a single solution. Various versions of subroutines INIT and EVAL are used to specify the various test problems. Most information passed between routines is stored in COMMON. For a description of important program variables, see Table II.

The program includes options to test the effects of the following:

1. Various orderings for choosing pairs of indices.
2. Sorting the diagonal of the curvature matrix.

In addition, the various constants for iteration and step size limits can be modified. The control cards are detailed in Table III.

TABLE I

ROUTINES USED IN COMPUTER PROGRAM

Routine	Purpose	Subroutines Called
MAIN	Initialize and iterate	INIT, EVAL, PUT, SCS, SWEP1, SWEP2, SWEP3
SORTER	Sort diagonal of C	
SCS	Multiply matrices	
PUT	Print table	
SWEP1	Column ordering	SORTER, PLANE, TALOR
SWEP2	Diagonal ordering	SORTER, PLANE, TALOR
SWEP3	Optional user specified ordering	SORTER, PLANE, TALOR
PLANE	Fit bivariate model	FIT, TRY, PUT
FIT	Fit univariate model	TRY
TRY	Sample function and save best point	EVAL
TALOR	Sample optimum of complete model	EVAL, PUT
INIT	Initialize problem	
EVAL	Evaluate function	

TABLE II

IMPORTANT PROGRAM VARIABLES

VARIABLE	DEFINITION*
AAA(15, 15)	(IN MAIN) CURVATURE MATRIX OF THE MODEL, S*C*ST, CALCULATED AT THE END OF OPTIMIZATION.
BB(15)	GRADIENT FOR MODEL.
BETER	(LOGICAL). INDICATES A SAMPLE HAS IMPROVED FUNCTION VALUE.
BPUT	(LOGICAL). OPTION TO CALL SUBROUTINE PUT.
CC(15)	DIAGONAL OF MODEL CURVATURE MATRIX. EQUALS UNIVARIATE CURVATURE. APPROXIMATE EIGENVALUES.
COPY	(LOGICAL). OPTION TO PRINT INPUT.
DEGRE	THE VALUE 180/PI.
DIR	+1 TO MAXIMIZE, -1 TO MINIMIZE. (-1)
DONE	(LOGICAL). SIGNALS TERMINATION.
FOUR	THE VALUE FOUR.
FREER	(LOGICAL). OPTION TO USE OPTIMUM FOR BIVARIATE FIT. FALSE USES UNIVARIATE SAMPLE STEP SIZES. (FALSE)
FREE5	(LOGICAL). OPTION TO BYPASS ROTATION RESTRICTION. FALSE PREVENTS CREATING REVERSE CURVATURE. (TRUE)
IBEST	NUMBER OF SAMPLES WHICH IMPROVED FUNCTION VALUE.
IFIT	NUMBER OF UNIVARIATE FITS.
IMOVE	NUMBER OF MOVES OF BASE POINT.
IPLAN	NUMBER OF BIVARIATE FITS.
IPRM	(IN MAIN) LOCATION OF PARAMETER TO BE CHANGED. ALSO, TEMPORARY TO READ NSWEP.
IPRM2	(IN MAIN) TEMPORARY TO READ NSAMP TO ALLOW DEFAULT TO PREVIOUS VALUE.
ISWEP	NUMBER OF SWEEPS.
ITRY	NUMBER OF FUNCTION EVALUATIONS.
ITALR	NUMBER OF TAYLOR SAMPLES.
KKPRM(3)	(IN MAIN) ARRAY OVERLAYED ON FIXED POINT PARAMETERS.
KPRT	UNUSED. PROVISION FOR OUTPUT UNIT NUMBER.
KR	INTEGER TO SET FREER. 0 SETS FALSE, 1 SETS TRUE. (0)
KRDR	UNUSED. PROVISION FOR INPUT UNIT NUMBER.
KSWEP	(IN MAIN) CHOICE OF ORDERING - 1=COLUMN, 2=DIAGONAL, 3=USER DEFINED.
K5	INTERGER TO SET FREE5. 0 SETS FALSE, 1 SETS TRUE. (1)
M	MAXIMUM NUMBER OF VARIABLES. (15)
N	NUMBER OF VARIABLES.
NPAIR	NUMBER OF PAIRS IN A SWEEP: N(N-1)/2
NSAMP	LIMIT ON NUMBER OF FUNCTION EVALUATIONS. DEFAULT IS 1000.
NSWEP	LIMIT ON NUMBER OF SWEEPS. DEFAULT IS 25.
N3	LIMIT ON NUMBER OF RETRIES OF UNIVARIATE FIT. (1)

TABLE II (Continued)

VARIABLE	DEFINITION*
CNE	THE VALUE ONE.
PI	THE VALUE PI.
PPRM(23)	(IN MAIN) ARRAY OVERLAYED ON FLOATING POINT PARAMETERS IN COMMON.
PRINT	(LOGICAL). OPTION TO PRINT AFTER EACH BILINEAR FIT.
PRM	(IN MAIN) NEW PARAMETER VALUE.
REV	(LOGICAL). OPTION TO SORT DESCENDING RATHER THAN ASCENDING.
SMALS	SMALLEST VALUE OF ELEMENT OF S USED TO CALCULATE LIMIT TO INSURE CHANGE IN X. VALUE IS 1/SQRT(N).
SMALX	NON-RELATIVE LOWER BOUND ON ESTIMATED ERROR IN X. (1E-40)
SMALY	NON-RELATIVE LOWER BOUND ON ESTIMATED ERROR IN Y. (1E-60)
SORT	(LOGICAL). OPTION TO MODIFY ORDERING TO SORT. DIAGONAL.
SS(15)	CURRENT DIRECTION VECTOR.
SSS(15,15)	MATRIX OF DIRECTION VECTORS. APPROXIMATE EIGENVECTORS.
TOLX	UNUSED. PROVISION FOR TERMINAL ACCURACY.
TRACE	(LOGICAL). OPTION TO PRINT AFTER EVERY FUNCTION EVALUATION.
TYLC	LOWER LIMIT ON CZZ RELATIVE TO MAX CZZ. (.1)
TYLX	LOWER LIMIT ON CZZ RELATIVE TO ERROR IN X. (1E-10)
TYLY	LOWER LIMIT ON CZZ RELATIVE TO ERROR IN Y. (1E-10)
TWO	THE VALUE TWO.
TZL	LOWER LIMIT ON Z RATIO. (.01)
TZLF	LOWER LIMIT ON Z RATIO FOR FITTING. (.3)
TZLR	FRACTION OF PREVIOUS LOWER LIMIT BASED ON ERROR. (.1)
TZLT	LOWER LIMIT ON Z RATIO AFTER TAYLOR SAMPLE. (.1)
TZLX	LOWER LIMIT ON Z TO INSURE CHANGE IN X. (1E-10)
TZUB	UPPER LIMIT ON Z RATIO IF BETTER POINT HAS BEEN FOUND.(10)
TZUFB	SAME AS TZUB FOR FITTING. (2)
TZUF1	UPPER LIMIT ON Z RATIO FOR FITTING IF BETTER POINT NOT FOUND AFTER FIRST SAMPLE. (1)
TZUF2	SAME AS TZUF1 FOR AFTER SECOND SAMPLE. (.5)
TZUP	UPPER LIMIT ON Z RATIO FOR BIVARIATE FIT. (2)
TZUT	UPPER LIMIT ON Z RATIO AFTER TAYLOR SAMPLE . (1)
TZU1	UPPER LIMIT ON Z RATIO IF BETTER POINT NOT FOUND. (.5)
TZO	FRACTION OF X FOR INITIAL Z. (.1)
XXBES(15)	X FOR BEST SAMPLE THUS FAR.
XXO(15)	BASE LOCATION.
YBEST	F(X) FOR BEST SAMPLE THUS FAR.
YL	LOWER LIMIT ON CZZ. (CHANGE IN Y)
YYC(15)	CZZ. (CHANGE IN Y)
YO	F(X) FOR BASE LOCATION.
ZBEST	STEP SIZE FOR BEST SAMPLE.
ZERO	THE VALUE ZERO.
ZZ(15)	LIMITED OPTIMUM STEP SIZE.
ZZL(15)	LOWER LIMIT ON Z.
ZZOPT(15)	OPTIMUM STEP SIZE WITH NO LOWER LIMIT.

* The value in parenthesis after definition is the default used in testing.

TABLE III

CONTROL CARDS

Card	Column	Format	Variable	Value	Meaning
1	1	I1	KSWEP	0	End of job
				1	Column ordering
				2	Diagonal ordering
				3	User ordering
	2	L1	SORT	*	Sort diagonal
	3	L1	REV	T	Sort descending
				F	Sort ascending
	4	L1	COPY	*	Print control cards
	5	L1	PRINT	*	Print results after bivariate fit
	6	L1	TRACE	*	Print every sample
	7	L1	BPUT	*	Call subroutine PUT
2	1-5	I5	NSWEP		Limit on number of sweeps (defaults to 25)
	6-10	I5	NSAMP		Limit on number of function evaluations (defaults to 1000)
3-n	1-5	I5	IPRM	>0	Location for real program constant
				0	End of modifications to program constant
				<0	Location for integer program constant
	6-15	E10.0	PRM		Value for program constant

* Value of "T" activates option; value of "F" deactivates (defaults to "F").

Main Program

The main program initializes the program and performs the main iteration loop. First, constants and parameters are initialized. Then control cards are read to set the options and counter limits. Additional cards are read to specify optional changes to algorithm parameters. Next, the model is initialized. Subroutine INIT is called to initialize the function, obtaining in return the number of variables, initial base location and initial step sizes. The parameter KSWEP chooses the proper ordering subroutine, SWEP1, SWEP2, or SWEP3, which is called for initialization.

The main iteration loop is then begun, in which the proper ordering subroutine is called. The ordering subroutine calls the other subroutines to perform the actual optimization. The loop is terminated when convergence is signalled or when the iteration limits on either the number of sweeps or the number of function evaluations is exceeded. Following the optimization, the final direction vectors and associated curvatures are printed. Subroutine SCS is called to calculate the curvature matrix, in the original coordinates, which is then printed. The program is then restarted at the point of reading control cards. The program is terminated when a blank option card is read.

Utility Subroutines

Subroutine SORTER sorts the diagonal of C. The actual interchanges are applied to a permutation vector MAP. The ordering subroutines then generate a modified ordering which simulates operations with C sorted. Sorting is descending if REV is true and ascending if REV is false. The

algorithm used for sorting is simple adjacent pair interchange, because after a few iterations the changes in MAP are infrequent.

Subroutine SCS calculates the curvature matrix A as in Equation III.4.

Subroutine PUT formats and prints any desired information at each iteration. Its main purpose is printing tables. One argument is passed from the calling program to indicate the current location in the calculations according to the scheme shown in Table IV.

TABLE IV

ARGUMENT FOR SUBROUTINE PUT

Value	Current Location
0	After initialization
1	After univariate
2	After bivariate fit and rotation
3	After entire sweep and Taylor sample

Ordering Subroutines

Subroutines SWEP1, SWEP2, and SWEP3 accomplish the ordering of pairs of indices. The first time the subroutine is called, the permutation

vector MAP is initialized. After that, one sweep is made each time the subroutine is called. Before each sweep, subroutine SORTER is called, if SORT is true. Pairs of indices (i,j) are then chosen, and modified to simulate operations with the diagonal of C sorted. Subroutine PLANE is called for each pair. After the sweep, subroutine TALOR is called.

Subroutine SWEP1 chooses pairs in column ordering as in Figure 1b. Subroutine SWEP2 chooses pairs in a diagonal ordering to favor two consecutive occurrences of each index as in Figure 8b. Subroutine SWEP3 is a dummy subroutine to allow the addition of another ordering. The parameter KSWEP chooses the appropriate ordering subroutine.

Model Updating Subroutines

Subroutine PLANE corrects the bivariate model of the function in the plane of the two chosen direction vectors and rotates the direction vectors. First, y_L of Equation IV.76 is calculated. Then subroutine FIT is called for direction s_i, giving b_i, c_{ii}, and a component size for the bivariate step. The base location is moved to the best point found. Subroutine FIT is then called again for direction s_j.

A temporary direction vector and step size are calculated and subroutine EVAL is used to sample one more point. The value of c_{ij} is then calculated, but limited to prevent the rotation from changing the sign of c_{ii} or c_{jj}. The base location is again moved to the best point found. The rotation matrix elements are calculated and b, C and S are rotated. New values for the univariate step sizes are calculated, limited by values calculated in subroutine FIT.

Subroutine FIT calculates the corrected model along a line. The samples are used to search for an improved location as discussed in

Chapter III. In addition to the error bounds of Chapter IV, heuristic bounds are used to insure the independent variables change and to keep the step size from growing or shrinking at more than a given rate. For a flowchart, see Figure 14.

First, a limit to insure x changes is found and applied to the previous step size. The first point is sampled and b is corrected. Limits are calculated and a new step size is found. The second point is sampled and b and c are corrected. Limits and step sizes are again calculated. If the previous samples do not bracket the estimated optimum, the worst of the two samples is replaced and the model calculations are repeated. Otherwise, the step size limit based on the error estimate of IV.75 is calculated. This limit is not calculated earlier because a reliable estimate of c is required.

If one of the samples has improved the function value, the fitting process is ended. If the function value has not been improved and the step size cannot be reduced or the number of recalculations exceeds a limit (usually two), one search point without a lower limit on step size is sampled and the process is ended. If none of the previous conditions has terminated the process, the fit is recalculated with new step size limits. If the desired step size for searching is within the limits for the second step, only the worst of the previous sample points is replaced for recalculation. Otherwise, both samples are replaced.

Before returning to the calling program, the exit section is performed. The exit section calculates one limit for use after the bivariat fit. Other limits are applied to the current step size for use in the bivariate fit.

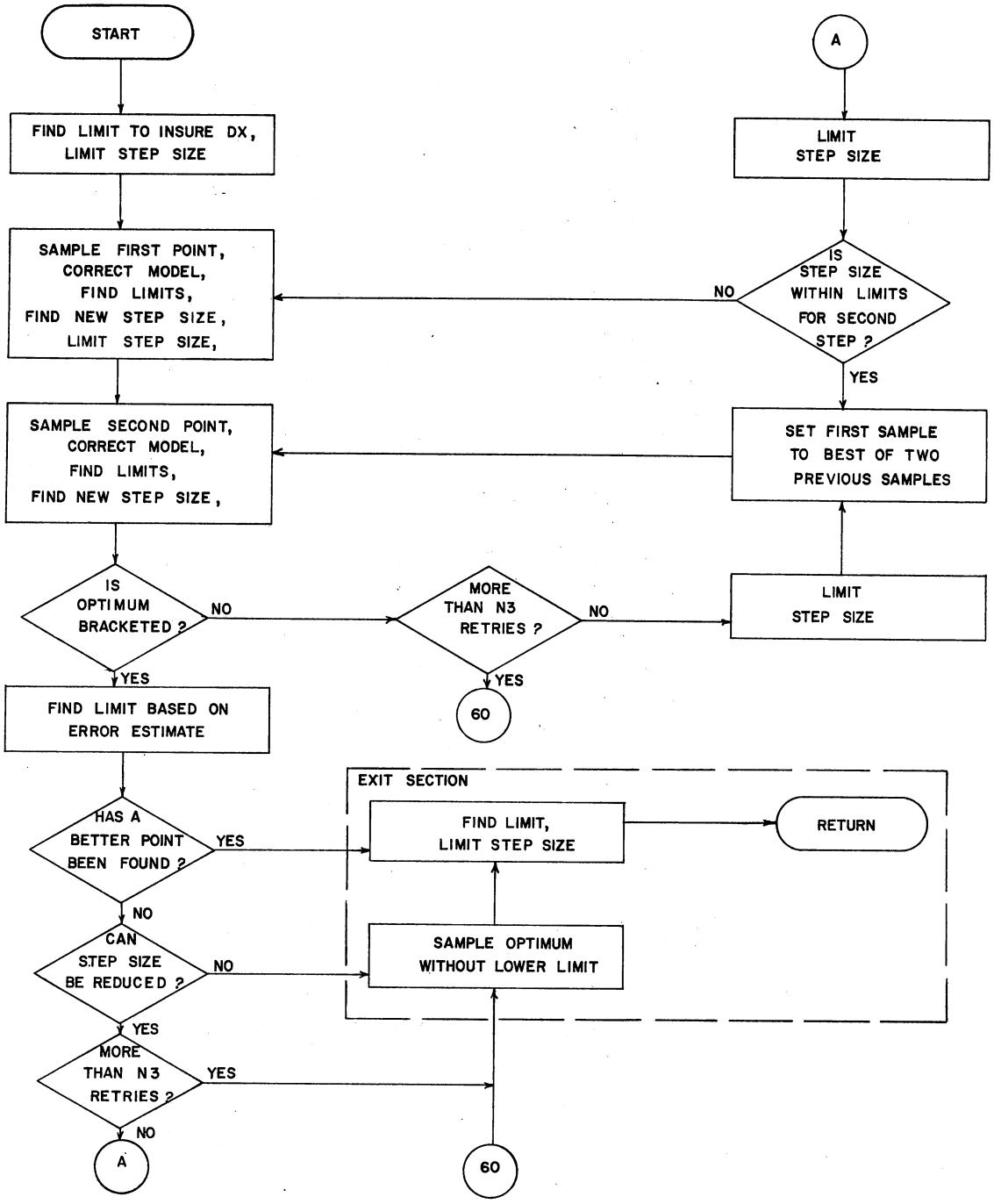

Figure 14. Flowchart of Subroutine FIT

Function Sampling Subroutines

Subroutine TRY samples the function and performs housekeeping tasks. The sample location is calculated from the direction vector and step size. After subroutine EVAL is called, the change in y, DY, and first order slope, B, are calculated. If the new point is better than the current best point, the current best point is updated.

Subroutine TALOR samples the function at the (upper-limited) optimum of the complete model and tests for convergence. First, the location of the optimum is calculated, along with model information in the direction of the optimum. Subroutine TRY is used to sample the point. The model is then corrected for the change in base location with lower limits on the new step sizes.

Problem Definition Subroutines

The purpose of subroutine INIT is to set initial conditions for a given problem. The conditions are returned to the main program by subroutine parameters. The parameters are

X(15)	Initial location (independent variables)
Z(15)	Initial step size. Default is 0.1X
DIR	+1 to maximize, -1 to minimize
N	Number of variables

For some problems INIT also sets initial conditions for the function evaluation subroutine, EVAL. For example, in curve fitting, the data points are read from cards. The information is passed to EVAL by the use of separate COMMON storage.

Subroutine EVAL calculates the function value for one sample point. The sample location is specified by a subroutine parameter and the function value is returned in the same way. The parameters are

F	Function value
X(15)	Sample location (independent variables)
N	Number of variables

The number of variables is included in the parameters for EVAL because several of the test problems are general and can be specified with any dimension.

CHAPTER VI

EXPERIMENTAL RESULTS

Introduction

The program described in Chapter V has been tested on an IBM 360/65 computer with double precision (about 15 decimal digits). This chapter summarizes the results of the testing along with comparisons to other methods. It should be kept in mind that the results for other methods have been obtained on various computers with various precisions. Normally, however, the precision will only affect the final stages of convergence.

A convergence criterion was not programmed. As indicated in Chapter III, the convergence criterion is commonly considered to be a separate problem. Convergence was not used to compare to other methods because many different criteria have been used, some of which are affected by the precision of the machine.

The information given for each function includes the number of sweeps, n_s, the number of linear searches, n_ℓ, and the number of function calls, n_f. Where it is of interest the approximation to the smallest eigenvalue, c_{min}, is listed. The smallest eigenvalue usually converges slowest giving an indication of the convergence of the direction vectors. The choice of ordering is indicated by COL for the ordering by columns and SEQ for the ordering to favor two occurrences of each index in turn.

SORT indicates an ascending sort of the diagonal of the curvature matrix, which implies searching "down the valley" first. REVERSE indicates a descending sort or searching "across the valley" first. As noted in Chapter IV, the "valley" depends partly on the scale; here it is assumed that the valley is not produced entirely by poor scaling.

Three Variable Quadratic Function

The function

$$f(x) = 3366(x_1^2 + x_2^2 + x_3^2 - x_1x_2 - x_1x_3 - x_2x_3)$$

$$+ (x_1^2 + x_2^2 + x_3^3) + \sqrt{3}\ (x_2 - x_1)(x_1 + x_2 - 2x_3)$$

is a quadratic with a very steep-sided valley. Minimization beginning at (10, 10, 10) results in a minimum at (0, 0, 0).

The results for column ordering without sorting are shown in Table V. The sudden convergence in the third sweep is explained by the convergence of the approximate eigenvectors as shown in Table VI. The Jacobi method requires two iterations to approximate the direction vectors (and the eigenvalues). As shown in Chapter IV, the accuracy of the eigenvectors then determines the rate of convergence of the x to the optimum.

Eight Variable Quadratic Function

A larger quadratic function was created to compare orderings. The function

$$f(x) = (x - \hat{x})^T A(x - \hat{x})$$

TABLE V

THREE VARIABLE QUADRATIC FUNCTION
WORK COLUMN ORDERING, NO SORTING

n_s	n_ℓ	n_f	f	x_1	x_2	x_3
0	0	0	$.300 \times 10^{3}$	10.000	10.000	10.000
1	6	24	$.299 \times 10^{3}$	9.981	9.984	9.983
2	12	44	$.294 \times 10^{3}$	9.902	9.902	9.902
3	18	64	$.255 \times 10^{-16}$	0.000*	0.000*	0.000*
4	24	86	$.231 \times 10^{-29}$	0.000*	0.000*	0.000*

* Zero to significant figures printed

TABLE VI

APPROXIMATE EIGENVALUES FOR
THREE VARIABLE QUADRATIC

n_s	f	C_{11}	C_{22}	C_{33}
0	$.300 \times 10^{3}$	-	-	-
1	$.299 \times 10^{3}$	740.65	14770.	4691.
2	$.294 \times 10^{3}$	2.25	15050.	5150.
3	$.255 \times 10^{-16}$	2.00	15050.	5150.
4	$.231 \times 10^{-29}$	2.000	15050.	5150.

was expressed in the form:

$$f(x) = (x - \hat{x})^T \; SCS^T \; (x - \hat{x})$$

so that eigenvalues and eigenvectors could be specified directly. To provide interaction between variables, the eigenvectors were chosen to be

$$S = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} ,$$

which is the Hadamard matrix of order eight (Hadamard, 1893; Paley, 1933). The diagonal elements of C,

$$\{c_{ii}\} = \{1, 1025, 1281, 1345, 1361, 1365, 1366, 1367\} ,$$

were used to provide both grouped and spread eigenvalues. (Due to the scaling of S, the eigenvalues are eight times the c_{ii} values.) The equivalent matrix is

$$A = \begin{pmatrix} 9111 & -1093 & -1607 & -963 & -1807 & -1083 & -1593 & -957 \\ -1093 & 9111 & -963 & -1607 & -1083 & -1807 & -957 & -1593 \\ -1607 & -963 & 9111 & -1093 & -1593 & -957 & -1807 & -1083 \\ -963 & -1607 & -1093 & 9111 & -957 & -1593 & -1083 & -1807 \\ -1807 & -1083 & -1593 & -957 & 9111 & -1093 & -1607 & -963 \\ -1803 & -1807 & -957 & -1593 & -1093 & 9111 & -963 & -1607 \\ -1593 & -957 & -1807 & -1083 & -1607 & -963 & 9111 & -1093 \\ -957 & -1593 & -1083 & -1807 & -963 & -1607 & -1093 & 9111 \end{pmatrix}$$

Optimum and initial points used were

$$\hat{x}^T = (2, 1, 1, 1, 1, 1, 1, 1)$$

and

$$x_o^T = (1, 2, 3, 4, 5, 6, 7, 8)$$

respectively.

The results for column ordering in Table VII compared to those for the three variable quadratic show the tendency of Jacobi method to converge in the same number of sweeps regardless of n. Of course, each sweep includes n(n-1) rotations so the number of calculations (and the number of function calls) is roughly proportional to n^2.

The convergence of the Jacobi method for the different orderings indicated by the smallest eigenvalues is shown in Table VIII. As expected, the column ordering is better for the initial convergence of the Jacobi method. For the column ordering, sorting the diagonal gives slightly faster convergence. Convergence to the optimum follows the

TABLE VII

EIGHT VARIABLE QUADRATIC FUNCTION WITH COLUMN ORDERING, NO SORTING

n_s	n_ℓ	n_f	f	c_{min}
0	0	0	$.264 \times 10^{6}$	-
1	56	164	$.135 \times 10^{4}$	30.6
2	112	338	$.251 \times 10^{1}$	9.4
3	168	504	$.831 \times 10^{-18}$	8.0
4	224	645	$.248 \times 10^{-28}$	8.0

TABLE VIII

EFFECT OF ORDERING ON EIGHT VARIABLE QUADRATIC PROBLEM

ORDERING	C_{min} $n_s=1$	C_{min} $n_s=2$	C_{min} $n_s=3$	f at $n_s = 3$
COL	30.6000	9.3928	8.0001	$.831 \times 10^{-18}$
COL, SORT	30.6366	8.0015	8.0000	$.129 \times 10^{-20}$
COL, REVERSE	30.6366	8.0000	8.0000	$.604 \times 10^{-3}$
SEQ	73.3525	10.1971	8.0001	$.269 \times 10^{-9}$
SEQ, SORT	73.3525	32.6754	8.0003	$.138 \times 10^{-9}$
SEQ, REVERSE	73.3525	8.0488	8.0000	$.114 \times 10^{-18}$

accuracy of the eigenvectors, except for the column ordering with descending sort. The comparison of convergence to the optimum for different orderings is discussed later.

Rosenbrock's Function

The curved valley of Rosenbrock (1960) is well known. The function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

is minimized beginning at (-1.2, 1.0). The minimum is (1.0, 1.0). Although artificial, the problem is useful for development and comparison of methods because it involves a large number of simple iterations. Ordering is not important because there is only one pair. The results in Table IX show that a function value of $.9 \times 10^{-11}$ is attained in 137 function evaluations. Powell (1964) reports reaching $.7 \times 10^{-11}$ in 151 function evaluations. Fletcher (1965) reports Powell's method reaches $.4 \times 10^{-8}$ in 145 function evaluations and the method of Davies, Swann and Campey reaches $.4 \times 10^{-6}$ in 169 function evaluations. For comparison, Rosenbrock (1960) took 200 function evaluations to reach $.1 \times 10^{-7}$. On the other hand, the gradient method of Fletcher and Powell (1963) reaches $.1 \times 10^{-7}$ in 18 gradient evaluations which corresponds roughly to 54 function evaluations.

Powell's Quartic Function

Powell (1962) gives a function,

$$f(x) = (x_1+10x_2)^2 + 5(x_3-x_4)^2 + (x_2-x_3)^4 + 10(x_1-x_4)^4 ,$$

TABLE IX

ROSENBROCK'S FUNCTION

n_s	n_ℓ	n_f	f	x_1	x_2
0	0	0	$.240 \times 10^{2}$	-1.200 000	1.000 000
1	2	6	$.427 \times 10^{1}$	-1.064 214	1.143 571
2	4	13	$.409 \times 10^{1}$	-1.022 067	1.040 780
3	6	20	$.353 \times 10^{1}$	-.867 636	.732 635
4	8	26	$.322 \times 10^{1}$	-.733 562	.570 513
5	10	34	$.275 \times 10^{1}$	-.631 673	.369 214
6	12	42	$.208 \times 10^{1}$	-.412 319	.140 509
7	14	50	$.173 \times 10^{1}$	-.095 359	-.063 488
8	16	60	.982	.015 023	-.010 817
9	18	66	.644	.218 761	.295 069
10	20	74	.425	.385 128	.126 666
11	22	80	.170	.589 127	.344 217
12	24	90	$.934 \times 10^{-1}$	.702 148	.486 196
13	26	96	$.572 \times 10^{-1}$	.764 591	.580 323
14	28	102	$.145 \times 10^{-1}$	.907 213	.815 360
15	30	110	$.324 \times 10^{-2}$	.945 473	.892 296
16	32	116	$.110 \times 10^{-2}$	.968 518	.936 968
17	34	122	$.152 \times 10^{-3}$	1.006 836	1.014 744
18	36	128	$.101 \times 10^{-6}$	.999 686	.999 377
19	38	136	$.902 \times 10^{-11}$	.999 999	.999 998
20	40	144	$.384 \times 10^{-14}$	1.000 000	1.000 000

to be minimized from (3, -1, 0, 1). The function is difficult because the curvature matrix is double singular at the minimum (0, 0, 0, 0). The resulting linear convergence can be seen in Table X for column ordering with the diagonal sorted. The corresponding approximate eigenvalues in Table XI show the two zero eigenvalues. Also note that the non-zero eigenvectors are approximated in only two sweeps. The function value is reduced to $.88 \times 10^{-9}$ in 223 function evaluations. Fletcher (1965) reports Powell's method reaches $.43 \times 10^{-9}$ in 208 function evaluations and the method of Davies, Swann and Campey reaches $.13 \times 10^{-9}$ in 180 evaluations.

Random Matrix Function

Fletcher and Powell (1963) present the function

$$f(x) = \sum_{i=1}^{n} (E_i - \sum_{j=1}^{n} (A_{ij} \sin x_j + B_{ij} \cos x_j))^2,$$

where the elements of A and B are random numbers between -100 and 100, and elements of $\hat{x}$ are random numbers between $-\pi$ and π. For the chosen $\hat{x}$

$$E_i = \sum_{i=1}^{n} (A_{ij} \sin \hat{x}_j + B_{ij} \cos \hat{x}_j)$$

Elements of x_o are displaced from those of $\hat{x}$ by random numbers between $-.1\ \pi$ and $.1\pi$.

Testing was done for n of 3, 5, and 10. For each value of n, three different random problems were created. The number of function evaluations required to reduce the error in the independent variables

TABLE X

POWELL'S QUARTIC FUNCTION WITH COLUMN ORDERING, DIAGONAL SORTED

n_s	n_ℓ	n_f	f	x_1	x_2	x_3	x_4
0	0	0	$.335\text{x}10^{3}$	$.300\text{x}10^{1}$	$-.100\text{x}10^{1}$	.0	$.100\text{x}10^{1}$
1	20	42	.570	.941	.909	.356	.553
2	40	75	$.163\text{x}10^{-1}$	.319	$.324\text{x}10^{-1}$	.120	.124
3	60	111	$.711\text{x}10^{-3}$	$.517\text{x}10^{-1}$	$.556\text{x}10^{-2}$	$-.345\text{x}10^{-1}$	$-.376\text{x}10^{-1}$
4	80	149	$.525\text{x}10^{-5}$	$.219\text{x}10^{-1}$	$.220\text{x}10^{-2}$	$-.495\text{x}10^{-2}$	$-.491\text{x}10^{-2}$
5	100	188	$.352\text{x}10^{-6}$	$.764\text{x}10^{-2}$	$.762\text{x}10^{-3}$	$-.581.10^{-2}$	$-.582\text{x}10^{-2}$
6	120	223	$.880\text{x}10^{-9}$	$.512\text{x}10^{-2}$	$.512\text{x}10^{-3}$	$.244\text{x}10^{-2}$	$.244\text{x}10^{-2}$
7	140	257	$.733\text{x}10^{-11}$	$.154\text{x}10^{-2}$	$.154\text{x}10^{-3}$	$.644\text{x}10^{-3}$	$.644\text{x}10^{-3}$
8	160	292	$.390\text{x}10^{-11}$	$.942\text{x}10^{-3}$	$.942\text{x}10^{-4}$	$.152\text{x}10^{-3}$	$.152\text{x}10^{-3}$
9	180	332	$.161\text{x}10^{-13}$	$.282\text{x}10^{-3}$	$.282\text{x}10^{-4}$	$.827\text{x}10^{-4}$	$.827\text{x}10^{-4}$
10	200	365	$.778\text{x}10^{-16}$	$.590\text{x}10^{-4}$	$.590\text{x}10^{-5}$	$.103\text{x}10^{-4}$	$.103\text{x}10^{-4}$

TABLE XI

CONVERGENCE OF EIGENVALUES FOR
POWELL'S FUNCTION

n_s	C_{11}	C_{22}	C_{33}	C_{44}
1	$.235 \times 10^{1}$	166.0078	121.8056	$-.355 \times 10^{-14}$
2	$.146 \times 10^{1}$	200.5988	15.0593	$.191 \times 10^{2}$
3	.0	202.2410	21.0133	$.207 \times 10^{1}$
4	$.127 \times 10^{-1}$	201.8421	20.0471	$.750 \times 10^{-1}$
5	$.367 \times 10^{-2}$	202.0033	20.0284	$.770 \times 10^{-1}$
6	$.386 \times 10^{-3}$	202.0003	20.0000	$.248 \times 10^{-2}$
7	$.262 \times 10^{-4}$	201.9997	20.0001	$.338 \times 10^{-5}$
8	$.293 \times 10^{-5}$	202.0000	20.0000	$.120 \times 10^{-3}$
9	$.259 \times 10^{-5}$	202.0000	20.0000	$.229 \times 10^{-4}$
10	$.854 \times 10^{-6}$	202.0000	20.0000	$.104 \times 10^{-5}$

to 10^{-4} is given in Table XII. Comparable results given by Powell (1964) for his own method and that of Rosenbrock (1960) are shown in Table XIII. In several cases the new method converges to a local optimum other than the intended one. Fletcher and Powell indicate that there are many optima and convergence to a different one sometimes occurs. The first of the three cases for n = 10 fails to achieve 10^{-4} accuracy in ten sweeps with either ordering. When terminated, the program was apparently approaching a different local optimum.

Curve Fitting Problems

Curve fitting is a more realistic problem and a difficult one. Osborne (1971) presents two models to be fit to tabulated data. The function to be minimized becomes

$$f(x) = \sum_{i=1}^{m} (F(t_i, x) - y_i)^2 .$$

The first model is

$$F(t, x) = x_1 + x_2 \exp(-x_4 t) + x_3 \exp(-x_5 t) .$$

The data values are given in Table XIV. Results for column ordering are shown in Table XV. The eigenvalues given by the program,

(14, 2.3, .00004, 132000, 4000)

indicate that the problem is another very narrow valley. The corresponding eigenvectors,

TABLE XII

NUMBER OF SWEEPS AND FUNCTION EVALUATIONS TO REACH $|x_i - \hat{x}_i| < 10^{-4}$ FOR RANDOM MATRIX FUNCTION WITH NO SORTING

	COL		SEQ	
n	n_s	n_f	n_s	n_f
3	3	63	2	42
3	4	79	4	77*
3	4	78	3	59
5	4	238*	2	130
5	3	182	2	122
5	3	188*	4	250
10	**		**	
10	4	1090	3	845
10	6	1663*	4	1144

* Converged to alternate local optimum
** Failed to converge in ten sweeps

TABLE XIII

OTHER METHODS APPLIED TO RANDOM MATRIX PROBLEM

Powell's Method		Rosenbrock's Method	
n	n_f	n	n_f
3	61		
3	61		
5	104	5	465
5	103	5	466
		5	388
10	329	10	1210
10	369	10	1258
		10	1295

$$S = \begin{pmatrix} .835 & -.549 & -.006 & -.024 & -.041 \\ .387 & .594 & -.705 & -.006 & -.032 \\ .391 & .586 & .709 & -.003 & .028 \\ .026 & -.023 & -.001 & .957 & .287 \\ .004 & -.053 & .003 & -.288 & .956 \end{pmatrix},$$

show a strong interaction among the first three variables (the term coefficients) and a significant interaction between the last two variables (the exponential decay rates). With this information, similar problems could be modified to make then easier to solve.

TABLE XIV

DATA FOR FIRST CURVE FITTING PROBLEM

t	y	t	y	t	y
0	.844	110	.718	220	.478
10	.908	120	.685	230	.467
20	.932	130	.658	240	.457
30	.936	140	.628	250	.448
40	.925	150	.603	260	.438
50	.908	160	.580	270	.431
60	.881	170	.558	280	.424
70	.850	180	.538	290	.420
80	.818	190	.522	300	.414
90	.784	200	.506	310	.411
100	.751	210	.490	320	.406

The second model is

$$f(t,x) = x_1 \exp(-x_5 t) + x_2 \exp(-x_6 (t-x_9)^2) + x_3 \exp(-x_7 (t-x_{10})^2) + x_4 \exp(-x_8 (t-x_{11})^2).$$

TABLE XV

FIRST CURVE FITTING PROBLEM WITH COLUMN ORDERING AND NO SORTING

n_s	n_ℓ	n_f	x_1	x_2	x_3	x_4	x_5
		f	C_{11}	C_{22}	C_{33}	C_{44}	C_{55}
0	0	0	0.5000000	1.5000000	-1.0000000	0.0100000	0.0200000
0.8790263D 00			0.0	0.0	0.0	0.0	0.0
1	20	60	0.3312236	1.5465290	-1.0077168	0.0103918	0.0210042
0.3400754D-02			12.867	6.292	0.130	112364.449	1568.257
2	40	120	0.3698178	1.7070741	-1.2239984	0.0121998	0.0228962
0.1383594D-03			9.897	3.310	0.022	87857.676	1471.688
3	60	178	0.3731657	1.7099966	-1.2367073	0.0123504	0.0232258
0.5841691D-04			14.164	2.240	-0.000	81342.757	1421.494
4	80	237	0.3736217	1.7564485	-1.2840194	0.0124693	0.0229690
0.5684712D-04			14.174	2.239	0.000	82773.009	1458.852
5	100	303	0.3736929	1.7661341	-1.2936902	0.0124886	0.0229133
0.5624552D-04			14.266	2.244	0.001	84516.648	1499.773
6	120	369	0.3738730	1.7873819	-1.3152240	0.0125383	0.0228024
0.5588268D-04			14.314	2.247	0.001	85882.093	1529.935
7	140	436	0.3744450	1.8357859	-1.3640271	0.0126546	0.0225673
0.5520508D-04			14.382	2.251	0.000	88241.409	1580.912
8	160	498	0.3745909	1.8406227	-1.3688557	0.0126668	0.0225429
0.5509555D-04			14.432	2.254	0.001	89027.257	1597.090
9	180	563	0.3748755	1.8782834	-1.4067361	0.0127468	0.0223665
0.5479609D-04			14.475	2.257	0.000	91371.754	1642.224
10	200	629	0.3750689	1.8965667	-1.4251567	0.0127865	0.0222873
0.5471430D-04			14.525	2.259	0.000	92664.314	1666.250
11	220	698	0.3751596	1.9056476	-1.4343367	0.0128061	0.0222497
0.5468872D-04			14.555	2.260	0.001	93258.667	1676.962
12	240	764	0.3752525	1.9178022	-1.4465247	0.0128306	0.0221968
0.5466214D-04			14.580	2.263	0.000	94081.293	1691.388
13	260	831	0.3752556	1.9184732	-1.4471859	0.0128318	0.0221935
0.5466131D-04			14.577	2.260	0.003	94132.650	1692.165
14	280	900	0.3753977	1.9342400	-1.4630712	0.0128643	0.0221292
0.5464907D-04			14.593	2.263	0.000	95104.157	1708.936
15	300	957	0.3754100	1.9358435	-1.4646837	0.0128675	0.0221227
0.5464895D-04			14.606	2.263	0.000	95293.529	1711.987
16	320	1021	0.3754101	1.9358470	-1.4646872	0.0128675	0.0221227
0.5464895D-04			14.190	2.313	0.000	132957.758	4523.271

The data is listed in Table XVI and the results are shown in Table XVII. This case provides a practical problem of many variables.

TABLE XVI

DATA FOR SECOND CURVE FITTING PROBLEM

t	y	t	y	t	y
0.	1.366	2.2	.694	4.4	.672
0.1	1.191	2.3	.644	4.5	.708
0.2	1.112	2.4	.624	4.6	.633
0.3	1.013	2.5	.661	4.7	.668
0.4	.991	2.6	.612	4.8	.645
0.5	.885	2.7	.558	4.9	.632
0.6	.831	2.8	.533	5.0	.591
0.7	.847	2.9	.495	5.1	.559
0.8	.786	3.0	.500	5.2	.597
0.9	.725	3.1	.423	5.3	.625
1.0	.746	3.2	.395	5.4	.739
1.1	.679	3.3	.375	5.5	.710
1.2	.608	3.4	.372	5.6	.729
1.3	.655	3.5	.391	5.7	.720
1.4	.616	3.6	.396	5.8	.636
1.5	.606	3.7	.405	5.9	.581
1.6	.602	3.8	.428	6.0	.428
1.7	.626	3.9	.429	6.1	.292
1.8	.651	4.0	.523	6.2	.162
1.9	.724	4.1	.562	6.3	.098
2.0	.649	4.2	.607	6.4	.054
2.1	.649	4.3	.653		

Comparison of Orderings

Table XVIII summarizes the results which compare the orderings. The first observation is that none of the orderings cause complete failure. The column ordering with descending sort causes slowest

TABLE XVII

SECOND CURVE FITTING PROBLEM WITH
COLUMN ORDERING AND NO SORTING

n_s	n_ℓ	n_f	x_1	x_2	x_3	x_4	x_5
	f		C_{11}	C_{22}	C_{33}	C_{44}	C_{55}
0	0	0	1.3000000	0.6500000	0.6500000	0.7000000	0.6000000
0.2093420D 01			0.0	0.0	0.0	0.0	0.0
1	110	333	1.1899570	0.3048836	0.5202528	0.4873668	0.5016136
0.1129138D 00			30.375	13.965	21.224	13.842	64.183
2	220	662	1.2467814	0.3315676	0.5831046	0.5462798	0.5618960
0.5767092D-01			8.001	12.060	19.871	13.911	71.153
3	330	992	1.2935216	0.4045601	0.6209244	0.5646968	0.6917888
0.4184402D-01			3.330	11.763	27.474	14.839	53.943
4	440	1329	1.3073287	0.4252806	0.6309330	0.5904110	0.7380161
0.4026296D-01			2.960	11.843	27.819	15.296	50.463
5	550	1671	1.3100218	0.4313626	0.6336042	0.5989897	0.7535135
0.4013820D-01			2.975	11.788	28.016	15.466	49.527
6	660	2014	1.3099772	0.4315537	0.6336617	0.5994305	0.7541830
0.4013774D-01			2.974	11.779	28.045	15.482	49.435

TABLE XVII (Continued)

x_6 C_{66}	x_7 C_{77}	x_8 C_{88}	x_8 C_{99}	x_{10} $C_{10,10}$	x_{11} $C_{11,11}$
3.0000000 0.0	5.0000000 0.0	7.0000000 0.0	2.0000000 0.0	4.5000000 0.0	5.5000000 0.0
2.0982742 0.326	1.1613520 0.212	3.6045267 0.154	2.4158393 3.437	4.5380937 2.366	5.6443125 9.691
1.5358380 0.183	1.2166643 0.095	5.4507358 0.022	2.4307116 4.170	4.5942314 2.450	5.6816599 15.257
1.0890311 0.216	1.1921386 0.119	5.1792022 0.008	2.3938802 4.642	4.5834035 1.630	5.6835096 16.435
0.9551153 0.163	1.3115275 0.079	4.9131033 0.008	2.3963891 4.859	4.5715682 1.206	5.6782114 17.126
0.9067318 0.151	1.3636656 0.065	4.8258524 0.007	2.3988278 4.920	4.5690447 1.147	5.6754883 17.346
0.9042891 0.147	1.3658118 0.056	4.8236989 0.006	2.3986850 4.923	4.5688746 1.185	5.6753415 17.403

convergence. A possible reason is that the column ordering chooses the first indices more frequently at the beginning of the sweep and the latter indices more at the end. With the descending sort the directions with greater curvature ("across the valley") are chosen many times before the directions with lesser curvature ("down the valley"). For a curved valley, optimizing "across the valley" too accurately reduces the domain of lower function values. As a result, it is difficult to find a better point "down the valley." This problem might be relieved by changing the scale.

At the other extreme, the column ordering with ascending sort and the sequential ordering with descending sort causes fast convergence on the functions tried. More exhaustive testing would be required to draw further conclusions.

TABLE XVIII

COMPARISON OF ORDERINGS

		$n_f/\log_{10} f$					
n	n_s	COL NO SORT	COL SORT	COL REVERSE	COL NO SORT	COL SORT	SEQ REVERSE
Eight variable quadratic function							
8	3	504/-18	517/-20	536/-3	557/-9	543/-9	528/-18
	4	645/-28	648/-29	693/-27	702/-30	690/-25	669/-25
Powell's quartic function							
4	6	222/-7	223/-9	228/-6	232/-7	233/-9	225/-9
Random matrix function							
3	4	84/-14	84/-14	84/-14	83/-18	83/-18	82/-18
3	4	79/-7			77/-6*		
3	4	78/-10			80/-15		
5	4	238/-10*			247/-27		
5	4	236/-27			237/-26		
5	4	241/-26*			250/-5		
10	5	1400/-1*			1440/-2*		
10	5	1331/-21			1310/-26		
10	5	1389/-4*			1391/-16		

* Converging to alternate optimum

CHAPTER VII

SUMMARY AND CONCLUSIONS

The objective of this research was to develop a new direct search method for unconstrained function optimization. The method is based on fitting a quadratic model and moving towards the optimum of the model along approximate eigenvectors of the curvature matrix. The operation of the method involves fitting the model, improving the eigenvectors and searching for a location with improved function value. The three processes are accomplished efficiently by an organization based on the Jacobi method for finding eigenvalues of a matrix. At the same time, the basic method allows flexibility in implementing several operations.

The calculations used for fitting the model are straightforward, but computational roundoff error can be a problem. Using the model to predict function values, some types of error can be estimated. The step size (distance between sample locations) directly controls the effects of the error on the calculations. Therefore, an analysis of the error has been completed which yields limits on the step size necessary to maintain sufficient accuracy.

A new analysis of the Jacobi method suggests sorting the diagonal of the working matrix. The same effect can be produced by modifying the order of choosing pairs. The analysis includes a proof that the modification (of the matrix or the ordering) does not invalidate the existing proofs regarding convergence of the Jacobi method. Testing

indicates a slight improvement in speed of convergence of the eigenvalues with the modified ordering. For optimization, however, the testing did not show a significant effect on convergence to the optimum.

The rate of confergence of the new method to the minimum of a positive definite quadratic function has been analyzed. The result shows that the rate depends on the error measure used for the Jacobi method. Under conditions where the Jacobi method converges quadratically, the distance to the optimum converges to zero at least quadratically.

To verify the operation of the new method, one version was programmed and tested on several problems. The algorithm converged at a reasonable rate in almost all cases. In fact, on problems with a small number of variables, the rate of convergence was approximately the same as some of the best previous methods. This result is encouraging when it is considered that the algorithm used was the first version of the method.

The new method is valuable due to the availability of the model of the function including the eigenvectors of the curvature matrix. One use fot the model is to evaluate "sensitivity" coefficients. The eigenvectors provide information of the interaction (correlation) of the independent variables. In many problems part of the model is known beforehand and may be set initially, obviously shortening the optimization. An important example is the solution of a constrained problem by repeated unconstrained optimization with a variable penalty function added. In that case, the change in the model due to changes in the penalty function might even be calculated before restarting the optimization. Another example of the use of the model is the optimization of

a large number of variables in groups, combining the models for the groups in a final complete optimization.

A possible extension of the new method is the direct inclusion of constraints. The problem of constrainted optimization is important, due to its natural occurrence in many situations and its inherent difficulty. With the new method, the situation is reduced to using the constrained optimum of the model, which is a known tractable function. Adding constraints in this way is possible because the location of the samples used to fit the model are not restricted. Another extension concerns problems where the gradient or curvature matrix can be obtained, including minimization of a sum of squares. In these cases additional information allows a significant reduction in the number of function evaluations required to determine the optimum, as shown by previously available methods.

SELECTED BIBLIOGRAPHY

Bard, Y.

1970 "Comparison of Gradient Methods for the Solution of Nonlinear Parameter Estimation Problems." SIAM J. Number. Anal. 7, 157-186.

1974 Nonlinear Parameter Estimation. New York: Academic Press.

Box, M. J.

1966 "A Comparison of Several Current Optimization Methods, and the Use of Transforms in Constrained Problems." Comp. J. 9, 66-77.

Box, M. J., D. Davies and W. H. Swann.

1969 "Non-Linear Optimization Techniques. ICI Monograph No. 5. London: Oliver and Boyd.

Brent, R. P.

1972 Algorithms for Minimization Without Derivatives. Englewood Cliffs, New Jersey: Prentice Hall.

Davidon, W. C.

1959 Variable Metric Method for Minimization. Washington: A. E. C. Research and Development Rep. ANL-5990 (Rev.).

Fletcher, R.

1965 "Function Minimization Without Evaluating Derivatives - A Review." Comp. J., 8, 33-41.

1969a Optimization. New York: Academic Press.

1969b "A Review of Methods for Unconstrained Optimization." Optimization. Ed. R. Fletcher. New York: Academic Press.

1970 "A New Approach to Variable Metric Algorithms." Comp. J., 13, 317-322.

Fletcher, R. and M. J. D. Powell.

1963 "A Rapidly Convergent Descent Method for Minimization." Comp. J., 6, 163-168.

Fletcher, R. and C. M. Reeves.

1964 "Function Minimization by Conjugate Gradients." Comp. J., 7, 149-154.

Forsythe, G. E. and P. Henrici.

1960 "The Cyclic Jacobi Method for Computing the Principle Values of a Complex Matrix." Trans. Amer. Math. Soc., 94, 1-23.

Gregory, R. T.

1953 "Computing Eigenvalues and Eigenvectors of a Symmetric Matrix on the Illiac." Math. Tab. Aids. Comp., 7, 215-220.

Hadamard, J.

1893 "Resolution d'une Question Relative aux Determinants." Bull. Sci. Math., Ser. 2, 17, Pt. 1, 240-246.

Hammerling, S. J.

1970 Latent Roots and Latent Vectors. Toronto: Univ. of Toronto Press.

Hansen, E. R.

1963 "On Cyclic Jacobi Methods." J. SIAM., 11, 448-459.

Henrici, P.

1958 "On the Speed of Convergence of Cyclic and Quasicyclic Jacobi Methods for Computing Eigenvalues of Hermitian Matrices." J. SIAM., 6, 144-162.

Himmelblau, D. M.

1972 "A Uniform Evaluation of Unconstrained Optimization Techniques." Numerical Methods for Nonlinear Optimization. Ed. F. A. Lootsma. New York: Academic Press.

Hooke, R. and T. A. Jeeves.

1961 "Direct Solution of Numerical and Statistical Problems." J. ACM., 8, 212-229.

Huang, H. Y.

1970 "Unified Approach to Quadratically Convergent Algorithms for Function Minimization." J. Optim. Theory and Appl., 5, 405-423.

Huang, H. Y. and A. V. Levy.

1970 "Numerical Experiments on Quadratically Convergent Algorithm for Function Minimization." J. Optim. Theory and Appl., 6, 269-282.

Jacobi, C. G. J.

1846 "Über ein Leichtes Verfahren, die in der Theorie der Säcularstörungen Vorkommenden Gleichungen Numerische Aufzulösen." J. Reine Angew Math., 30, 51-95.

Kowalik. J. and M. R. Osborne.

1968 Methods for Unconstrained Optimization Problems. New York: Elsevier.

Murray, W., ed.

1972 Numerical Methods for Unconstrained Optimization. New York: Academic Press.

Nelder, J. A. and R. Mead.

1965 "A Simplex Method for Function Minimization." Comp. J., 7, 308-313.

Osborne, M. R.

1972 "Some Aspects of Non-Linear Least Square Calculations." Numerical Methods for Nonlinear Optimization. Ed. F. A. Lootsma. New York: Academic Press.

Paley, R. E. A. C.

1933 "On Orthogonal Matrices." J. Math. Phys., 12, 311-320.

Polak, E.

1971 Computational Methods in Optimization: A Unified Approach. New York: Academic Press.

Powell, M. J. D.

1962 "An Iterative Method for Finding Stationary Values of a Function of Several Variables." Comp. J., 5, 147-151.

1964 "An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives." Comp. J., 7, 155-162.

Rosenbrock, H. H.

1960 "An Automatic Method for Finding the Greatest or Least Value of a Function." Comp. J., 175-184.

Sargent, R. W. H. and D. J. Sebastian.

1972 "Numerical Experience with Algorithms for Unconstrained Minimization." Numerical Methods for Nonlinear Optimization. Ed. F. A. Lootsma. New York: Academic Press.

Schönhage, A.

1961 "Zur Konvergenz des Jacobi-Verfahrens." Numer. Math., 3, 374-380.

Schwarz, H. R.

1973 Numerical Analysis of Symmetric Matrices. Englewood Cliffs, New Jersey: Prentice Hall.

Stewart, G. W.

1967 A Modification of Davidon's Minimization Method to Accept Difference Approximations of Derivatives. J. ACM., 72-83.

Swann, W. H.

1964 Report on the Development of a New Direct Search Method of Optimization. Edinburgh: ICI Ltd. Cent. Inst. Lab., Research Note 64/3.

1969 "A Survey of Non-linear Optimization Techniques." FEBS Letters, 2, Suppl. 39-55.

Wilkinson, J. H.

1962 "Note on the Quadratic Convergence of the Cyclic Jacobi Process." Numer. Math., 4, 296-300.

1965 The Algebraic Eigenvalue Problem. Oxford: Clarendon Press.

Zangwill, W.

1967 "Minimizing a Function Without Calculating Derivatives." Comp. J., 10, 293-296.

APPENDIX

LISTING OF COMPUTER PROGRAM

```
C     THE PURPOSE OF THIS PROGRAM IS TO OPTIMIZE AN UNCONSTRAINED
C     (SMOOTH) FUNCTION WITHOUT EXPLICIT GRADIENT.  IT WAS WRITTEN BY
C     FRED WITZ FOR PH.D. RESEARCH.  THE METHOD IS DESCRIBED AND THE
C     PROGRAM IS EXPLAINED IN THE RELATED DISSERTATION, OKLAHOMA STATE
C     UNIVERSITY, MAY, 1976.
C
C     NOTE THAT DOUBLE AND TRIPLE LETTERS ARE USED THROUGHOUT FOR
C     VECTORS AND MATRICES, RESPECTIVELY.  MOST COMMENTS USE THE SINGLE
C     LETTER IN THE MATHEMATICAL SENSE.
C
      IMPLICIT REAL*8 (A-H, O-Z)
C                                          COMMON BLOCK
      LOGICAL SORT, REV, COPY, PRINT, TRACE, BPUT, BETER, DONE
      LOGICAL FREE5, FREER
      COMMON /JOECOM/ ZERO, ONE, TWO, PI, FOUR, DEGRE, DIR
     +, SSS(15, 15), XXO(15), YO, BB(15), CC(15), ZZOPT(15), ZZ(15)
     +, XXBES(15), YBEST, ZBEST, SS(15), SMALS, YYC(15), YL, ZZL(15)
      COMMON /JOECOM/ TZO, TOLX, SMALX, SMALY
     +, TYLX, TYLY, TYLC, TZLR, TZLX, TZLF, TZL, TZLT
     +, TZUF1, TZUF2, TZU1, TZUFB, TZUB, TZUP, TZUT
      COMMON /JOECOM/ M, KRDR, KPRT, N, NSWEP, NSAMP, NPAIR
     +, ISWEP, ITALR, IPLAN, IFIT, ITRY, IBEST, IMOVE
     +, N3, K5, KR
      COMMON /JOECOM/ BETER, DONE, SORT, REV, COPY, PRINT, TRACE, BPUT
     +, FREE5, FREER
C                                          END COMMON
      DIMENSION AAA(15, 15)
      DIMENSION PPRM(19), KKPRM(3)
      EQUIVALENCE (PPRM(1), TZO), (KKPRM(1), N3)
      ABS(W) = DABS(W)
      SIGN(W, W1) = DSIGN(W, W1)
      SQRT(W) = DSQRT(W)
    1 FORMAT(1X)
    2 FORMAT(/3H Y=, G15.7, 36X, 3H X=, (T58, 5G15.7) )
C
C                                          INITIALIZATION
      KRDR = 5
      KPRT = 6
C
      ZERO = 0.0
      ONE = 1.0
      TWO = 2.0
      PI = 3.1415926535897900
      FOUR = 4.0
      DEGRE = 180.0D0/PI
      M = 15
C                                          DEFAULTS
      NSWEP = 25
      NSAMP = 1000
C                                          INITIAL STEP RATIO
      TZO = .1
C                                          TERMINATION ERROR IN X
      TOLX = 1.E-4
C                                          INSURE DX
      SMALX = 1.E-40
      SMALY = 1.E-60
      TYLX = 1.E-10
      TYLY = 1.E-10
      TYLC = .01
      TZLR = .1
```

```
      TZLX = 1E-10
C                                          RATE OF Z REDUCTION
      TZLF = .3
      TZL = .01
      TZLT=.1
C                                          RATE OF Z GROWTH
      TZUF1 = 1.
      TZUF2 = .5
      TZU1 = .5
      TZUFB = 2.
      TZUB = 10.
      TZUP = 2.
      TZUT = 1.
      N3 = 1
      K5 = 1
      KR = 0
C                                          CIJ TERM RATIO
      TZLC = .1
 200  CONTINUE
C                                          ZERO WORKSPACES FOR PRINT
      SMALS = 0.
      YL = 0.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C                                          READ PARAMETER CHANGES
 21   FORMAT(I1, 9L1)
      READ(5, 21) KSWEP, SORT, REV, COPY, PRINT, TRACE, BPUT
      IF(KSWEP .EQ. 0) GO TO 990
 22   FORMAT(//17H PARAMETER INPUT- / 1X, I1, 9L1)
      IF(COPY)WRITE(6, 22) KSWEP, SORT, REV, COPY, PRINT, TRACE, BPUT
 23   FORMAT(10I5)
      READ(5, 23) IPRM, IPRM2
 24   FORMAT(1X, 10I5)
      IF(COPY) WRITE(6, 24) IPRM, IPRM2
      IF(IPRM .GT. 0) NSWEP=IPRM
      IF(IPRM2 .GT. 0) NSAMP = IPRM2
 250  CONTINUE
 25   FORMAT(I5,E10.0)
      READ(5, 25) IPRM, PRM
 26   FORMAT(1X, I5, G10.3)
      IF(COPY)WRITE(6,26)IPRM, PRM
      IF(IPRM) 254, 256, 252
 252  CONTINUE
      PPRM(IPRM)=PRM
      GO TO 250
 254  CONTINUE
      KKPRM(IPRM)=PRM
      GO TO 250
 256  CONTINUE
C                                          CONVERT LOGICAL PARAMETERS
      FREE5 = .TRUE.
      IF(K5 .EQ. 0) FREE5= .FALSE.
      FREER = .FALSE.
      IF(KR .NE. 0) FREER = .TRUE.
C                                          PRINT PARAMETERS
 41   FORMAT(/10H LIMITS=   , 2I5)
      IF(COPY) WRITE(6, 41) NSWEP, NSAMP
 42   FORMAT(/12H PARAMETERS=, 100(/1X, 10G10.3) )
      IF(COPY) WRITE(6, 42) PPRM
 44   FORMAT(1X, 10I10)
      IF(COPY) WRITE(6, 44) KKPRM
```

```
      IF(COPY) WRITE(6, 1)
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C                                          INITIALIZE MODEL
      DO 368 I = 1, M
      DO 338 J = 1, M
      SSS(I, J) = ZERO
  338 CONTINUE
      SSS(I, I) = ONE
      XXO(I) = ONE
      BB(I) = ZERO
      CC(I) = ZERO
      ZZ(I) = ZERO
      ZZL(I) = ZERO
      ZZOPT(I) = ZERO
      YYC(I) = ZERO
  368 CONTINUE
C                                          SET DEFAULTS FOR INIT
C                                          + = MAXIMIZE, - = MINIMIZE
      DIR = -1.0
      N=2
C                                          INITIALIZE FUNCTION
      CALL INIT(XXO, ZZ, DIR, N)
C                                          INITIALIZE STEPSIZE, ETC.
      DO 458 I = 1, N
      XXBES(I) = XXO(I)
      IF(ZZ(I) .EQ. ZERO) ZZ(I) = TZO*XXO(I)
      IF(ABS(ZZ(I)) .LT. SMALX) ZZ(I) = ONE
  458 CONTINUE
      NPAIR = (N*(N - 1) )/2
      DIR = SIGN(ONE, DIR)
      SMALS = SQRT(ONE/N)
C                                          SAMPLE INITIAL POINT
      CALL EVAL(YO,XXO,N)
      YBEST = YO
      ZBEST = ZERO
C                                          INITIALIZE COUNTERS
      ISWEP = 0
      ITALR = 0
      IPLAN = 0
      IFIT = 0
      ITRY = 0
      IBEST = 0
      IMOVE = 0
C                                          INITIALIZE SWEEP
      IF(KSWEP - 2) 461, 462, 463
  461 CONTINUE
      CALL SWEP1(0, N)
      GO TO 469
  462 CONTINUE
      CALL SWEP2(0, N)
      GO TO 469
  463 CONTINUE
      CALL SWEP3(0, N)
  469 CONTINUE
      IF(.NOT. SORT) GO TO 479
   47 FORMAT(' DIAGONAL SORTED INCREASING')
      IF(.NOT. REV) WRITE(6, 47)
   48 FORMAT(' DIAGONAL SORTED DECREASING')
      IF(REV) WRITE(6, 48)
  479 CONTINUE
```

```
C                                                PRINT INITIAL POINT
      IF(PRINT .OR. TRACE) WRITE(6, 2) YO, (XXO(I), I = 1, N)
      IF(BPUT) CALL PUT(0)
      IF(TRACE) WRITE(6, 1)
      IF(PRINT) WRITE(6, 1)
C
C                                              ********************
C                                              MAIN ITERATION LOOP
C
      DO 599 ISWEP = 1, NSWEP
      DONE = .TRUE.
      IF(KSWEP - 2) 510, 520, 530
 510  CONTINUE
      CALL SWEP1(ISWEP, N)
      GO TO 590
 520  CONTINUE
      CALL SWEP2(ISWEP, N)
      GO TO 590
 530  CONTINUE
      CALL SWEP3(ISWEP, N)
 590  CONTINUE
      IF(DONE) GO TO 610
      IF(ITRY .GT. NSAMP) GO TO 600
 599  CONTINUE
C     END ISWEP
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C                                                EXITS
 60   FORMAT(' EXCEEDED MAXIMUM ITERATIONS')
 600  WRITE(6, 60)
      WRITE(6, 1)
      GO TO 900
 61   FORMAT(' $$$$$      STANDARD END')
 610  WRITE(6, 61)
C     GO TO 900
 900  CONTINUE
 91   FORMAT(T9, 'C=', (T12, 5G24.16) )
      WRITE(6, 91) (CC(I), I = 1, N)
      WRITE(6, 1)
 92   FORMAT(T9, 'S=', (T12, 5G24.16) )
      DO 928 I = 1, N
 928  WRITE(6, 92) (SSS(I, J), J = 1, N)
      WRITE(6, 1)
C                                                A = S C ST
C                                                THAT IS, THE SECOND PARTIALS
C                                                OF THE FUNCTION F
      CALL SCS(AAA, SSS, CC, N)
 94   FORMAT(T9, 'A=', (T12, 5G24.16) )
      DO 948 I = 1, N
 948  WRITE(KPRT, 94) (AAA(I, J), J = 1, N)
      WRITE(KPRT, 1)
      DO 968 I = 1, N
 968  WRITE(KPRT, 94) (AAA(I, J), J = 1, N)
      GO TO 200
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C                                                PROGRAM TERMINATION
 990  CONTINUE
 99   FORMAT('-***** END OF JOB')
      WRITE(6,99)
      STOP
      END
```

```
C
C
      SUBROUTINE SORTER(MAP)
C
C     THIS SUBROUTINE SORTS THE ELEMENTS OF CC. THE SORT IS DECREASING
C     FOR REV=FALSE AND INCREASING FOR REV=TRUE.  THE ELEMENTS OF CC
C     ARE NOT MOVED - THE VECTOR MAP LISTS THE INDICES OF CC IN ORDER.
C     THE METHOD IS A SIMPLE ADJACENT PAIR INTERCHANGE BUBBLE SORT.
C     IF SORT=FALSE, SORTING IS NOT DONE.
C
      IMPLICIT REAL*8 (A-H, O-Z)
      LOGICAL DONES
C                                          COMMON BLOCK
      LOGICAL SORT, REV, COPY, PRINT, TRACE, BPUT, BETER, DONE
      LOGICAL FREE5, FREER
      COMMON /JOECOM/ ZERO, ONE, TWO, PI, FOUR, DEGRE, DIR
     +, SSS(15, 15), XXO(15), YO, BB(15), CC(15), ZZOPT(15), ZZ(15)
     +, XXBES(15), YBEST, ZBEST, SS(15), SMALS, YYC(15), YL, ZZL(15)
      COMMON /JOECOM/ TZO, TOLX, SMALX, SMALY
     +, TYLX, TYLY, TYLC, TZLR, TZLX, TZLF, TZL, TZLT
     +, TZUF1, TZUF2, TZU1, TZUFB, TZUB, TZUP, TZUT
      COMMON /JOECOM/ M, KRDR, KPRT, N, NSWEP, NSAMP, NPAIR
     +, ISWEP, ITALR, IPLAN, IFIT, ITRY, IBEST, IMOVE
     +, N3, K5, KR
      COMMON /JOECOM/ BETER, DONE, SORT, REV, COPY, PRINT, TRACE, BPUT
     +, FREE5, FREER
C                                          END COMMON
      DIMENSION MAP(15)
      IF(.NOT. SORT) RETURN
      NU2 = N - 1
      NL2 = 1
      LEN = NU2
      DO 598 I = 1, LEN
      DONES = .TRUE.
      JU = NL2 + 1
      MU = MAP(JU)
      XU = CC(MU)
C                                          PAIRWISE DESCENDING
C                                          DO JL = NU2, NL2, -1
      DO 498 JL2 = NL2, NU2
      JL = NL2 + NU2 - JL2
      ML = MAP(JL)
      XL = CC(ML)
      IF(REV) GO TO 420
C                                          SORT INCREASING
      IF(XL .LE. XU) GO TO 439
      GO TO 430
 420  CONTINUE
C                                          SORT DECREASING
      IF(XL .GE. XU) GO TO 439
 430  CONTINUE
C                                          INTERCHANGE
      MAP(JL) = MU
      MAP(JU) = ML
      DONES = .FALSE.
      GO TO 469
 439  CONTINUE
      MU = ML
      XU = XL
 469  CONTINUE
```

```
      JU = JL
  498 CONTINUE
      IF(DONES) GO TO 900
      NL2 = NL2 + 1
  598 CONTINUE
  900 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE SCS(AAA, SSS, CC, N)
C
C     THIS SUBROUTINE CALCULATES THE MATRIX PRODUCT A = S*C*ST, WHERE
C     ST IS THE TRANSPOSE OF S, AND C IS A DIAGONAL MATRIX WITH THE
C     DIAGONAL STORED IN CC.
C
      IMPLICIT REAL*8 (A-H, O-Z)
      DIMENSION AAA(15, 15), SSS(15, 15), CC(15), WW(15)
      ZERO = 0.
C                                          A = S*C*ST
      DO 599 I = 1, N
C                                          W = S*C
      DO 369 J = 1, N
      WW(J) = SSS(I, J)*CC(J)
  369 CONTINUE
      DO 569 J = I, N
      W = ZERO
C                                          A = W*ST  OR  AT = S*WT
      DO 469 K = 1, N
      W = W + SSS(J, K)*WW(K)
  469 CONTINUE
      AAA(I, J) = W
      AAA(J, I) = W
  569 CONTINUE
  599 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE PUT(LEVEL)
C
C     THIS SUBROUTINE ALLOWS THE PRINTING OF ANY INFORMATION SO THAT THE
C     FORMAT CAN BE CHANGED FOR DIFFERENT PROBLEMS.  LEVEL INDICATES THE
C     CURRENT LOCATION IN THE ALGORITHM AS FOLLOWS. 1=INITIALIZATION
C     2=AFTER UNIVARIATE FIT, 3=AFTER BIVARIATE FIT AND ROTATION,
C     4=AFTER TAYLOR (GAUSS) STEP.
C
      IMPLICIT REAL*8 (A-H,O-Z)
C                                          COMMON BLOCK
      LOGICAL SORT, REV, COPY, PRINT, TRACE, BPUT, BETER, DONE
      LOGICAL FREE5, FREER
      COMMON /JOECOM/ ZERO, ONE, TWO, PI, FOUR, DEGRE, DIR
     +, SSS(15, 15), XXO(15), YO, BB(15), CC(15), ZZOPT(15), ZZ(15)
     +, XXBES(15), YBEST, ZBEST, SS(15), SMALS, YYC(15), YL, ZZL(15)
      COMMON /JOECOM/ TZO, TOLX, SMALX, SMALY
     +, TYLX, TYLY, TYLC, TZLR, TZLX, TZLF, TZL, TZLT
     +, TZUF1, TZUF2, TZU1, TZUFB, TZUB, TZUP, TZUT
      COMMON /JOECOM/ M, KRDR, KPRT, N, NSWEP, NSAMP, NPAIR
     +, ISWEP, ITALR, IPLAN, IFIT, ITRY, IBEST, IMOVE
     +, N3, K5, KR
```

```
      COMMON /JOECOM/ BETER, DONE, SORT, REV, COPY, PRINT, TRACE, BPUT
     +, FREE5, FREER
C                                          END COMMON
      IF(LEVEL.EQ. 1) RETURN
      IF(LEVEL.EQ. 2) RETURN
      WRITE(6, 31) ISWEP, IFIT, ITRY, (XXO(K), K = 1, N)
      WRITE(6, 32) YO, (CC(K), K = 1, N)
 31   FORMAT(/1X, 3I5, (1X,  5G20.10) )
 32   FORMAT(1H  , E15.7, (1X,  5G20.10) )
      RETURN
      END
C
C
      SUBROUTINE SWEP1(ISWEP, N)
C
C     THIS SUBROUTINE CHOOSES ALL SUPER-DIAGONAL ELEMENTS (PAIRS OF
C     INDICES) IN NATURAL COLUMN ORDER.
C     THE INDICES ARE THEN PERMUTED ACCORDING TO MAP TO SIMULATE
C     OPERATIONS OF C WITH THE DIAGONAL SORTED.  PLANE IS CALLED FOR
C     EACH (PERMUTED) PAIR OF INDICES.  SORTER IS CALLED BEFORE EACH
C     COMPLETE SWEEP TO UPDATE THE SORTING OF THE DIAGONAL OF C.
C
      IMPLICIT REAL*8 (A-H, O-Z)
      DIMENSION MAP(20)
      IF(ISWEP .GT. 0) GO TO 200
 10   FORMAT(1HO , 'COLUMN ORDERING')
      WRITE(6,10)
      DO 168 IK = 1, N
      MAP(IK) = IK
 168  CONTINUE
      RETURN
 200  CONTINUE
      CALL SORTER(MAP)
      DO 598 JK2 = 2, N
      JM1 = JK2 - 1
      JK = MAP(JK2)
      DO 388 IK2 = 1, JM1
      IK = MAP(IK2)
      CALL PLANE(IK, JK)
 388  CONTINUE
 598  CONTINUE
      CALL TALOR
      RETURN
      END
C
C
      SUBROUTINE SWEP2(ISWEP, N)
C
C     THIS SUBROUTINE CHOOSES ALL SUPER-DIAGONAL ELEMENTS (PAIRS OF
C     INDICES) IN A DIAGONAL ORDER WHICH FAVORS TWO SEQUENTIAL
C     OCCURANCES OF EACH INDEX.
C     THE INDICES ARE THEN PERMUTED ACCORDING TO MAP TO SIMULATE
C     OPERATIONS OF C WITH THE DIAGONAL SORTED.  PLANE IS CALLED FOR
C     EACH (PERMUTED) PAIR OF INDICES.  SORTER IS CALLED BEFORE EACH
C     COMPLETE SWEEP TO UPDATE THE SORTING OF THE DIAGONAL OF C.
C
      IMPLICIT REAL*8 (A-H, O-Z)
      DIMENSION MAP(20)
      IF(ISWEP .GT. 0)GO TO 200
 10   FORMAT(1HO , 'SEQ2 ORDERING, FAVOR EACH INDEX TWICE')
```

```
      WRITE(6,10)
C                                          INITIALIZE
      NO2 = N/2
      DO 159 I = 1, N
  159 MAP(I) = I
      RETURN
C
  200 CONTINUE
      CALL SORTER(MAP)
C                                          MAIN LOOP
      DO 499 II2 = 1, NO2
      JJ = 1
      JJSAV = JJ
      II = 1
      DO 489 JJ2 = 1, N
      IISAV = II
      II = JJ
      JJ = II + II2
      IF(JJ .GT. N) JJ = JJ - N
      IF(JJ .NE. IISAV) GO TO 310
      IF(N .NE. 2) GO TO 480
  310 CONTINUE
C                                          FIT AND OPTIMIZE PLANE
      IK = MAP(II)
      JK = MAP(JJ)
      CALL PLANE(IK, JK)
  480 CONTINUE
C                                          END JJ
      IF(JJ .NE. JJSAV) GO TO 489
      JJ = JJ + 1
      JJSAV = JJ
  489 CONTINUE
  498 CONTINUE
C                                          END II
  499 CONTINUE
C                                          TAYLOR STEP
      CALL TALOR
      RETURN
      END
C
C
      SUBROUTINE SWEP3(ISWEP, N)
C
C     THIS SUBROUTINE ALLOWS FOR A THIRD ORDERING FOR CHOOSING PAIRS.
C
      IMPLICIT REAL*8 (A-H,C-Z)
      DIMENSION MAP(15)
      STOP
      END
C
C
      SUBROUTINE PLANE(IK,JK)
C
C     THIS SUBROUTINE CONTROLS THE BIVARIATE FIT AND THE ROTATION OF THE
C     DIRECTION VECTORS IN THE PLANE OF S(I) AND S(J)
C     IK AND JK SPECIFY THE TWO INDICES REFERRED TO IN COMMENTS AS I
C     AND J.
C
      IMPLICIT REAL*8 (A-H, O-Z)
C                                          COMMON BLOCK
```

```
      LOGICAL SORT, REV, COPY, PRINT, TRACE, BPUT, BETER, DONE
      LOGICAL FREE5, FREER
      COMMON /JOECOM/ ZERO, ONE, TWO, PI, FOUR, DEGRE, DIR
     +, SSS(15, 15), XX0(15), Y0, BB(15), CC(15), ZZOPT(15), ZZ(15)
     +, XXBES(15), YBEST, ZBEST, SS(15), SMALS, YYC(15), YL, ZZL(15)
      COMMON /JOECOM/ TZ0, TOLX, SMALX, SMALY
     +, TYLX, TYLY, TYLC, TZLR, TZLX, TZLF, TZL, TZLT
     +, TZLF1, TZUF2, TZU1, TZUFB, TZUB, TZUP, TZUT
      COMMON /JOECOM/ M, KRDR, KPRT, N, NSWEP, NSAMP, NPAIR
     +, ISWEP, ITALR, IPLAN, IFIT, ITRY, IBEST, IMOVE
     +, N3, K5, KR
      COMMON /JOECOM/ BETER, DONE, SORT, REV, COPY, PRINT, TRACE, BPUT
     +, FREE5, FREER
C                                          END COMMON
      ABS(W) = DABS(W)
      SIGN(W, W1) = DSIGN(W, W1)
      SQRT(W) = DSQRT(W)
      ATAN2(W, W1) = DATAN2(W, W1)
 1    FORMAT(' ')
 71   FORMAT(' DIRECTION', I3, '       Z=', G15.7, ' C=', G15.7, ' S=',
     +       (T58, 5G15.7) )
 72   FORMAT(' CROSS RUN ', I4, '     R=', G15.7, ' C=', G15.7, ' S=',
     +       (T58, 5G15.7) )
      IPLAN = IPLAN + 1
      YLX = ZERO
      YLC = ZERO
      DO 239 I = 1, N
C                                          SUM ABS G*X
      W = ZERO
      DO 229 J = 1, N
      W = W + SSS(I, J)*BB(J)
 229  CONTINUE
      YLX = YLX + ABS(W*XX0(I) )
C                                          MAX CZZ
      IF(YYC(I) .GT. YLC) YLC = YYC(I)
 239  CONTINUE
      YL = TYLY*ABS(Y0) + TYLX*YLX + TYLC*YLC
C
C                                          UNIVARIATE FIT IN DIRECTION S(I)
      CI = CC(IK)
      ZI = ZZ(IK)
      ZLI = ZZL(IK)
      DO 339 I = 1, N
 339  SS(I) = SSS(I, IK)
      ZBEST = ZERO
      CALL FIT(ZI, BI, CI, ZLI, ZUI, ZLFI, ZUFI)
      IF(TRACE) WRITE(6, 71) IK, ZI, CI, (SS(K), K = 1, N)
      IF(BPUT) CALL PUT(1)
      IF(TRACE) WRITE(6, 1)
      YYC(IK) = ABS(CI*ZI*ZI)
      ZZL(IK) = ZLI
      ZBEST = ZBEST
C                                          MOVE TO BEST POINT FOUND
      IF(Y0 .EQ. YBEST) GO TO 360
      IMOVE = IMOVE + 1
      DO 359 I = 1, N
 359  XX0(I) = XXBES(I)
      Y0 = YBEST
      BI = BI + CI*ZBEST
      ZI = ZI - ZBEST
```

```
      IF(ABS(ZI) .LT. ZLFI) ZI = SIGN(ZLFI, ZI)
      ZBESI = ZERO
  360 CONTINUE
C
C                                          UNIVARIATE FIT IN DIRECTION S(J)
      CJ = CC(JK)
      ZJ = ZZ(JK)
      ZLJ = ZZL(JK)
      DO 369 I = 1, N
  369 SS(I) = SSS(I, JK)
      ZBEST = ZERO
      CALL FIT(ZJ, BJ, CJ, ZLJ, ZUJ, ZLFJ, ZUFJ)
       IF(TRACE) WRITE(6, 71) JK, ZJ, CJ, (SS(K), K = 1, N)
      IF(BPUT) CALL PUT(1)
      IF(TRACE) WRITE(6, 1)
      YYC(JK) = ABS(CJ*ZJ*ZJ)
      ZZL(JK) = ZLJ
      ZBESJ = ZBEST
C
C                                          BIVARIATE FIT
      Z = SQRT(ZI*ZI + ZJ*ZJ)
      WI = ZI/Z
      WJ = ZJ/Z
C                                          S = ZI*SI + ZJ*SJ
      DO 389 I = 1, N
  389 SS(I) = WI*SSS(I, IK) + WJ*SS(I)
      C = CI*WI*WI + CJ*WJ*WJ
      ZBEST = ZERO
      CALL TRY(Z, DY, B)
      CIJ = (DY - ZI*(BI + ZI*CI/TWO) - ZJ*(BJ + ZJ*CJ/TWO) )/(ZI*ZJ)
C                                          LIMIT CIJ
      IF(FREER) GO TO 400
      W = SQRT(ABS(CI*CJ) )
      IF(ABS(CIJ) .GT. W) CIJ = SIGN(W, CIJ)
  400 CONTINUE
C                                          MOVE TO BEST POINT FOUND
      IF(ZBEST .EQ. ZERO) GO TO 420
      ZBESI = ZI
      ZBESJ = ZJ
  420 CONTINUE
      IF(YO .EQ. YBEST) GO TO 430
      IMOVE = IMOVE + 1
      DO 429 I = 1, N
  429 XXO(I) = XXBES(I)
      YO = YBEST
      BI = BI + CI *ZBESI + CIJ*ZBESJ
      BJ = BJ + CIJ*ZBESI + CJ *ZBESJ
  430 CONTINUE
C
C                                          ROTATE IN PLANE
      W1 = CJ - CI
      W2 = SQRT(FOUR*CIJ*CIJ + W1*W1)
      WCOS2 = (W2 + ABS(W1) )/(TWO*W2)
      WCOS = SQRT(WCOS2)
      WSIN = SIGN(ONE, W1)*CIJ/(WCOS*W2)
      WSIN2 = WSIN*WSIN
C                                          R = COS, SIN
C                                             -SIN, COS
C                                          RT = R TRANSPOSE
C                                          C = RT*C*R
```

```
C                                                  (C STORED DIAGONAL ONLY)
      W = TWO*WSIN*WCOS*CIJ
      WI = CI
      CI = WCOS2*WI + WSIN2*CJ - W
      CJ = WSIN2*WI + WCOS2*CJ + W
      CC(IK) = CI
      CC(JK) = CJ
      W = ATAN2(WSIN, WCOS)*DEGRE
      IF(TRACE) WRITE(6, 72) IPLAN, W, CIJ, (SS(K), K = 1, N)
C                                                  B = RT*B
      WI = BI
      BI = WCOS*WI - WSIN*BJ
      BJ = WSIN*WI + WCOS*BJ
      BB(IK) = BI
      BB(JK) = BJ
C                                                  S = S*R CR ST = RT*ST
      DO 459 I = 1, N
      WI = SSS(I,IK)
      WJ = SSS(I,JK)
      SSS(I, IK) = WCOS*WI - WSIN*WJ
      SSS(I, JK) = WSIN*WI + WCOS*WJ
  459 CONTINUE
C
C                                                  CALCULATE NEW OPTIMUM STEP SIZE
C
C                                                  IF C IS NONZERO AND OF CORRECT
C                                                  SIGN AND IF B/C IS LESS THAN ZU
C                                                  Z = -B/C , OTHERWISE
C                                                  Z = ZU WITH SIGN FOR DOWNHILL
      ZI = SIGN(ZUI,BI*DIR)
      IF(ABS(BI) .LT. -DIR*CI*ABS(ZI) ) ZI = -BI/CI
      ZZOPT(IK) = ZI
      IF(ABS(ZI) .LT. ZLI) ZI = SIGN(ZLI, ZI)
C
      ZJ = SIGN(ZUJ, BJ*DIR)
      IF(ABS(BJ) .LT. -DIR*CJ*ABS(ZJ) ) ZJ = -BJ/CJ
      ZZOPT(JK) = ZJ
      IF(ABS(ZJ) .LT. ZLJ) ZJ = SIGN(ZLJ, ZJ)
C
      ZZ(IK) = ZI
      ZZ(JK) = ZJ
      IF(TRACE) WRITE(6, 71 ) IK, ZI, CI, (SSS(K, IK), K = 1, N)
      IF(TRACE) WRITE(6, 71 ) JK, ZJ, CJ, (SSS(K, JK), K = 1, N)
      IF(PRINT)WRITE(6,2)YO,ISWEP,ITRY,IBEST,CIJ,(XXO(K),K=1,N)
    2 FORMAT(' Y=',G15.7,3X,3I5,' C=',G15.7, ' X=',(T58,5G15.7) )
      IF(BPUT) CALL PUT(2)
      IF(TRACE) WRITE(6, 1)
      RETURN
      END
C
C
      SUBROUTINE FIT(Z,B,C,ZL,ZU,ZLF,ZUF)
C
C     THIS SUBROUTINE PERFORMS THE UNIVARIATE FIT WHICH ALSO SEARCHES
C     FOR THE OPTIMUM
C
      IMPLICIT REAL*8 (A-H, O-Z)
C                                                  COMMON BLOCK
      LOGICAL SORT, REV, COPY, PRINT, TRACE, BPLT, BETER, DONE
      LOGICAL FREE5, FREER
```

```
      COMMON /JOECOM/ ZERO, ONE, TWO, PI, FOUR, DEGRE, DIR
     +, SSS(15, 15), XXO(15), YO, BB(15), CC(15), ZZOPT(15), ZZ(15)
     +, XXBES(15), YBEST, ZBEST, SS(15), SMALS, YYC(15), YL, ZZL(15)
      COMMON /JOECOM/ TZO, TOLX, SMALX, SMALY
     +, TYLX, TYLY, TYLC, TZLR, TZLX, TZLF, TZL, TZLT
     +, TZUF1, TZUF2, TZU1, TZUFB, TZUB, TZUP, TZUT
      COMMON /JOECOM/ M, KRDR, KPRT, N, NSWEP, NSAMP, NPAIR
     +, ISWEP, ITALR, IPLAN, IFIT, ITRY, IBEST, IMOVE
     +, N3, K5, KR
      COMMON /JOECOM/ BETER, DONE, SORT, REV, COPY, PRINT, TRACE, BPUT
     +, FREE5, FREER
C                                          END COMMON
      ABS(W) = DABS(W)
      SIGN(W, W1) = DSIGN(W, W1)
      SQRT(W) = DSQRT(W)
 81   FORMAT(T58,5G15.7)
      IFIT = IFIT + 1
      BETER = .FALSE.
      I3 = 0
C                                          FIND LIMIT TO INSURE X CHANGES
C                                          ZLX = MAX(X(I)/S(I) )
      ZLX = SMALX
      DO 19 I = 1, N
      S = ABS(SS(I) )
      IF(S .LT. SMALS) GO TO 19
      X = ABS(XXO(I) )
      IF(ZLX*S .LT. X) ZLX = X/S
 19   CONTINUE
      ZLX = TZLX*ZLX
C                                          USE FRACTION OF PREVIOUS ZLC
C                                          UNTIL C IS RELIABLE
      ZLC = TZLR*ZL
C                                          PRELIMIT
      IF(ABS(Z) .LT. ZLX) Z = SIGN(ZLX, Z)
C
C                                          FIRST STEP
 20   CONTINUE
      Z1 = Z
      CALL TRY(Z1, DY1, B1)
      B = B1 - C*Z1/TWO
C                                          SET LIMITS
      ZA = ABS(Z)
      ZL1  = TZL  *ZA + ZLX
      ZL   =  ZL1     + ZLC
      ZLF1 = TZLF *ZA + ZLX
      ZLF  =  ZLF1    + ZLC
      ZUF  = TZUF1*ZA + TZUFB*ABS(ZBEST)
      IF(TRACE)WRITE(6,81)ZLC,ZLF1,ZLF,Z1,ZUF
     +, ZLX, ZL1, ZL, Z, ZU
C                                          IF C IS NON-ZERO AND OF CORRECT
C                                          SIGN AND IF B/C IS LESS THAN ZUF
C                                          Z = -B/C , ELSE
C                                          Z = ZUF WITH SIGN FOR DOWNHILL
      Z = SIGN(ZUF, DIR*B)
      IF(ABS(B) .LT. -DIR*C*ZUF) Z = -B/C
C                                          INSURE NEW STEP IS AT LEAST ZLF
C                                          FROM ZERO AND FROM Z1
      IF(ABS(Z - ZBEST) .GT. ZLF) GO TO 40
      IF(ZA .LT. TWO*ZLF) Z = ZBEST + DIR*B1
      Z = ZBEST + SIGN(ZLF, Z - ZBEST)
```

```
C
C                                          SECOND STEP
 40   CONTINUE
      Z2 = Z
      CALL TRY(Z2, DY2, B2)
      C = TWO*(B2 - B1)/(Z2 - Z1)
      B = B2 - C*Z2/TWO
      ZA = ABS(Z)
      ZUF = TZUF2*ZA + TZUFB*ABS(ZBEST)
      ZU1  = TZU1 *ZA
      ZU = ZU1 + TZUB*ABS(ZBEST)
C                                        IF C IS NONZERO AND OF CORRECT
C                                        SIGN AND IF B/C IS LESS THAN ZU
C                                        Z = -B/C , OTHERWISE
C                                        Z = ZU WITH SIGN FOR DOWNHILL
      Z = SIGN(ZU, DIR*B)
      IF(ABS(B) .LT. -DIR*C*ZU ) Z = -B/C
      ZA = ABS(Z)
      IF(ZA .LT. SIGN(TWO,Z)*Z1) GO TO 50
      IF(ZA .LT. SIGN(TWO,Z)*Z2) GO TO 50
      IF(I3 .GT. N3) GO TO 50
      I3 = I3 + 1
      IF(TRACE)WRITE(6,81)ZLC,ZLF1,ZLF,Z1,ZUF
     +, ZLX, ZL1, ZL, Z, ZU
      IF(ZA .GT. ZUF) Z = SIGN(ZUF,Z)
      IF(ZA .LT. ZLF)Z = SIGN(ZLF, Z)
      GO TO 52
 50   CONTINUE
C                                        IF C IS NONZERO AND OF CORRECT
C                                        SIGN AND SQRT(ABS(YL/C)) IS LESS
C                                        THAN OR EQUAL ZU THEN
C                                        Z = SQRT(ABS(YL/C) )
C                                        OTHERWISE, ZLC = ZU
      ZLC = ZU
      IF(YL .LT. -DIR*C*ZLC*ZLC) ZLC = SQRT(ABS(YL/C))
      ZL = ZL1 + ZLC
C
C                                        TEST FOR COMPLETION, I.E.
C                                        BETTER POINT FOUND OR TOO MANY
C                                        REPETITIONS
      IF(BETER) GO TO 70
      IF(ABS(Z1) .LT. TWO*ZL) GO TO 60
      IF(ZA .LT. TZLF*ZA + ZLX + ZLC) GO TO 60
      IF(I3 .GE. N3) GO TO 60
      I3 = I3 + 1
C
C                                        REPEAT
      ZLF = ZLF1 + ZLC
      IF(TRACE)WRITE(6,81)ZLC,ZLF1,ZLF,Z1,ZUF
     +, ZLX, ZL1, ZL, Z, ZU
      IF(ZA .LT. ZL) Z = SIGN(ZL, Z)
C                                        IF NEW ESTIMATE IS EXTREME,
C                                        REPEAT BOTH STEPS
      IF(ZA .LT. ZLF) GO TO 20
      IF(ZA .GT. ZUF) GO TO 20
C                                        OTHERWISE, REPEAT SECOND STEP
 52   CONTINUE
      IF(DIR*(DY1 - DY2) .GT. ZERO) GO TO 58
      DY1 = DY2
      B1 = B2
```

```
      Z1 = Z2
   58 CONTINUE
      GO TO 40
C
C                                     EXIT SECTION
C
C
C                                     FAILED TO FIND BETTER POINT,
C                                     TRY OPTIMUM WITH NO UPPER LIMIT
   60 CONTINUE
      IF(ZA .LT. ZLX) GO TO 70
      CALL TRY(Z, DY1, B1)
C                                     FIND LIMIT FOR USE AFTER
C                                     BIVARIATE FIT
   70 CONTINUE
      ZU = ZU1 + TZUP*ABS(ZBEST)
      IF(TRACE)WRITE(6,81)ZLC,ZLF1,ZLF,Z1,ZUF
     +, ZLX, ZL1, ZL, Z, ZU
C                                     LIMIT Z FOR BIVARIATE SAMPLES
      IF(ZA .LT. ZLF) Z = SIGN(ZLF, Z)
      IF(ZA .GT. ZUF) Z = SIGN(ZUF, Z)
      IF(FREE5) RETURN
C                                     OPTION FOR BETTER ACCURACY,
C                                     SET Z TO BETTER OF ACTUAL SAMPLES
      Z = Z1
      IF(DIR*(DY2 - DY1) .GT. ZERO) Z = Z2
      RETURN
      END
C
C
      SUBROUTINE TRY(Z, DY, B)
C
C     THIS SUBROUTINE EVALUATES THE FUNCTION AT A POINT A DISTANCE Z
C     FROM XXO IN DIRECTION SS, IT ALSO CALCULATES DY AND B AND
C     MAINTAINS THE BEST POINT FOUND THUS FAR.
C
      IMPLICIT REAL*8 (A-H, O-Z)
C                                     COMMON BLOCK
      LOGICAL SORT, REV, COPY, PRINT, TRACE, BPUT, BETER, DONE
      LOGICAL FREE5, FREER
      COMMON /JOECOM/ ZERO, ONE, TWO, PI, FOUR, DEGRE, DIR
     +, SSS(15, 15), XXO(15), YO, BB(15), CC(15), ZZOPT(15), ZZ(15)
     +, XXBES(15), YBEST, ZBEST, SS(15), SMALS, YYC(15), YL, ZZL(15)
      COMMON /JOECOM/ TZO, TOLX, SMALX, SMALY
     +, TYLX, TYLY, TYLC, TZLR, TZLX, TZLF, TZL, TZLT
     +, TZLF1, TZUF2, TZU1, TZUFB, TZUB, TZUP, TZUT
      COMMON /JOECOM/ M, KRDR, KPRT, N, NSWEP, NSAMP, NPAIR
     +, ISWEP, ITALR, IPLAN, IFIT, ITRY, IBEST, IMOVE
     +, N3, K5, KR
      COMMON /JOECOM/ BETER, DONE, SORT, REV, COPY, PRINT, TRACE, BPUT
     +, FREE5, FREER
C                                     END COMMON
      DIMENSION XX(20)
      ITRY = ITRY + 1
      DO 29 I = 1, N
   29 XX(I) = XXO(I) + Z*SS(I)
C                                     EVALUATE NEW POINT
      CALL EVAL(Y, XX, N)
      DY = Y - YO
      B = DY/Z
      IF(DY*DIR .GE. ZERO) BETER = .TRUE.
```

```
      IF( (Y - YBEST)*DIR .LE. ZERO) GO TO 70
C                                          NEW BEST POINT
 1    FORMAT(' Y=', G15.7, '*Z=', G15.7, 18X, '*X=', (T58, 5G15.7) )
      IF(TRACE) WRITE(6, 1) Y, Z, (XX(K), K = 1, N)
      IBEST = IBEST + 1
      DONE = .FALSE.
      ZBEST = Z
      DO 69 I = 1, N
 69   XXBES(I) = XX(I)
      YBEST = Y
      RETURN
C                                          POINT NOT BETTER
 70   CONTINUE
 2    FORMAT(' Y=', G15.7, ' Z=', G15.7, 18X, ' X=', (T58, 5G15.7) )
      IF(TRACE) WRITE(6, 2) Y, Z, (XX(K), K = 1, N)
      RETURN
      END
C
C
      SUBROUTINE TALOR
C
C     THIS SUBROUTINE SAMPLES THE FUNCTION AT THE OVERALL OPTIMUM OF
C     THE MODEL (GAUSS POINT).
C
      IMPLICIT REAL*8 (A-H, O-Z)
C                                          COMMON BLOCK
      LOGICAL SORT, REV, COPY, PRINT, TRACE, BPUT, BETER, DONE
      LOGICAL FREE5, FREER
      COMMON /JOECOM/ ZERO, ONE, TWO, PI, FOUR, DEGRE, DIR
     +, SSS(15, 15), XX0(15), Y0, BB(15), CC(15), ZZCPT(15), ZZ(15)
     +, XXBES(15), YBEST, ZBEST, SS(15), SMALS, YYC(15), YL, ZZL(15)
      COMMON /JOECOM/ TZ0, TOLX, SMALX, SMALY
     +, TYLX, TYLY, TYLC, TZLR, TZLX, TZLF, TZL, TZLT
     +, TZUF1, TZUF2, TZU1, TZUFB, TZUB, TZUP, TZUT
      COMMON /JOECOM/ M, KRDR, KPRT, N, NSWEP, NSAMP, NPAIR
     +, ISWEP, ITALR, IPLAN, IFIT, ITRY, IBEST, IMOVE
     +, N3, K5, KR
      COMMON /JOECOM/ BETER, DONE, SORT, REV, COPY, PRINT, TRACE, BPUT
     +, FREE5, FREER
C                                          END COMMON
      ABS(W) = DABS(W)
      SIGN(W,W2) = DSIGN(W,W2)
      SQRT(W) = DSQRT(W)
      ITALR = ITALR + 1
C                                          FIND MODEL IN DIRECTION OF
C                                          OPTIMUM POINT.
      Z = ZERO
      B = ZERO
      C = ZERO
      DO 4939 I = 1, N
      W = ZERO
      DO 4929 J = 1, N
4929  W = W + SSS(I, J)*ZZOPT(J)
      SS(I) = W
      ZI = ZZCPT(I)
      IF(ABS(ZI) .LT. SMALX) GO TO 4939
      Z = Z + ZI*ZI
      B = B + ZI*BB(I)
      C = C + ZI*ZI*CC(I)
4939  CONTINUE
```

```
      Z = SQRT(Z)
      IF(ABS(Z) .LT. SMALX) GO TO 497
      DO 4949 I = 1, N
 4949 SS(I) = SS(I)/Z
      B = B/Z
      C = C/Z/Z
      ZBEST = ZERO
C                                        EVALUATE POINT
      CALL TRY(Z,DY,B)
   71 FORMAT(' TAYLOR', I3, I5, '     B=', G15.7, ' C=', G15.7, ' S=',
     +       (T58, 5G15.7) )
      IF(TRACE) WRITE(6, 71) ISWEP, ITRY, B, C, (SS(K), K = 1, N)
    1 FORMAT(' ')
      IF(TRACE) WRITE(6, 1)
C                                        MOVE TO POINT IF BETTER
      IF(YC .EQ. YBEST) GO TO 497
      IMOVE = IMOVE + 1
      B = B + C*ZBEST/TWO
C                                         IF C IS NONZERO AND OF CORRECT
C                                         SIGN AND IF B/C IS LESS THAN ZU
C                                         Z = -B/C , OTHERWISE
C                                         Z = ZU WITH SIGN FOR DOWNHILL
      Z = TZUT*Z
      IF(ABS(B).LT.-DIR*C*Z)Z = -B/C
C                                         CORRECT Z FOR MOVE
      Z = Z/ZBEST
      DO 4959 I = 1, N
      ZI = Z*ZZOPT(I)
      ZZOPT(I) = ZI
      ZL = TZLT*ABS(ZZ(I))
      IF(ZL .LT. ZZL(I)) ZL = ZZL(I)
      IF(ABS(ZI) .LT. ABS(ZL)) ZI = SIGN(ZL,ZI)
      ZZ(I) = ZI
 4959 XXC(I) = XXBES(I)
      YO = YBEST
      ZBEST = ZERO
  497 CONTINUE
    2 FORMAT(' Y=',G15.7,3X,3I5,18X,' X=',(T58,5G15.7))
      IF(PRINT)WRITE(6,2)YO,ISWEP,ITRY,IBEST,(XXO(K),K=1,N)
    3 FORMAT(T55, ' C=',(T58,5G15.7) )
      IF(PRINT) WRITE(6, 3) (CC(K), K = 1, N)
      IF(BPUT) CALL PUT(3)
      IF(PRINT)WRITE(6,1)
      RETURN
      END
C
C
      SUBROUTINE INIT(X, ZZ, DIR, NX)
C
C     THIS IS A SAMPLE OF THE PROBLEM INITIALIZATION SUBROUTINE.
C     X IS THE INITIAL LOCATION.  ZZ IS THE INITIAL STEP SIZE WITH
C     TZO*X WHERE TZO HAS THE DEFAULT 0.1 .  DIR = +1. SPECIFIES
C     MAXIMIZATION, DIR = -1. SPECIFIES MINIMIZATION (DEFAULT).
C     N SPECIFIES THE NUMBER OF ELEMENTS OF X.
C     THE PROBLEM IS ROSENBROCKS CURVED VALLEY
C     STARTING AT X1 = -1.2 AND X2 = 1.0 .
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(15), ZZ(15)
    1 FORMAT(' FUNCTION: ROSENBROCK''S CURVED VALLEY')
```

```
      WRITE(6, 1)
      NX = 2
      X(1) = -1.2
      X(2) = 1.0
      RETURN
      END
C
C
      SUBROUTINE EVAL(F, X, N)
C
C     THIS IS A SAMPLE OF THE PROBLEM FUNCTION EVALUATION SUBROUTINE.
C     F RETURNS THE FUNCTION VALUE, X PROVIDES THE CURRENT LOCATION
C     (INDEPENDANT VARIABLES) AND N GIVES THE NUMBER OF ELEMENTS OF X
C     AS SET BY INIT (FOR USE IN GENERALIZED PROBLEMS).
C     THE PROBLEM IS ROSENBROCKS CURVED VALLEY
C     Y = (X1 - 1.)**2 + 100.*(X1**2 - X2)**2
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(15)
      A = X(1)
      B = A - 1.
      C = A*A - X(2)
      F = B*B + 100.0*C*C
      RETURN
      END
```

VITA

Fred Earl Witz

Candidate for the Degree of

Doctor of Philosophy

Thesis: A NEW DIRECT SEARCH METHOD FOR UNCONSTRAINED FUNCTION OPTIMIZATION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Fargo, North Dakota, October 14, 1946, son of Richard and Marjorie Witz.

Education: Graduated from Fargo Central High School, Fargo, North Dakota, in 1964; graduated from North Dakota State University, Fargo, North Dakota, with a Bachelor of Science degree in Electrical Engineering in 1968; received the Master of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma, in 1970; completed the requirements for Doctor of Philosophy degree from Oklahoma State University, Stillwater, Oklahoma in May, 1976.

Professional Experience: Computer Programmer, Electrical Engineering Department, Oklahoma State University, Summer 1969 and 1970; Computer Programmer, Agricultural Engineering Department, Oklahoma State University, 1974; Tutor, College of Engineering, Oklahoma State University, 1971-1975; Computer Programmer, Engineers for Agriculture, Stillwater, Oklahoma, 1975.

Professional and Honorary Organizations: Member of Tau Beta Pi, Eta Kappa Nu.