A SURVEY OF NON-SMOOTH OPTIMISATION METHODS AND AN EVALUATION OF A METHOD FOR MINIMAX OPTIMISATION

By

ROSEMARY FERNANDES

Bachelor of Technology
Indian Institute of Technology
Madras, India
1977

Master of Engineering
Asian Institute of Technology
Bangkok, Thailand
1979

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements
for the Degree of
MASTER OF SCIENCE
December 1985

A SURVEY OF NONSMOOTH OPTIMISATION TECHNIQUES

AND AN EVALUATION OF A METHOD FOR

MINIMAX OPTIMISATION

Thesis Approved:

J. P. Chandler
Thesis Adviser

Sharlyn A Thoreson

Norman N. Durham
Dean of Graduate College

PREFACE

This thesis surveys the recent developments in nondifferentiable optimisation and examines the performance of a two-stage method suggested by Hald and Madsen. A modification is suggested for the second stage and a comparison is presented.

I would like to express my deep appreciation and thanks to my adviser, Dr. J. P. Chandler, for his intelligent guidance, thoughtfulness and encouragement.

I am also thankful to my other committee members, Dr. Thoreson and Dr. Grace for their advice and support.

Very special thanks are due to my friend, Mei-Hui Chen, for her continued support and help in typing the thesis.

I am very grateful to my parents, Mr. and Mrs. Thomas and to my Parents-in-law, Dr. and Mrs. Fernandes for their encourgement and understanding.

I wish to dedicate this thesis to my husband Gerard, and my son, Shane, who have made everything seem so good and worthwhile.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Nonsmooth optimisation or nondifferentiable optimisation (NDO), as opposed to smooth optimisation, refers to problems where the objective function to be minimised is not necessarily differentiable everywhere. This phenomenon occurs frequently in mathematics and optimisation. Furthermore, nondifferentiable functions are, in general, more difficult to minimise than smooth functions. Hence there is a need to find efficient and practical methods to solve the NDO problem.

In recent years there has been a growing interest in developing techniques to solve nonsmooth optimisation problems [27]. Various approaches have been suggested, many of them are based on methods already available for smooth optimisation. There is an enormous amount of literature available on smooth optimisation, the methods of steepest descent and conjugate gradients, and also quasi-Newton methods have reasonable extensions to non-smooth optimisation problems.

At present there is a considerable interest in this area and it is not possible to say yet what the best approaches are [27]. A survey of the recent developments in this field is presented in chapter II.

Problems in NDO can [31], in general, be treated as problems with random discontinuities in the objective function or as problems in which a great deal of information is available about the nature of the discontinuities. Most nondifferentiable optimisation problems can be formulated as composite functions [27]. However, in practice this may be complicated or may require too much storage. There are various algorithms to solve such composite functions. A common kind of composite function studied is the Minimax problem, which can be defined as the minimisation of a function $F(x)$ where

$$F(x) = \max \{ f_j(x) \}, \qquad j = 1, \ldots\ldots, m$$

and $f_j(x)$ are smooth functions.

When the only information available at any point x is $f(x)$ and a normal vector g to a supporting hyperplane, the problem is more difficult to solve. If the function f is nondifferentiable at x, g is referred to as the subgradient at x. The subdifferential ∂f is defined as the set of all subgradients at x. This class of problem is called the basic NDO. Fewer methods are available for basic NDO. Algorithms for basic NDO have not progressed far because of the limited availability of information.

Some simple examples of problems [12] that occur in NDO are described below. The first example is that of finding the best solution to an overdetermined system (m>n) such as occurs in data fitting applications. Given a set of data points, the problem of finding the best linear fit so that the error is minimised is a non-differentiable problem.

$$\sum_{i=0}^{n} e_i \quad \text{where } e_i = | m x_i + b - y_i | \text{ is}$$

nondifferentiable as a function of m and b.

Consider a simple problem in elasticity. An elastic band whose upper end is fixed and lower end is tied to a unit point mass. When the band is stretched by a positive amount x, it exerts an upward (restoring) force propotional to x. When unstretched no force is exerted. When the mass is oscillating vertically the force, f is given by

$$f(x) = \begin{cases} g - kx & \text{if } x >= 0 \\ g & \text{if } x <= 0. \end{cases}$$

where g is the acceleration due to gravity and k is the propotionality constant for Hooke's law. The function is continuous but may not be differentiable at 0.

Another example is when the constraints are themselves dependent on parameters.

$$\begin{aligned} &\text{Min} && f(x) \\ &\text{subject to} && g(x) + p <= 0 \\ & && h(x) + q = 0 \end{aligned}$$

The solution v(p,q) depends upon p and q and is not differentiable everywhere, e.g. where g(x) + p = 0.

One of the most important applications of NDO is in the area of nonlinear programming through the use of exact penalty functions [28]. By reformulating some difficult problems in linear and nonlinear programming as NDO problems, we can increase the ability to handle such problems.

A study of a method for minimax problems by Hald and Madsen [34] and modification of this method following Fletcher's [28] guidelines is described in chapter III. The performance of the modified method is tested using the test problems described in chapter IV, and by comparison to similar methods. Some mathematical definitions are given in appendix A. A large bibliography is also included. Appendix B contains the program listing.

CHAPTER II

A SURVEY

The interest in developing techniques to solve NDO problems has been recent. Until 1964, the method most investigated [9] for the minimisation of nondifferentiable functions was the so called "cutting plane method", discovered by Cheney and Goldstein [10] and independently by Kelley [47]. Cutting plane methods have been used widely in constrained optimisation.

Cutting plane algorithms are elementary in principle. A series of improving approximate linear programs, whose solutions converge to the solution of the original problem, are developed. Cutting plane algorithms determine the hyperplane that separates a current point x from the constraint set. Algorithms differ in the manner in which the hyperplane is selected. This selection is an important aspect of the algorithm, since it is the distance of the hyperplane from the current point that determines the rate of convergence of the method [56]. Nondifferentiable convex functions allow the possibility of a number of supporting hyperplanes as illustrated in figure 1.

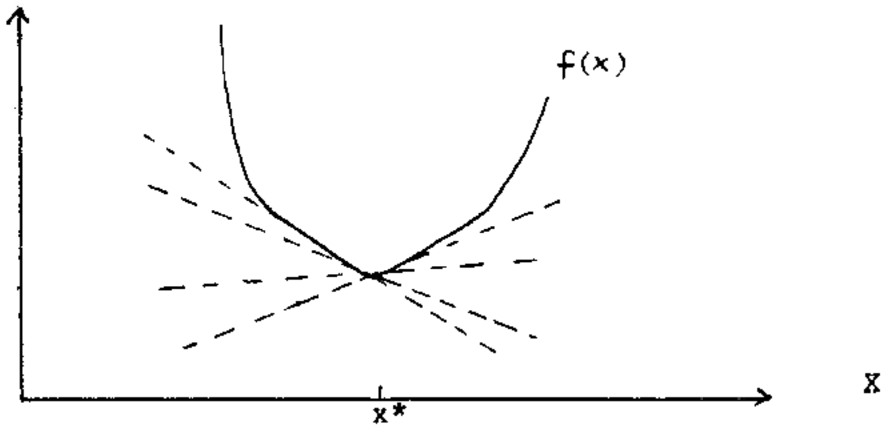

Figure 1. Supporting hyperplanes to non-differentiable convex function

A description of cutting plane methods is given by Leunberger [56] and Zangwill [94]. The convergence of the cutting plane methods does not depend upon the differentiability of the objective function. As observed by Wolfe [88] the rate of convergence of cutting plane algorithms seems better for non-smooth functions than it is for smooth functions. A refinement of the cutting plane method is given by Hogan [45]. Some results on the convergence rates of cutting plane algorithms are given by Eaves and Zangwill [21] and Wolfe [89].

In 1964, Shor [81] pioneered the subgradient algorithm. Since that time the method has been highly developed in the Sovient Union. Subgradient (SG) optimisation is a technique that attempts to solve the problem of minimising a general nondifferentiable convex function, and is about the simplest possible general method for solving basic NDO problems. Shor's method is applicable to any convex function. A good survey of Soviet research in this field is given by Poljak [73]. It reviews the research efforts by Soviet authors in developing subgradient methods for NDO.

The minimisation method using space dilation in the direction of the difference of two successive gradients due to Shor [82] has been found [9] to be a very effective method for difficult non-differentiable problems. It has been observed [51] that good results are obtained by using Shor's method of space dilation and quasi-Newton methods. For certain structured LP programming problems whose size makes any known version of the simplex method impractical, the simple algorithm due to Shor has proved to be effective [45]. But that it does not converge as fast as even the steepest descent methods when the function is differentiable.

A convex function f(x) allows the possibility of a number of supporting hyperplanes at a nondifferentiable point x as was shown in figure 1. For each hyperplane we can define

$$f(x+h) >= f(x) + h^T g$$

where g is a normal vector to a hyperplane at x. Such a vector is referred to as the subgradient at x. The set of all subgradients at x is referred to as the subdifferential at x and is defined by

$$f(x) = \{ g \mid f(x+h) >= f(x) + h^T g \}$$

To solve the basic nondifferentiable problem Lemarechal [51] considers also an extension of the powerful method of conjugate gradients which has been widely used in unconstrained optimisation of smooth functions. In [53] Lemarechal tries to synthesize conjugate subgradient methods and to extend them to a wider class of bundle methods. The method is based on "bundling" subgradients. The objective function is required to be regular. [see appendix A]

A similar method based on bundling subgradients is described by Wolfe[88]. This method is reasonably effective for both differentiable and nondifferentiable convex objective functions. When f is quadratic this method is exactly that of Hestenes and Stiefel [45].

The bundle methods try to accumulate information locally about the subdifferential of the objective function. A bundle method is a line search method which solves subproblems to define the step direction. The subgradients are used to find the step direction, and are added to the bundle B on sucessive iterations. The method continues this way until $o \in B$. Then the bundle is reset, for instance, to the current subgradient and the iteration is continued. With careful manipulation of B [27], a convergence result can be proved and a suitable termination test obtained. In these methods a sequence $\{x_k\}$ is generated where

$$x_{k+1} = x_k + h_k p_k$$

where h_k is stepsize and p_k is the direction.

Mifflin's algorithm [66] is a modification of the algorithm by Lemarechal [53]. This version differs from that of Lemarechal because of its rules for line search termination and the associated updating of the search direction. Mifflin's method can be used on a wider class of optimisation problems with only minimal restrictions on the allowable type of constraints or objective function [66].

Subgradient methods have been used to solve large scale problems. Generalisations of the SG methods beyond convex objective functions have been attempted by Nurminskii [69] [70] with partial success.

The application and extension of the relaxation method, referred to as subgradient relaxation methods, to certain dual problems in network scheduling is discussed by Fisher et al [24]. Chaney and Goldstein [9] present an extension of the subgradient method to max families and quasi-differentiable functions. An algorithm for solving ordinary nonlinear programming problems in a NDO context is described by Pshenichnyi [75]. The rate of convergence of this method is also investigated. A good bibliography can be found a book by Lemarechal and Mifflin [54].

A class of algorithms for minimising any convex, not necessarily differentiable, function f of several variables is described by Kiwiel [48]. The methods require only the calculation of f and one subgradient of f at designated points. These methods generalise Lemarechal's bundle method [53]. Instead of using all previously computed subgradients, the method uses an aggregate subgradient which is recursively updated as the algorithms proceed. The algorithms can be viewed as an extension of Pshenichnyi's linearisation method [75]. The concept of aggregation has also been applied in [49] [50], to a modified algorithm due to Mifflin [66].

Application of a boxstep method to column generation problems and a variety of scheduling problems is described by Marsten [62]. The performance of the boxstep method is compared to that of subgradient optimisation methods.

Application of some versions of steepest descent methods to NDO have been considered by Demjanov [17] and Bertsekas and Mitter [6]. A survey of the area and an . extensive bibliography may be found in [6]. Most of these methods are restricted in its application to non-differentiable problems, and do not seem to have a straight foward implementation in the general case. A procedure by Cullum et al to certain solve nondifferentiable sums of eigenvalues of symmetric matrices based on steepest descent is given in [14].

Function comparison methods (also known as direct search methods), a class of general methods for minimising smooth functions, have also been applied to NDO problems. The only advantage of these methods is that they are in general simple. The major disadvantage is that few guarantees can be made regarding convergence; moreover, they are often slow. In these methods, successive estimates x of the minimiser x* is made by comparing the values of the objective function at a general set of points including x. Examples of direct search methods are the simplex method of Nelder and Mead [61], and methods of Rosenbrock [78], Hooke and Jeeves [46], Spendley, Hext and Hemsworth [83] and Davies, Swann and Campey. Although the method of Powell [74] is in principle a conjugate direction method, the computation of partial derivatives is not required. A similar method is that of Zangwill [93].

The simplex method is used more often than the other direct search methods, and the general principles are described below. A simplex in R may be thought of as a polyhedron with n+1 distinct vertices, denoted by v_i, i = 1, ..., n+1. Hence by replacing any point v_i by w, we obtain a new simplex. Given a set of rules for changing the current simplex and by requiring that each vertex of the simplex is a value of the function F(x), we can generate a sequence of simplices so that the final simplex may have the minimiser x* as one of the vertices. The precision of the estimate depends upon the size of the final simplex.

Spendley, Hext, and Hemsworth [83] appear to have been the first authors to propose a simplex method, but their strategy was too rigid to permit rapid convergence in most cases. An efficient simplex method is that described by Nelder and Mead [61].

It has been suggested by Wright et al [31] that direct search methods may be used to solve the non-differentiable optimisation problem, when the function or its gradient is discontinuous at its solution or when the gradient has many discontinuities or when the discontinuties have no special structure.

Variable metric methods, also known as quasi-Newton methods, are effective for minimising smooth functions. An application of quasi-Newton methods to NDO problems is suggested by Han [40]. He developed a class of methods for minimising a nondifferentiable function which is the maximum of a finite number of smooth functions. The method proceeds by solving iteratively quadratic programming subproblems to generate search directions. The combined Hessian matrices B in the quadratic programming problems are updated in a variable metric way. The stepsize procedure does not use an exact line search. However, as pointed out by Fletcher [29], the combined Hessian matrix B is updated by differences in the qradient of a Lagrangian function and hence depends upon Lagrange multiplier estimates. If the estimates become unbounded then B is likely to become unbounded.

Various other methods have been developed for nondifferentiable functions. The most general class of NDO composite functions is the minimax problem as defined by (1.1). Most of the methods surveyed here are applicable to the minimax problem.

For such problems, an algorithm with second order convergence can be obtained by linearising the individual functions over which the max is taken. Studies of this type of method has been conducted by Osborne and Watson [72] and Charalambous and Conn [8]. As with Gauss-Newton methods, convergence is not guaranteed. This can be solved by using a restricted step type of method. Application of a restricted step type method to overdetermined systems (m>n) of nonlinear equation has been investigated by Madsen [57]. The functions are assumed to be continuous. The algorithm is based on successive linear approximations to these functions. The resulting linear systems are solved subject to bounds. The convergence of this algorithm is guaranteed and the rate of convergence on regular functions is quadratic. However, on singular functions [see appendix A], the convergence is only linear. In order to obtain a better rate of convergence Hald and Madsen [34] have proposed a two-stage algorithm. The stage1 algorithm is the same as the one described by Madsen [57]; a switch to stage2 is made when irregularity is detected. The stage2 algorithm uses a quasi-Newton method. Another method to solve the problem of singular functions is suggested by Madsen and Schjaer-Jacobsen [59].

General nonlinear minimax approximation problems [1] involving a finite point set have been reformulated and solved by well-established methods such as the barrier function method of Fiacco and McCormick [23]. Application of NDO in the area of nonlinear programming through the use of penalty functions is described by Fletcher [27]. Another approach is to use an algorithm for nonlinear programming as a means of generating a direction of search, and to use the exact penalty function as the criterion function to be minimised approximately. This approach is described by Han [38], Coleman and Conn [13], and Mayne [64].

A general algorithm for composite nondifferentiable optimisation problems has been presented by Fletcher. In [28] Fletcher considers the minimisation of composite functions from a nonlinear optimisation viewpoint. This class of composite functions is quite general since it includes exact penalty function, nonlinear minimax functions and best approximations. Using both linear approximations of the constraints and quadratic approximation to F, Fletcher proves that the method has second order rate of convergence. He also shows that his method converges globally if a trust region is incorporated on the stepsize. The method is called the QL method, since it makes both quadratic and linear approximations.

Rockafellar [76] and Womersley [92] both deal with optimality conditions. Wormersley derives second order necessary and sufficient conditions for problems involving piecewise smooth functions. Rockafellar derives first order conditions for problems whose constraints and objective function are locally lipschitz. Optimality conditions have also been described by Fletcher [27].

Currently, research is being carried out in many of these areas. Because of its simplicity, the subgradient method has received much attention, but it is at best linearly convergent. The Bundle methods are also being investigated. The possibility of using quasi-Newton methods to update the matrix B in the quadratic programming subproblems is being studied. The modified BFGS formula given by Powell [74] is expected to work well.

CHAPTER III

A MINIMAX METHOD

The method descrbed in this chapter is the method proposed by Hald and Madsen [33]. It combines linear programming and quasi-Newton methods for minimax optimisation, and consists of two stages. The algorithm used in stage 1 is based on successive linearisations of the objective function. The resulting linear subproblems are solved subject to bounds. The bounds are adjusted depending on how good the approximation is to the objective function. It was proved [33] that the stage 1 algorithm has quadratic convergence when there are n+1 active functions at x*, that is, when the function is regular. In other words, the problem satisfies the Haar condition. [see Appendix A] The stage 2 quasi-Newton algorithm is used only if an irregular solution is detected. In this case, second order derivative information may be needed in order to obtain a fast final rate of convergence. If stage 2 iteration is unsuccessful, then a switch is made back to stage 1. Several switches may be necessary before the solution is found.

It has been proved [36] that the algorithm will always converge to a stationary point of the problem.

Details of Hald and Madsen Method

The minimax problem can be defined as the minimisation of a maximum function F(x), where the maximum is taken over a finite set.

$$F(x) = \max \{f_1(x),\ f_2(x),\ \ldots\ldots,\ f_m(x)\}, \qquad (1)$$

$f_j(x)$, $j = 1, m$ are smooth functions,

$x = \{x_1,\ x_2,\ \ldots\ldots,\ x_n\}$.

The objective function is, in general, a non-differentiable function having discontinuous first partial derivatives at the minimum. The minimum is normally situated at a point where two or more functions are equal. When the minimum is well determined, only first order information is required, and the convergence is quadratic. However, if the minimum lies in a smooth valley, a quasi-Newton method is used to obtain a fast final rate of convergence.

The method consists of four parts:

(i) STAGE 1 ITERATION

(ii) CONDITIONS FOR SWITCHING TO STAGE 2

(iii) STAGE 2 ITERATION

(iv) CONDITIONS FOR SWITCHING BACK TO STAGE 1

(i) STAGE 1 ITERATION

The minimiser x* for the objective function F(x) defined by (1) is determined by sucessive iterations. Suppose an approximate feasible estimate of the minimiser at the kth iteration is x_k. The increment h_k is determined as a vector that minimises $F(x_k, h_k)$, which is linearly approximated by $\bar{F}(x_k, h_k)$, using Taylor's series.

$$\bar{F}(x_k, h_k) = \max \left\{ f_j(x_k) + \sum_{i=1}^{n} \frac{\partial f_j}{\partial x_i}(x_k).h_i \right\} \qquad (2)$$

$$j = 1, \ldots m$$

subject to the constraint

$$\| h \| = \max | h_1, h_2, \ldots . h_n | <= \Lambda_k , \quad \Lambda_k > 0. \qquad (2a)$$

Since (2) is valid only for small values of h, the value of $\| h \|$ is forced to be small enough by using the restriction (2a).

The value of Λ depends on how good the linear approximation is to the objective function, and is chosen as large as possible subject to a certain measure of agreement being maintained between each f_j and its linearisation.

The above problem can be transformed into the following linear program by introducing an extra variable p

$$\underset{h,\, p}{\text{Minimise}} \quad p$$

$$\text{Subject to} \quad f_j(x_k) + \sum_{i=1}^{n} \frac{\partial f_j}{\partial x_i}(x_k)\, h_i <= p \qquad (3)$$

$$-\Lambda_k <= h <= \Lambda_k$$

This problem can be solved by a standard linear programming method. We have used the method for quadratic and linear programming by Lemke. The formulation of (3) for Lemke's algorithm is described in a later section.

The point $x_{k+1} = x_k + h_k$ can be accepted as the next point in the iteration if the function $F(x_{k+1})$ decreases. However, as pointed out by Fletcher [28], this condition is not sufficient to guarantee convergence. The following condition is used

$$F(x_k) - \bar{F}(x_k+h_k) >= C_1[\ F(x_k) - \bar{F}(x_k,\ h_k)\ ] \qquad (4)$$

where

C_1 is a small positive number.

That is, if the decrease in the objective function exceeds a small multiple of the decrease predicted by linear approximation it implies there is adequate agreement between objective function and its approximation.

If the condition (4) is satisfied, then

$x_{k+1} = x_k + h_k$ otherwise,

$x_{k+1} = x_k$

There is no line search involved.

Determination of Λ_{k+1}

The value of Λ_{k+1} depends upon how well the iteration approximates the linear function to the actual, and is determined so as to try and provide the inequality

$F(x_k\ ,\ h_k) < F(x_k)$.

If the decrease in the objective function

$$F(x_k) - F(x_{k+1}\ ,\ h_k) \text{ is } <= C_2[\ F(x_k) - F(x_k,\ h_k)\ ], \qquad (5)$$

$C_1 < C_2 < 1$.

then the decrease in F is rather poor. Hence we use a smaller bound

$$\Lambda_{k+1} = C_3 \; \| h_k \|, \qquad C_3 < 1. \tag{6}$$

If $F(x_k) - F(x_k + h_k) <= C_4[\, F(x_k) - \bar{F}(x_k, h_k)\,],$ (7)

$C_2 < C_4 < 1.$

Then the decrease in F is close to the decrease predicted by linear approximation, hence the bound is increased

$$\Lambda_{k+1} = C_5 \; \| h_k \|. \tag{8}$$

In all other cases,

$$= C_6 \; \| h_k \|. \tag{9}$$

The parameters C_1, C_2, C_3, C_4, C_5 and C_6 are arbitrary and are not very sensitive. The values generally used are 0.01, 0.25, 0.75, 2.0, 1.0 or 0.5, respectively.

Determination of active set

An important concept is that of an active set. For each iteration in stage 1, the active set A is determined. It is defined by the index set,

$$A_k = A(x_k) = \{\, j \mid F(x_k) - f_j(x_k) <= \epsilon_1 \} \tag{10}$$

where ϵ_1 is a small positive number defined by the user. We have used $\epsilon_1 = .01F(x\;)$. This defines the functions that are "active" at x. A* is defined as follows,

$$A^* = A(x^*) = \{\, j \mid F(x^*) = f_j(x^*)\, \}, \tag{11}$$

and contains the index set of the functions that are active at the solution.

(ii) CONDITIONS FOR SWITCHING TO STAGE 2

A switch is made to stage 2 when a smooth valley is detected through the solution. In general, at the minimum (x*) some functions are equal. Suppose that the number of such functions is S and the functions are f_j, such that,

$$F(x^*) = F\ (x^*) > f_j(x^*) \tag{12}$$

for $\quad j \in A(x^*)$

$\quad i \notin A(x^*)$.

Then, the following must hold in the valley and at the solution

$$f_{j0}(x) - f_j(x) = 0, \qquad (j \neg= j\ ) \tag{13}$$

$j \in A(x^*)$ and $j_0 \in A(x^*)$ is fixed.

If s >= n+1, then the Haar condition is satisfied. This implies that the Jacobian $\{ f'(x^*) \mid f_j(x^*) = F(x^*) \}$ has a rank n. Then the minimum is well determined and there is no smooth valley. However, if s <= n, then the Jacobian has rank < n, and we require more information to obtain a fast convergence.

Suppose the latest three iterates x_k, x_{k-1} , x_{k-2} have been calculated in stage 1 then a switch to stage 2 is made if the following conditions (14), (15), and (16) are satisfied.

If $\lambda_j >= 0 \qquad j \in A$,

and $\quad \Sigma \lambda_j = 1$

then,

$$\| h_k \| = \Lambda_k \tag{14}$$

$$A_{k-1} = A_{k-2} = A_k \tag{15}$$

$$\left\| \sum_{j \in A} \lambda_j f'_j(x_k) \right\| <= \epsilon_2 \tag{16}$$

where $\quad \epsilon_2 > 0$.

Note: Condition (16) is tested only if (14) and (15) are satisfied, and is true when x_k is close to a solution x^* with $A^* = A_k$.

These conditions ensure that unnecessary iterations are avoided in stage 2. If the quasi-Newton iteration is started with the wrong active set, a switch would be made back to stage 1 after a few iterations.

(iii) STAGE 2 ITERATION

Stage 2 is used only when the curvature effects are not negligible and the value of x is close to the minimum x*.

Suppose the functions that are equal at the minimum be defined as in (13). Then for a local minimum the following conditions must hold:

$$\sum_{j \in A} \lambda_j f'_j(x) = 0, \tag{17}$$

$$\left(\sum_{j \in A} \lambda_j\right) - 1 = 0, \tag{18}$$

$$\lambda_j >= 0$$

and $$f_{j0}(x) - f_j(x) = 0 \tag{19}$$

where, $jo \in A$,

$j \in A$,

$jo \neg= j$

The unknowns are λ and x. A quasi-Newton method is used at this stage. The quasi-Newton method used at this stage should be locally and linearly convergent.

Instead of using the quasi-Newton iteration as suggested by Hald and Madsen, I have used a method similar to one described by Fletcher [30] for the stage 2 iteration.

It has been proven that this method has quadratic convergence and hence is an improvement over the one suggested by Hald and Madsen. A comparison is presented in the next chapter.

The conditions (17) and (18) become the Kuhn-Tucker conditions when (1) is put into the following form

$$\begin{array}{ll} \text{Min} & v \\ \text{subject to} & f_j(x) <= v \end{array} \qquad (20)$$

By using the following quadratic approximation for f

$$f(x+h) = f(x) + f'(x) * h + h^T f''(x) * h.$$

We can determine h , at the kth iteration from

$$\begin{array}{ll} \underset{h,\ v}{\text{Min}} & v + 1 / 2\ h^T B\ h \\ \text{subject to} & f\ (x_k) + f'(x_k) * h <= v \end{array} \qquad (21)$$

where, B is defined by

$$B = \sum_j \lambda_j f_j, \quad j \in A$$

As described before, the restricted stepsize condition (22) is introduced to ensure convergence,

$$\|\ h\ \| <= \Lambda_k \qquad (22)$$

Hence problem (21a) can be written as

$$\begin{array}{ll} \underset{h,\ v}{\text{Min}} & v + 1 / 2\ h^T B\ h \\ \text{subject to} & v - f'(x_k) * h >= f(x_k) \\ & h + \Lambda >= 0 \\ & -h + \Lambda >= 0 \end{array} \qquad (23)$$

This is a quadratic programming problem, and is solved using Lemke's algorithm.

(iv) CONDITIONS FOR SWITCHING BACK
TO STAGE 1

A switch is made back to stage 1 if any of the following conditions (24), (25) or (26) fail to hold. Suppose $r(x,\lambda)$ denotes the vector of the left hand side of (17), (18) and (19). In order to continue the quasi-Newton iteration, the length of the vector r should decrease.

$$\| r(x_{k+1}, \lambda_{k+1}) \| <= \eta \| r(x_k, \lambda_k) \| \quad (24)$$

where $0 < \eta < 1$. (We use $\eta = .999$.)

A test that no function with an index from outside the active set becomes dominating is made

$$F(x_{k+1}) = \max \{ f_j(x_{k+1}) \}, \; j \in A \quad (25)$$

The multipliers corresponding to the active set should be non-negative

$$\lambda_j >= 0, \quad j \in A. \quad (26)$$

These conditions ensure that convergence is maintained in stage 2.

Methods Used for Stage 1 and Stage 2

The algorithms used for the stage 1 and stage 2 iterations are described in this section.

(i) ALGORITHM USED TO SOLVE THE LINEAR
PROGRAM OF STAGE 1

The method used to solve the linear program of stage 1 and the quadratic program of stage 2 is the Lemke's algorithm for quadratic and linear programming. Lemke's algorithm is an extension of the Simplex method to solve

$$\begin{aligned} &\text{minimise } 1/2x^T Gx + g^T x \quad \text{where G is positive definite} \\ &\text{subject to } A^T x >= b \\ &\qquad\qquad\quad x >= 0 \end{aligned} \tag{27}$$

Using Wolfe's dual, this can be restated using Lagrangian multipliers y for the constraints $A^T x >= b$, and u for bounds $x >= 0$.

The associated Lagrangian function L(x, y, u) is then expressed as,

$$L(x,y,u) = 1/2x^T Gx + g^T x - y^T (A^T x - b) - u^T x. \tag{28}$$

Define slack variables

$$v = A^T x - b. \tag{29}$$

The first order necessary conditions (or the Kuhn-Tucker (KT) conditions: see appendix A) for (28) are then

$$\begin{aligned} &u - Gx + Ay = g \\ &v - A^T x = -b \\ &u,\ y,\ v,\ x >= 0 \\ &y^T v = 0 \\ &x^T u = 0 \end{aligned} \tag{30}$$

The linear complementarity problem then be expressed as

$$\begin{aligned} &w - Mz = q \\ &w^T z = 0 \\ &w >= 0,\ z >= 0 \end{aligned} \tag{31}$$

where,

$$w = \begin{vmatrix} u \\ v \end{vmatrix}, \quad z = \begin{vmatrix} x \\ y \end{vmatrix}, \quad M = \begin{vmatrix} G & -A \\ A & 0 \end{vmatrix}, \quad q = \begin{vmatrix} g \\ -b \end{vmatrix}.$$

FORMULATION FOR STAGE 1

The stage 1 linear problem to be solved may be written as,

$$\begin{array}{ll} \text{Minimise} & p \\ \text{Subject to} & p - \bar{F}(x_k, h_k) >= 0 \\ & h + \Lambda >= 0 \\ & -h + \Lambda >= 0 \end{array} \tag{32}$$

Introduce non-negative variables r and s, defined by

$$r_i - s_i = h \qquad i = 1, \ldots, n \tag{33}$$

$$r_{n+1} - s_{n+1} = p$$

The Lagrangian function for the stage 1 linear problem can be expressed as

$$L(x, y, u) = g^T x - y^T(Ax-b) - u^T(x_k)$$

where,

$$x = [\, r_1, \ldots, r_{n+1}, s_1, \ldots, s_{n+1} \,] \tag{34}$$

$$b = [\, f_j(x_k), -\Lambda, -\Lambda \,] \qquad j = 1, \ldots, m$$

$$\Lambda = \Lambda e_m$$

$$e_n = [\, 1, 1, \ldots, 1] \quad \text{n-vector}$$

$$e_m = [\, 1, 1, \ldots, 1] \quad \text{m-vector}$$

$$g = [\, 0, \ldots, 0, 1, 0, \ldots, -1 \,]$$

$$J = \begin{vmatrix} \dfrac{\partial f_1(x_k)}{\partial x_1} & \cdots & \dfrac{\partial f_1(x_k)}{\partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial f_m(x_k)}{\partial x_1} & \cdots & \dfrac{\partial f_m(x_k)}{\partial x_n} \end{vmatrix}$$

$$A = \begin{vmatrix} -J_{m*n} & I_{n*1} & J_{m*n} & -I_{n*1} \\ I_{(n+1)*n} & 0_{(n+1)*1} & -I_{(n+1)*n} & 0_{(n+1)*1} \\ -I_{(n+1)*n} & 0_{(n+1)*1} & I_{(n+1)*n} & 0_{(n+1)*1} \end{vmatrix}$$

I = m*m unit matrix

I = n*n unit matrix

J is an m*n matrix

A is an (m+2(n+1))*(2(n+1)) matrix

The Kuhn-Tucker conditions are the same as equations (30).

The linear complementarity problem (32) is solved using

$$w = \begin{vmatrix} u \\ v \end{vmatrix}, \quad z = \begin{vmatrix} x \\ y \end{vmatrix}, \quad M = \begin{vmatrix} 0 & -A \\ A & 0 \end{vmatrix}, \quad q = \begin{vmatrix} g \\ -b \end{vmatrix}.$$

(iv) ALGORITHM USED FOR THE QUADRATIC PROGRAM OF STAGE 2

The quadratic programming problem (26) can be solved using the same algorithm of stage 1: Lemke's algorithm for quadratic and linear programming.

The variables are as described in (35). The only addition is the matrix G which can be formulated as shown below

$$G = \begin{vmatrix} W_{n*n} & 0_{n*1} & -W_{n*n} & 0_{n*1} \\ 0_{1*n} & 0_{1*1} & 0_{1*n} & 0_{1*1} \\ -W_{n*n} & 0_{n*1} & 0_{n*n} & 0_{n*1} \\ 0_{1*n} & 0_{1*1} & W_{1*n} & 0_{1*1} \end{vmatrix}$$

Termination Criteria

It has been proved in [19] that the method converges to a stationary point. The kth iteration is terminated when the following is true

$$x_{k+1} - x_k <= s.$$

The value of s used is .5d-5. ϵ_2 in equation (16) is determined as follows

$$\epsilon_2(x_k) = 0.5 \min_{j \in A} \| f' (x) \| \quad , s > 1$$

$$= .01\ F(x_k) \quad , s = 1.$$

CHAPTER IV

TESTING AND DISCUSSION

The performance of the modified algorithm is examined by comparing the number of iterations required to obtain a convergence using the same termination criteria as used by Hald and Madsen. The method is also compared to the method of Charalambous and Conn [8] using the test problems described in their paper.

The number of iterations required by stage1 and stage2 independently is also evaluated and a comparison is presented. It was observed that the method is sensitive to the initial value of Λ. For each test problem, different values of Λ were given and the rate of convergence tabulated. A line search was also used to improve the convergence of slowly converging iterations.

The iterations are counted for each linear or quadratic subproblem solved. The test problems used are described below.

Test Problems

Example 1.

This is the example 2 of Madsen [56].

$f_1(x) = x_1^2 + x_2^2 + x_1 x_2$

$f_2(x) = \sin x_1$

$f_3(x) = \cos x_2$

Starting point (3,1), $\Lambda = 1.2$

$x^* = [\ -.4533,\ .90659\ ]$, $F^* = [\ .61643,\ .43793,\ .61643]$

The table below is a comparison of the number of iterations required by stage1, stage2, and the combined method to the method by Madsen [57]. The maximum stepsize is also indicated.

TABLE I

COMPARISON OF NUMBER OF ITERAIONS TO SOLVE PROBLEM 1

	No. of iterations	Function value	$\|\| h \|\|$
Stage 1	27	.61643d0	.5d-5
Stage 2	9	.61643d0	.5d-5
combined	9	.61643d0	.5d-5
Madsen	20	.61643d0	.67d-4

The stagel method is essientially the method of Madsen. The convergence of stagel is linear as second order information is not considered as shown in table II. The effect of Λ is shown in table III. The algorithm is very sensitive to the underflow criteria used in Lemke algorithm. Using 1.0d-15 we do not get a solution for problem 1. We need to use 1.0d-16.

TABLE II

COMPARISON OF STAGE1 AND STAGE2 CONVERGENCE RATE

Iteration No.	Stage 1		Stage 2	
	F	| h |	F	| h |
1	.13d2	.12d1	.13d2	.12d1
2	.399d1	.11d1	.399d1	.1098d1
3	.1788d1	.55d0	.2244d1	.1098d1
4	.851d0	.55d0	.1291d1	.1098d1
5	.851d0	.14d0	.796d0	.350d0
6	.743d0	.14d0	.635d0	.191d0
7	.644d0	.27d0	.61659d0	.14d-1
8	.644d0	.68d-1	.61643d0	(<=).5d-5
9	.627d0	.68d-1		
10	.619d0	.68d-1		
27	.61643d0	(<=).5d-5		

TABLE III

COMPARISON OF CONVERGENCE RATES USING DIFFERENT Λ

	No. of iterations	F
.5	15	.61643d0
.75	slow convergence	
1.0	13	.71249d0*
1.2	8	.61643d0
1.5	slow convergence	

As can be seen the value of the initial restriction on stepsize is important. Using an inaccurate quadratic line search only when the function value increases improves the convergence properties considerably as shown in table IV. This is especially true when the convergence is very slow. In table IV convergence is obtained in a smaller number of iterations than in table III. However, the value of Λ is still important. This is because the function that is "active" initially may not be the same for different initial conditions. Using a line search for this problem has improved the rate of convergence for all the values of Λ.

TABLE IV

COMPARISON OF CONVERGENCE RATES FOR DIFFFERENT Λ USING LINE SEARCH

	No. of iterations	F
.5	15	.61643d0
.75	14	.61643d0
1.0	9	.61643d0
1.2	8	.61643d0
1.5	8	.61643d0

Example 2.

The following nonlinear programming problem is considered by Hald and Madsen [33] and by Charalambous and Conn [8].

Minimise $$f(x) = (x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2+10x_5^6+7x_6^2+x_7^4-4x_6x_7-10x_6-8x_7+1000.$$

subject to $$g_2(x) = -2x_1^2-3x_2^4-x_3-4x_4^2-5x_5+127 > 0$$

$$g_3(x) = -7x_1-3x_2-10x_3^3-x_4+x_5+282 > 0$$

$$g_4(x) = -23x_1-x_2^2-6x_6^2+8x_7+196 > 0$$

$$g_5(x) = -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3 - 5x_6 + 11x_7 > 0$$

This transformed to the minmax problem as follows

Minimise $\quad f_j(x) \qquad j = 1, \ldots, 5$

where $\quad f_j = f - 10g(j) \qquad j = 2, \ldots, 5$

and $\quad f_1 = f$

Note that a large constant (1000) is introduced so that the convergence is to the maximum positive value of F. Using

$x = (3, 3, 0, 5, 1, 3, 0)$

and $\Lambda = 0.5$

We make the following comparison.

TABLE V

COMPARISON OF NUMBER OF ITERATIONS TO SOLVE PROBLEM 2

	No. of Iterations	F	$\|\| h \|\|$
Stage 1	16	.69864d3	.5d-3
Stage 2	14	.68063d3	.5d-5
Combined	15	.68063d3	.5d-5
Hald & M.	23	.68063d3	.5d-5
Char.& Conn	150	.68063d3	.5d-5

The solution is x = [2.33050, 1.95137, -0.47754, 4.36573, -.62449, 1.03813, 1.59423]

F = [680.63, 680.63, -1844.987, -728.1519, 680.63]

The effect of Λ is as shown in table VI. The number of iterations obtained using a line search is shown in table VII.

TABLE VI

COMPARISON OF CONVERGENCE RATES USING DIFFERENT Λ

	No. of iterations	F
.5	14	.68063d3
.75	very slow convergence	
1.0	5	.691898d3
1.2	very slow convergence	

TABLE VII

COMPARISON OF CONVERGENCE RATES FOR DIFFERENT Λ USING LINE SEARCH

	No. of iterations	F
.5	8	.68063d3
.75	very slow convergence	.1591d4
1.0	very slow convergence	.68998d3
1.2	very slow convergence	.91460d3
1.5	very slow convergence	.68755d3

Using the line search improved the convergence for Λ = 0.5. However, the line search did not greatly improve the convergence in other cases because the function that is "active" initially is not an "active" function in the final convergence. Also only a slow decrease in the active function was noticed. Hence improving the initial active function does not improve the rate of convergence rapidly. Use of a cubic interpolation in the line search improved the convergence rate.

Example 3. The Rosen-Suzuki problem [77] is considered.

Minimise $f(x) = x_1^2+x_2^2+2x_3^2+x_4^2-5x_1-5x_2-21x_3+7x_4+100.$

subject to $g_1(x) = -x_1^2-x_2^2-x_3^2-x_4^2-x_1+x_2-x_3+x_4+8 > 0$

$g_2(x) = -x_1^2-2x_2^2-x_3^2-2x_4^2+x_1+x_4+10 > 0$

$g_3(x) = -x_1^2-x_2^2-x_3^2-2x_1+x_2+x_4+5 > 0$

The same transformation described in example 3 is used. The initial value of x = (0, 0, 0, 0), and Λ = 0.5. The solution is x = (0, 1, 2, -1) and F = (44, 44, 54, 44). Table shown below shows the effect of Λ . The results obtained by using a line search when the function value increases is shown in table X. Using line search greatly improved the convergence rate in this problem.

TABLE VIII

COMPARISON OF NUMBER OF ITERATIONS TO SOLVE PROBLEM 3

	No. of iterations	Function value	$\|\| h \|\|$
Stage 1	45	.5600372d2	.5d-4
Stage 2	9	.56d2	.5d-5
combined	11	.56d2	.5d-5
H. & M.	16	.56d2	.5d-5
C. & C.	37	.56d2	.5d-5

TABLE IX

COMPARISON OF CONVERGENCE RATES USING DIFFERENT Λ

	No. of iterations	F
.5	9	.44d2
.75	not conv. in 35 iter.	
1.0	not conv. in 40 iter.	
1.2	not conv. in 29 iter.	

TABLE X

COMPARISON OF CONVERGENCE RATES FOR DIFFERENT Λ USING LINE SEARCH

	No. of It.	F
.5	9	.44d2
.75	11	.44d2
1.0	35	.43997d2
1.2	8	.44d2

Example 4.

The problem used by Charalambous and Conn [8] is considered.

$$f_1(x) = x_1^4 + x_2^2$$

$$f_2(x) = (2 - x_1)^2 + (2 - x_2)^2$$

$$f_3(x) = 2 \exp(-x_1 + x_2)$$

$x^* = [1.13903, .89956]$, $F^* = [1.95222, 1.95222, 1.57409]$.

The initial value of $x = (1, -0.1)$ and Λ used $= 1.2$. The contours of the problem are shown in figure 2.

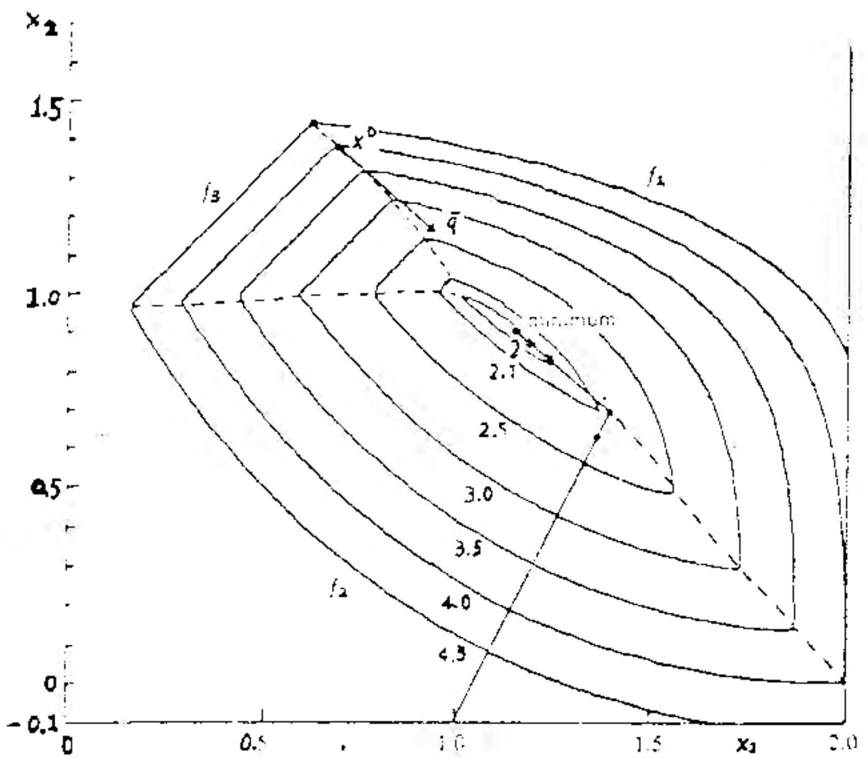

Figure 2. The contours of problem 6

TABLE XI

COMPARISON OF NUMBER OF ITERATIONS TO SOLVE PROBLEM 4

	No. of iterations	Function value	$\|\| h \|\|$
Stage 1	19	.19522d1	.5d-5
Stage 2	8	.19522d1	.5d-5
Combined	9	.19522d1	.5d-5
C. & C.	21	.19522d1	.5d-5

Table below showns the effect of Λ . A line search was not used as the convergence was quite fast in this problem.

TABLE XII

COMPARSION OF CONVERGENCE RATES USING DIFFERENT Λ

	No. of iterations	F
.5	8	.19522d1
.75	9	.19522d1
1.0	9	.19522d1
1.2	8	.19522d1

Example 5

$$f_1(x) = x_1^2 + x_2^4$$

$$f_2(x) = (2 - x_1)^2 + (2 - x_2)^2$$

$$x^* = [1,\ 1],\ F^* = [2,\ 2,\ 2].$$

The initial value of $x = (1, -0.1)$ and $\Lambda = 1.2$. The contours of the problem are shown in figure 3. Table XIV shows the effect of Λ. No line search is necessary as the convergence was rapid.

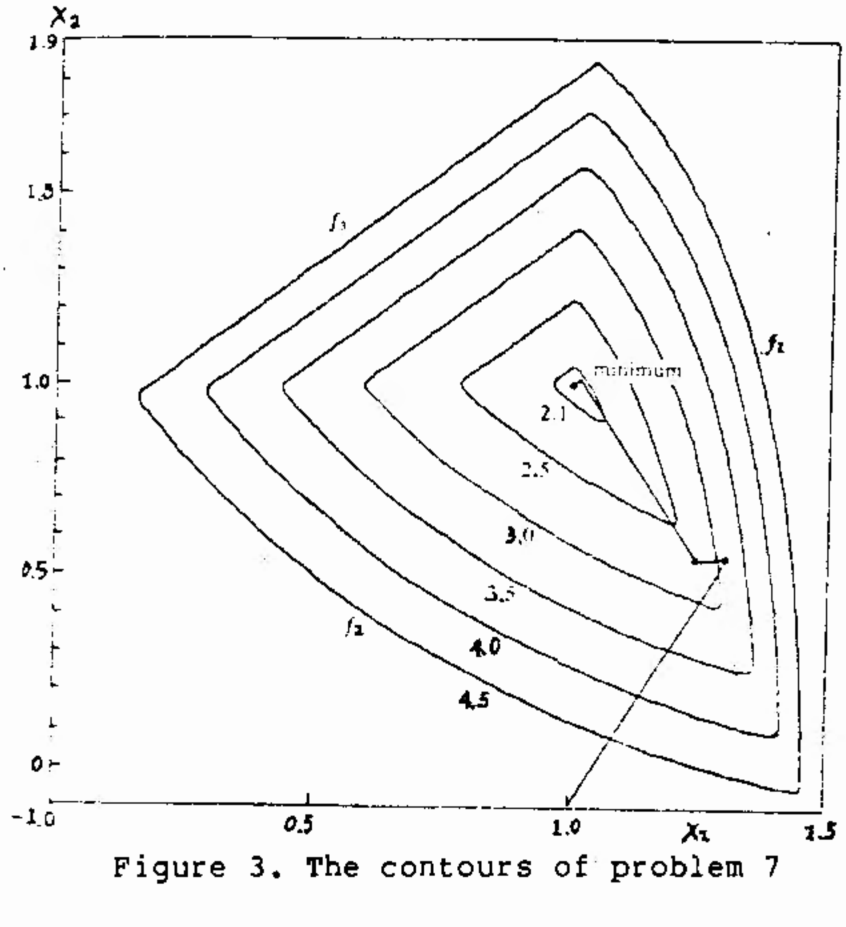

Figure 3. The contours of problem 7

TABLE XIII

COMPARISON OF NUMBER OF ITERATIONS USED TO SOLVE PROBLEM 5

	No. of iterations	F	$\|\| h \|\|$
Stage 1	6	.200000d1	.5d-5
Stage 2	5	.200000d1	.5d-5
Combined	5	.200000d1	.5d-5
C. & C.	8	.2d0000d1	.5d-5

TABLE XIV

COMPARISON OF CONVERGENCE RATES USING DIFFERENT Λ

	No. of iterations	F
.5	6	.200000d1
.75	6	.200000d1
1.0	6	.200000d1
1.2	6	.200000d1
1.5	6	.200000d1

CHAPTER V

SUMMARY

There is considerable interest in the development of algorithms for NDO problems, but it is not possible to say yet what the best approaches are. Most of the algorithms surveyed in chapter II have some common features.

Many methods are line search methods in which on each iteration a direction of search is determined and $x_{k+1} = x_k + \alpha_k h_k$ is obtained by choosing α_k to minimise the objective functions along a line. A typical line search algorithm uses a combination of sectioning and interpolation. An aspect to be considered is when the line search minimum is non-smooth. In this case it is not appropriate to try to make the stepsize small, since such a point may not exist. Fletcher [26] recommends a different test, that a line search is terminated when the predicted reduction is sufficiently small. This test has been used by Hald and Madsen for stagel iteration.

Most methods for NDO can be considered as extensions of methods available for smooth optimisation. The simplest method for basic NDO, the subgradient method, is an analogue of the steepest descent method. The method is at best linearly convergent. Similar algorithms using conjugate gradients are the bundle methods. The use of approximations to form linear and quadratic subprograms is another class of methods. Quasi-Newton methods have been used in conjuction with some of these methods to obtain faster convergence when the curvature effects cannot be neglected.

There is at present considerable interest in developing methods for NDO problems. The applications of NDO methods to practical problems in linear and nonlinear programming is being studied.

The method of Hald and Madsen [33] is an effective method for solving NDO problems. A modification of the method is studied in this thesis. The method as described by Fletcher [28] is used for stage 2 instead of a quasi-Newton method as suggested by Hald and Madsen. An inaccurate quadratic line search is used when the predicted value of the function increases. This increases the efficiency of the algorithm in most cases.

From the numerical evidence presented it can be seen that the choice of initial restriction Λ is very important. As noted before, the efficiency of the algorithm also depends upon the efficiency of the linear and quadratic programming method used. Using a line search improves the convergence properties in general. However when the initial active function is not a final active function, a line search for that function does not improve the rate of convergence rapidly.

The modified method has good convergence properties and may have wide application. It has proved to have equal or faster rate of convergence than the method of Hald and Madsen or that of Charalambous and Conn, for the problems considered in chapter IV.

CHAPTER VI

SUGGESTIONS FOR FURTHER STUDY

There is one feature of the method of Fletcher [26] that is different from similar methods for smooth optimisation, known as the Maratos effect. For smooth unconstrained optimisation when x_k is close to x^*, the basic method reduces the objective function and second order rate of the basic method is observed. However, as observed by Maratos [62], this does not happen in NDO. In some NDO problems, in which second order effects are significant at the solution, x_k can be arbitrarily close to x^* and the unit step of the basic algorithm can fail to reduce the function $F(x)$. This effect is most likely to occur when the discontinuity in derivative is large. Further studies in this area may greatly improve the application of the method to a general problem.

A further modification to the above algorithm is to use an updating procedure to obtain the next combined Hessian matrix.

BIBLIOGRAPHY

(1) Balinski, M.L. and Wolfe, P. Nondifferentiable Optimization, Math. Prog. Study 3 (1975), N.Holland, Amesterdam.

(2) Bandler, J.W. and Charalambous, C. "Nonlinear Programming Using Minimax Techniques." J. Opt. Th. and Appl., Vol. 13 (1974), 607-619.

(3) Bazaraa, M.S., Goode, J.F. and Shetty, C.M. "Optimality Criteria in Nonlinear Programming without Differentiability." Operations Res., Vol. 19 (1971), 77-86.

(4) Bazhenov, L.G. "On Convergence Conditions of a Minimization Method of almost Differentiable Functions." Cybernetics, Vol. 8 (1972), 607-609.

(5) Bertsekas, D.P. "Nondiffereentiable Optimization via Approximation." Nondifferentiable Optimization , Math. Prog., study 3 N.Holland, Amesterdam (1975)

(6) Bertsekas, D.P. and Mitter, S.K. "A Descent Algorithm for Optimization Problems with Nondifferentiable Cost Functionals." SIAM J. on Control, Vol. 11 (1973), 637-652.

(7) Broyden, C.G., Dennis, J.E. and More, J.J. "On the Local and Superlinear Convergence of Quasi Newton Methods." Math. Comput., Vol. 12 (1973), 223-246.

(8) Charalambous, C. and Conn, A.R. "An Efficient Method to Solve the Minimax Problem Directly." SIAM J. Num. Anal., Vol. 15 (1978), 162-187.

(9) Chaney, R. and Goldstein, A. "An Extension of the Method of Subgradients." Nonsmooth Optimization, Math. Proc., study 3 (1975), 31-50.

(10) Cheney, E.W. and Goldstein, A.A. "Newton's Method for Convex Programming and Chebyshev Approximation." Numerische Mathematik, (1959), 253-268.

(11) Clarke, F.H. "Generlised Gradients and Applications." T.A.M.S, 205 (1975), 247-262.

(12) Clarke, F.H. Optimization and Nonsmooth Analysis, Wiley, Newyork (1983).

(13) Coleman, T. F. and Conn, A. R. "Nonlinear programming via an Exact Penalty Function: Flobal Analysis." Univ. of Waterloo, Tech. Report CS-80-31 (1980).

(14) Cullum, J., Donath, W.E. and Wolfe, P. "The Minimization of Certain Non-differentiable Sums of Eigenvalues of Symmetric Matrices." Math. Prog., Study 3 (1975), 35-55.

(15) Danskin, J.M. The Theory of Max-min, Springer, New York (1967).

(16) Demjanov, V.F. "Algorithms for Some Minimax Problems." J. Comp. Sys. Sci., Vol. 2 (1968),342-380.

(17) Demjanov, V.F. and Malozemov, V.N. Introduction to Minimax, John Wiley, New York (1974).

(18) Demjanov, V.F. and Rubinov, A.M. Approximation Methods in Optimization Problems, American Elsevier, New York (1970).

(19) Dennis, J.E. and More, J.J. "Quasi-Newton Methods and Theory." SIAM Review, Vol. 19 (1977), 46-89.

(20) De Angelis, V. "Minimization of a Separable Function Subject to Linear Constraints." Proc. Princeton Symposium on Math. Prog., (1970) 503-510.

(21) Eaves, B.C. and Zangwill, W.I. "Generalised Cutting Plane Algorithms." Working paper No. 274, Center for Research in MGT. SC., Univ. of Ca., Berkeley, (1969).

(22) Elzinga, J. and Moore, T.G. "A Generalized cutting plane algorithm." Working paper No. 274, Center for Research in MGT. SC., Univ. of Ca., Berkeley, (1969).

(23) Fiacco, A.V. and McCormick, G.P. "Computational Algorithm for the Sequential Unconstrained Minimization Technique for Nonlinear Programming." MGT. SC. Vol. 10, No. 4 (1964).

(24) Fisher, M.L., Northup, W.D and Shapiro, J.F. "Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience." Math. Prog. Study 3, Amesterdam, N. Holland (1975).

(25) Fletcher, R. "Methods Related to Langrangian

Functions." Numerical Methods for Constrained Optimization, ed. Gill, P.E, Murray, W. (1974), 219-239.

(26) Fletcher, R. Pratical Methods of Optimization, Vol. I, Unconstrained Optimization, Wiley, New York (1980).

(27) Fletcher, R. Pratical Methods of Optimization, Vol. II, Unconstrained Optimization, Wiley, New York (1981).

(28) Fletcher, R. "A Model Algorithm for Composite Nondifferentiable Optimization Problems." Numerical Techniques in NDO, Math. Prog., study 17 (1982), N.Holland, Amesterdam.

(29) Fletcher, R. "A New Approach to Variable Metric Algorithms." Comp. J., Vol. 13 (1970).

(30) Fletcher, R. and Watson, G. A. "First and Second Order Conditions for a Class of NDO Problems." Math. prog., Vol. 18 (1980), 291-307.

(31) Gill, P.E., Murray, W. and Wright, M. Practical Optimization, Academic Press, New York.

(32) Goffin, J.L. "On Convergence Rates of Subgradient Optimization Methods." Math. Prog., Vol. 13 (1977), 329-347.

(33) Goffin, J.L. "Nondifferentiable Optimization and the Relaxation Method." N.S. Optimisation, Proc. IIASA Workshop (1977), 5-30.

(34) Hald, J. and Madsen, K. "Combined LP and Quasi Newton Methods for Minimax Optimization." Math. Prog., Vol. 20 (1981), 49-62.

(35) Hald, J. and Madsen, K. "A 2-stage Algorithm for Minimax Optimization." Lecture notes in Control and Inf. Science 14 (1978), 225-239.

(36) Hald, J. and Schjaer-Jacobsen, H. "Linearly Constrained Minimax Optimization." Math. Prog., Vol. 14 (1978), 208-223.

(37) Han, S.P. "Dual Variable Metric Methods for Constrained Optimisation." SIAM J. on Control and Optimization, Vol. 15 (1977), 546-565.

(38) Han, S.P. "A Globally Convergent Method for Nonlinear Programming." J. of Opr. Theory and Appl., Vol.

22 (1977), 297-309.

(39) Han, S.P. "Superlinearly Convergent Variable Metric Algorithm for General Nonlinear Programming Problems." Math. Prog., Vol. 11 (1976), 263-282.

(40) Han, S.P. "Variable Metric Methods for Minimizing a Class of Nondifferentiable Functions." Math. Prog., Vol. 20 (1981), 1-13.

(41) Han, S.P. "A Hybrid method of Nonlinear Programming." N. L. Prog., 3 (1977), 297-309.

(42) Held, M., Wolfe, P. and Crowder, H. "Validation of Subgradient Optimization." Math. Prog., Vol. 6 (1974), 62-88.

(43) Hestenes, M.R. "Multiplier and Gradient Methods." J. Opt. Theo. Applns., 4 (1969), 303-320.

(44) Hestenes, M.R. and Stiefel, E. "Methods of Conjugate Gradients for Solving Linear Systems." J. of Research of the National Bureau of Standards, 49 (1952), 409-436.

(45) Hogan, W.W., Marsten, R.E. and Blankenship, J.W. "Boxstep: a New Strategy for Large Scale Mathematic Programming." Discussion paper No. 46, N. W. Univ., Evanston, IU (1973).

(46) Hooke, R. and Jeeves J.A. "Direct Search Solution of Numerical and Statistical Problems." J. Assoc. Compt. Mach., Vol. 8 (1961), 212.

(47) Kelley, J.E. "The Cutting Plane Method for Solving Convex Programs." J. Soc. Ind. Appl. Math., Vol. 8 (1960), 703-712.

(48) Kiwiel, K.C. "An Aggregate Subgradient Method for Nonsmooth Convex Minimzation.", Math. Prog., Vol. 27 (1983), 320-341.

(49) Kiwiel, K.C. A "V.M.M. of Centers for Nonsmooth Minimization." CP-81-23 Ins. of App. Syst. Anal., Laxemburg, Austria (1981).

(50) Kiwiel, K.C. "Efficient Algorithm for Nonsmooth Optimization and their Applications." Ph.D thesis, Dept. of Electronics, Univ. of Warsaw, Warsaw, Poland.

(51) Lemarechal, C. "An Extension of Davidon Methods to Nondifferentiable Problems. "Nondifferentiable

Optimization, Mathematical Programming Study 3, N. Holland, Amesterdam. (1975).

(52) Lemarechal, C. "Note on an Extension of Davidon Methods to Nondifferentiable Functions." Math. Prog., Vol. 7 (1974), 384-387.

(53) Lemarechal, C. "Bundle Methods in Nonsmooth Optimization." Nonsmooth Optimization, Proc. IIASA workshop (1977), Pergamon Press, Oxford.

(54) Lemarechal, C. and Mifflin, R. Nonsmooth Optimization, Proc. IIASA workshop (1977), Pergamon Press, Oxford.

(55) Lemarechal, C. "Nondifferentiable Optimization: Subgradient and ϵ-subgradient Methods." Lecture Notes in Eco. and Math. Systems, No. 117 (1975).

(56) Leunberger, D.G. Introduction to Linear and Nonlinear Programming, Addison-Wesley, Reading, Mas. (1974).

(57) Madsen, K. "An Algorithm for Minimax Solution of Overdermined Systems of Nonlinear Equations." J. Inst. Math. Appl., Vol. 16 (1975), 321-328.

(58) Madsen, K. "Minimax Solution of Nonlinear Equations Without Calculating Derivatives." Nondifferentiable Optimization, Math. Prog., Study 3 (1975), 110-126.

(59) Madsen, K. and Schjaer-Jacobsen, H. "Linearly Constrained Minimax Optimization." Math. Prog., Vol. 14 (1978), 208-223.

(60) Madsen, K. and Schjaer-Jacobsen, H. "Singularities in Minimax Optimization of Networks." IEEE Trans. on Circuits and Systems (1976), 456-460.

(61) Nelder, J.A. and Mead, R. "A Simplex Method for Function Minimization." Compt. J., Vol. 7 (1965), 308-313.

(62) Marsten, R.E. "The Use of Boxstep Method in Discrete Optimization." M. P., study 3 (1975).

(63) Maratos N. "Exact Penalty Function Algorithms for Finite Dimensional and Control Optimization Problems." Ph.D thesis, Unvi. of London (1980).

(64) Mayne, B.Q. "On the Use of Exact Penalty Functions to Determine Step Length in Optimization

Algorithms." Numerical Analysis, Dundee, Lecture Notes in Mathmetic 773, Springer-Verlag, Berlin (1980).

(65) Mifflin, R. "An Algorithm for Constrained Optimization with Semi-smooth Functions." Math. Oper. Res., Vol. 2 (1977), 191-207.

(66) Mifflin, R. "A Modification and an Extension of Lemarechal's Algorithm for Nonsmooth Minimization." Numerical Techniques for Nondifferentiable Optimization, Math. Prog., Study 17 (1982), 77-90.

(67) Mifflin, R. "Semismooth and Semiconvex Functions in Constrained Optimization." SIAM J. on Control and Optimization, Vol. 15 (1976), 959-972.

(68) Murtagh, B.A. and Soliman, F.I. "Subgradient Optimization Applied to a Discrete Nonlinear Program in Engineering Design." Math. Prog., Vol. 25 (1983), 1-12.

(69) Nurminskii, E.A. "Minimization of Nondifferentiable Functions in Presence of Noise." Kiberntics, Vol. 10 (1974), 59-61.

(70) Nurminskii, E.A. "The Quasi-Gradient Method for Solving of the Nonlinear Programming Problems." Cybernetics, Vol. 9 (1973), 145-150.

(71) Ortega, J.M. and Rheinboldt, W.C. Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, New York.

(72) Osborne, M.R. and Watson, G.A. "An Algorithm for Minimax Approximation in Nonlinear Case." Computer J., Vol. 12 (1969), 64-69.

(73) Poljak, B.T. "Subgradient Methods: A Survey of Soviet Research." Nonsmooth Optimisation, Proc. IIASA Workshop (1977), Pergamen Press, Oxford.

(74) Powell, M.J.D. "A Fast Algorithm for Nonlinear Constrained Optimization Calculations." Dundee Conference on Numerical Analysis, (1977).

(75) Pschenchinyi, B.N." Nonsmooth Optimization and Nonlinear Programming." Nonsmooth Optimization, Proc. IIASA Workshop (1977), 71-78.

(76) Rockafellar, R.T." Lagrange Multipliers and Subderivatives of Optimal Value Functions in

Nonlinear Programming." Math. Prog., Study 17 (1982), 28-66.

(77) Rosen J.B. and Suzuki, S. "Construction of Nonlinear Programming Test Problems." Comm. ACM, Vol. 8 (1965), 113-120.

(78) Rosenbrock H.H. "An Automatic Method for Finding the Greatest or Least Value of a Function." Comp. J., Vol. 3 (1960), 175.

(79) Sachs, E. "Global Convergence of Quasi Newton Type Algorithm for some Nonsmooth Optimisation Problems." J. Optimisation Theory Appl., Vol. 40 (1983).

(80) Shor, N.Z. and Shabashova, L.P. "Solution of Minimax Problems by Generalized Gradient Method with Space Dilation." Kebernetika 8, 1 (1972), 82-88.

(81) Shor, N.Z. "On the Structure of Algorithms for the Numerical Solution of Problems of Optimal Planning and Design." Dissertation, KIEU, USSR (1964). (in Russian).

(82) Shor, N.Z. and Zhurbenro, N.G. "A Minimization Method Using Space Dilation in the Direction of the Difference of Two Successive Gradients." Cybernetics, Vol. 7 (1971), 450-459.

(83) Spendly, W., Hext, G.R., and Hemsworth, F.R. "Sequential Application of Simplex Designs in Optimization and Evaluationary Operation." Technimetrice, 4, 441-461.

(84) Tewarson, R.P. "On Minimax Solutions of Linear Equations." Comp. J., Vol. 15 (1972), 277-279.

(85) Watson, G.A. "The Minimax Solution of an Overdetermined System of Nonlinear Equations." J. Inst. Math. Appl., Vol. 23 (1979), 167-180.

(86) Wierzbicki, A.P. "Lagrangian Functions and Nondifferentiable Optimization." WP-78-63, International Inst. for App. Syst. Analysis, Laxemburg, Austria (1978).

(87) Wolfe, P. "Note on a Method of Conjugate Subgradients for Minimisation of Nondifferentiable Functions." Math. Prog., Vol 7 (1974), 380-383.

(88) Wolfe, P. "A Method of Conjugate Subgradients for Minimisation of Nondifferentiable Functions."

Math. Prog., Study 3, N. Holland, Amesterdam (1975).

(89) Wolfe, P. "Sufficient Minimization of Piecewise Linear Univariate Functions." N. S. Optimization, Proc. of IIASA Workshop (1977), 103-126.

(90) Wolfe, P. "A Method of Conjugate Subgradients for Minimizing Nondifferentiable Functions." Math. Prog. Study 3 (1975), 145-173.

(91) Wolfe, P. "Convergence Theory in Nonlinear Programming." Integer and Nonlinear Programming, ed. Abadie, (1970).

(92) Womersley, R.S. "Optionality Conditions for Piecewise Smooth Functions." Math. Prog., Study 17 (1982), 13-27.

(93) Zangwill, W.I. "Nonlinear Programming via Penalty Functions." Mgt. Sci., Vol. 13 (1967), 344-358.

(94) Zangwill, W.I. Nonlinear Programming. A Unified Approach, Prentice Hall, New Jersey (1969).

APPENDIX A

DEFINITIONS

Definition 1

The Lipschitz Condition

Let Y be a subset of X. A function f: Y -- R is said to satisfy a Lipschitz condition (on y) provided that, for some nonnegative scalar K, one has

$$| f(y) - f(y') | <= K \| y - y' \|$$

for all points y, y' in Y; that is also referred to as a Lipschitz condition of rank K.

Definition 2 .

The Kuhn-Tucker Conditions

The Kuhn Tucker conditions for the nonlinear programming problem,

$$\begin{array}{lll} \text{minimise} & f(x) & \\ \text{subject to} & C_i(x) = 0 & i \in E \\ & C_i(x) >= 0 & i \in I \end{array}$$

is described below.

If x* is a local minimiser of the above problem, then there exist Lagrange multipliers λ* such that x*, λ* satisfy the following system.

$$L(x, \lambda) = 0$$

$$C_i(x) = 0 \qquad i \in E$$

$$C_i(x) >= 0 \qquad i \in I$$

$$\lambda_i >= 0 \qquad i \in I$$

$$\lambda_i C_i(x) = 0 \qquad \forall i$$

The above conditions are valid when the vectors a_i^*, $i \in A$ are independent, where $a_i = \partial C_i$. The final condition $\lambda^* C^* = 0$ is referred to as the complementarity condition and states that both λ_i and C_i cannot be nonzero, or equivalently that inactive constraints have a zero multiplier. If there is no i such that $\lambda_i^* = C_i^* = 0$ then strict complementarity is said to hold. The case $\lambda_i^* = C_i^* = 0$ is an intermediate state between a constraint being strongly active and being inactive.

Definition 3

Regular and Singular Minimax Problem

The minimax problem is singular with respect to the solution x* if the matrix

$$D = \{ \partial f_j / \partial x_i (x^*) \} \qquad j \in A$$
$$i = 1, \ldots, n$$

has rank less than n. Otherwise the problem is regular.

Note : "A" denotes the acture set which consists of the index of the functions that attain the maximum value at x^*.

Definition 4

Haar Condition

Haar Condition is satisfied when any subset of the set

$$\{ f'(x^*) \mid f(x^*) = F(x^*) \}$$

has maximal rank. This ensures that no smooth valley passes through the solution.

APPENDIX B

THE SUMMARY OF DIFFERENT METHODS

TABLE XV

NUMBER OF ITERATIONS FOR DIFFERENT METHODS

	Prob. 1	Prob. 2	Prob 3.	Prob. 4	Prob. 5
Stage 1	27	16	45	19	6
Stage 2	9	14	9	8	5
Stage	9	15	11	9	5
H. & M.	20*	23	16	-	-
C. & C.	-	150	37	21	8

* Line search did not improve the rate of convergence.

TABLE XVI

NUMBER OF ITERATIONS FOR DIFFERENT Λ

	Prob. 1		Prob. 2		Prob. 3		Prob. 4.		Prob. 5	
		using LS		using LS		using LS		using LS		using LS
.5	15	15	14	8	9	9	8	*	6	*
.75	slow	14	slow	slow	slow	11	9	*	6	*
1.0	13	9	5	slow	slow	35	9	*	6	*
1.2	8	8	slow	slow	slow	8	8	*	6	*
1.5	slow	8	–	–	–	–	–	–	–	–

* Line search did not improve the rate of convergence.

APPENDIX C

PROGRAM LISTING

```
$JOB
C            MODIFIED HALD AND MADSEN ALGORITHM
C                          FOR
C                MINIMAX OPTIMISATION
C
C
C********************** REFERENCES *************************
C      1.    HALD,J  AND  MADSEN,K "COMBINED LP AND QUASI NEWTON
C                   METHODS FOR MINIMAX OPTIMIZATION" MATH. PROG.
C                   20,(1981).
C      2.    FLETCHER,R. "A MODEL ALGORITHM FOR COMPOSITE NDO
C                   PROBLEM", MATH. PROG. STUDY 17, (1982).
C**********************************************************
C
C      THIS IS A PROGRAM FOR SOLVING MINIMAX PROBLEMS USING
C      LINEAR AND QUADRATIC APPROXIMATIONS
C
C******************** VARIABLE REFERENCE *******************
C
C      FAPR     : APPROXIMATE FUNCTION VALUE
C      F(*)     : FUNCTION VALUE
C      G(*,*)   : HESSIAN MATRIX
C      GLM(*)   : LAGRANGE MULTIPLIERS ASSOCIATED WITH EACH FUNCTION
C      H(*)     : STEPSIZE
C      K1       : NUMBER OF ITERATIONS IN STAGE 1
C      K2       : NUMBER OF ITERATIONS IN STAGE 2
C      XJ(*,*)  : JACOBIAN FOR DERIVATIVES
C      XLAMDA   : RESTRICTION ON STEPSIZE H
C
C
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(20)
      COMMON /STAGE/ISTAGE,K1,K2,NBUG
      COMMON /STG/HMAX,PREFOB
C
C       READ IN THE NUMBER OF PROBLEMS TO BE SOLVED
C
      IOUT=6
      IN=5
C
      NO=6
      NBUG=0
      WRITE(IOUT,20) NO
   20 FORMAT(1H0,10X,17H NO OF PROBLEMS= ,I2)
C
C       OBTAIN INITIAL VALUES FOR X AND LAMDA AND
C                CALL STAGE1.
C
      DO 30 NOPROB=1,NO
      WRITE(IOUT,25) NOPROB
   25 FORMAT(1H1,14H**PROBLEM NO: ,I2)
      CALL INITIL (NOPROB,X,N,XLAMDA)
      K1=0
      K2=0
      ISTAGE=2
      CALL STAGE1(NOPROB,X,N,XLAMDA)
   30 CONTINUE
      STOP
      END
```

```
C
C
C      SUBROUTINE INITIALISES THE X VALUES
C
C
       SUBROUTINE INITIL(NOPROB,X,N,XLAMDA)
C
C
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION X(20)
       GOTO (10,20,30,40,20,60,70),NOPROB
   10  N=2
       X(1)=3.D0
       X(2)=1.D0
       XLAMDA=1.2D0
       GOTO 1000
   20  N=2
       X(1)=-1.2D0
       X(2)=1.D0
       XLAMDA=.5D0
       GO TO 1000
   30  N=7
       X(1)=3.D0
       X(2)=3.D0
       X(3)=0.D0
       X(4)=5.D0
       X(5)=1.D0
       X(6)=3.D0
       X(7)=0.D0
       XLAMDA=0.50D0
       GO TO 1000
   40  N=4
       X(1)=0
       X(2)=0
       X(3)=0
       X(4)=0
       XLAMDA=0.50D0
C
       GO TO 1000
   60  M=3
       N=2
       X(1)=1
       X(2)=-0.1
       XLAMDA=0.750D0
       GO TO 1000
C
   70  GO TO 1000
C
 1000  RETURN
       END
```

```
C
C
C     SUBROUTINE CARRIES OUT THE STAGE1 AND STAGE2 ITERATIONS
C
C
      SUBROUTINE STAGE1(NOPROB,X,N,XLAMDA)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION XNXT(20),X(20),F(20),FNXT(20),H(20),GLM(20),GLA(20)
      DIMENSION XJ(20,20),SUM(20)
C
      COMMON /SWTH/RESDUL(20),RPRE(20),R(20),SUM,SUMLGM
      COMMON  /ACTIV/ AS(3,20),NPTR,NXTPTR,NS(3)
      COMMON /STAGE/ ISTAGE,K1,K2,NBUG
      COMMON /STG/HMAX,PREFOB
C
      ECONV=0.5D-3
      HCONV=0.5D-5
      NPTR=1
      NXTPTR=1
      C1=1.0D-2
      C2=2.5D-1
      C3=7.5D-1
      C4=2.5D-1
      C5=2.D0
      C6=1.D0
      J0=0
C
C     DETERMINE ITERATION NO. K, THE FUNCTION VALUES AND JACOBIAN.
C
      DO 2 I=1,20
      GLM(I)=0.D0
    2 CONTINUE
      CALL FUNCTN(NOPROB,X,N,F,M,J0)
      CALL MAX(F,M,FMAX)
      FOBJ=FMAX
      CALL DERIV(NOPROB,X,N,XJ,M)
C
C     DETERMINE THE ACTIVE SET OF FUNCTIONS.
C
      PREFOB=0.D0
    5 K1=K1+1
      IF(K1.GT.35) GOTO 1000
      IF (ISTAGE.EQ.2)K2=K2+1
      WRITE(IOUT,3) K1,K2
    3 FORMAT(///' ','****ITERATION NUMBER=',I3,I3,'****')
      CALL ACTIVE(FOBJ,F,M)
C
C     DETERMINE THE COEFFICIENTS AM AND Q FOR LEMKE'S ALGORITHM.
C
      CALL COEFF(F,M,XJ,N,XLAMDA,MN,GLM,X,NOPROB)
      IFLAG=0
      IF (NBUG.EQ.1) WRITE(IOUT,6) (X(I),I=1,N)
    6 FORMAT(1H0,1X,2HX=,10(E12.5))
      IF (NBUG.EQ.1) WRITE(IOUT,8) (F(I),I=1,M)
    8 FORMAT(1H0,1X,2HF=,10(E12.5))
      IF (NBUG.EQ.1) WRITE(IOUT,9) (GLM(I),I=1,M)
    9 FORMAT(1H0,4HLGM=,10(E12.5))
C
C     CALL LEMKE
C
      CALL LEMKE(MN,IFLAG)
C
C     DETERMINE FUNCTION VALUE PREDICTED BY LP(FARP), INCREMENT(H )
C     AND LAGRANGE MULTIPLIERS(LMG)                               K
C
      CALL HVAL(H,N,GLA,FAPR,F,M,XJ)
      IF (NBUG.EQ.1) WRITE(IOUT,7) FAPR
    7 FORMAT(/1X,'FAPR=',E15.5)
```

```
C
C     DETERMINE FUNCTION VALUE AT X +H
C                                  K  K
C
      INT=0
      IHFLG=0
      DO 10 I=1,N
      IF(DABS(H(I)).LT.(2.D0*XLAMDA).OR.DABS(H(I)).LT.1.D1)GOTO 11
      IF(H(I).LT.0.D0) H(I)=-2.D0*XLAMDA
      IF(H(I).GT.0.D0) H(I)=2.D0*XLAMDA
   11 XNXT(I)=X(I)+H(I)
   10 CONTINUE
      IF(IHFLG.EQ.1) GO TO 44
      CALL FUNCTN(NOPROB,XNXT,N,FNXT,M,J0)
      CALL MAX(FNXT,M,FMAX)
      FOBNXT=FMAX
      IF (NBUG.EQ.1) WRITE(IOUT,21)FOBNXT
   21 FORMAT(/1X,'FOBNXT=',E15.5)
      IF (NBUG.EQ.1) WRITE(IOUT,22)FOBJ
   22 FORMAT(/1X,'FOBJ=',E15.5)
      CALL ACTIVE(FOBJ,F,M)
C
C     DETERMINE  F      F(X )-F(X ,H )
C                          R     R  K
C
      DIFFK=FOBJ-FOBNXT
      IF(DIFFK.GE.0.D0) GO TO 15
      IF(K1.GT.3) GO TO 890
      J0=K1
      GO TO 891
  890 J0=NS(1)
  891 CALL LINSCH(NOPROB,J0,FOBJ,FOBNXT,X,H,N)
      DO 990 I=1,N
      XNXT(I)=X(I)+H(I)
  990 CONTINUE
      J0=0
      CALL FUNCTN(NOPROB,XNXT,N,FNXT,M,J0)
      CALL MAX(FNXT,M,FMAX)
      FOBNXT=FMAX
      DIFFK=FOBJ-FOBNXT
   15 IF (NBUG.EQ.1) WRITE(IOUT,23) DIFFK
   23 FORMAT(' ','DIFFK1',E15.5)
C
C     TEST FOR CONVERGENCE.
C
      DO 14 I=1,N
      IF (NBUG.EQ.1) WRITE(IOUT,991) H(I)
  991 FORMAT(' ','NEW H',E15.5)
      H(I)=DABS(H(I))
   14 CONTINUE
      CALL MAX(H,N,HMAX1)
      HMAX=HMAX1
      IF(HMAX.LE.HCONV) GOTO 1000
      IF(HMAX.GT.HCONV) GOTO 30
      GOTO 1000
C
C     DETERMINE F(X )=FAPRX
C                  K
C
   30 DIFAPR=FOBJ-FAPR
      RATIO=DIFFK/DIFAPR
      IF (NBUG.EQ.1) WRITE (IOUT,32) RATIO
   32 FORMAT(/1X,'RATIO=',E15.5)
C
C     CHANGE X,F,LGM IF LINEARISATION IS GOOD.
C
      IF(RATIO.LE.1.D0)GO TO 31
      IF(RATIO.GT.1.99D0) GO TO 38
   31 IF(RATIO.LT.C1) GOTO 38
      DO 35 I=1,M
      F(I)=FNXT(I)
      GLM(I)=GLA(I)
   35 CONTINUE
```

```
      PREFOB=FOBJ
      FOBJ=FOBNXT
      DO 40 I=1,N
      X(I)=XNXT(I)
   40 CONTINUE
      CALL DERIV(NOPROB,X,N,XJ,M)
C
C     CALL SWITCH IF ACTIVE SET IS LT N+1
C
   38 IF(K1.LT.3) GO TO 41
      IF (NBUG.EQ.1) WRITE(IOUT,1111) NS(NPTR),N
 1111 FORMAT(' ','NS(NPTR),N',2I4)
      IF(NS(NPTR).GT.N)GO TO 41
C     CALL SWITCH(XLAMDA,GLM,HMAX,N,F,XJ,FOBJ,M)
C
C     ELSE X REMAINS UNCHANGED. DETERMINE NEXT LAMDA.
C
   41 EDIF=.01D0*XLAMDA
C
      IF(RATIO.GT.C2) GOTO 42
      IF (RATIO.LT.1.75D0) GO TO 42
      XLAMDA=C4*HMAX
      IF (NBUG.EQ.1) WRITE(IOUT,999) XLAMDA
  999 FORMAT(' ','XLAMDA',E15.5)
      GOTO 5
   42 IF(RATIO.LT.C3) GOTO 50
      IF(RATIO.GT.1.25) GO TO 50
      DIFHL=(RATIO-1)
      IF(DIFHL.GT.ECONV) GO TO 50
      XLAMDA=4*XLAMDA
      GO TO 5
      IF(HMAX.NE.XLAMDA) GO TO 50
   44 XLAMDA=C5*XLAMDA
      IF (NBUG.EQ.1) WRITE(IOUT,43) DIFFK
   43 FORMAT(' ','DIFFK2',E15.5)
      IF(DIFFK.LT.0.D0) XLAMDA=C4*HMAX
      GOTO 5
   50 XLAMDA=C6*HMAX
      IF (NBUG.EQ.1) WRITE(IOUT,51) HMAX
   51 FORMAT(' ','HMAX-XLAMDA',E15.5)
      GOTO 5
 1000 WRITE(IOUT,6) (X(I),I=1,N)
      WRITE(IOUT,8) (F(I),I=1,M)
      WRITE(IOUT,300)
  300 FORMAT(/15X,'***CONVERGENCE***')
      RETURN
      END
```

```
C
C
C     THIS SUBROUTINE SWITCHES THE STAGES DEPENDING ON THE
C     EXISTING CONDITIONS
C
C
      SUBROUTINE SWITCH(XLAMDA,GLM,HMAX,N,F,XJ,FOBJ,M)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION GLM(20),F(20),XJ(20,20),XJM(20)
      DIMENSION XJL(20),SUM(20),XJMX(20)
C
      COMMON /SWTH/RESDUL(20),RPRE(20),R(20),SUM,SUMLGM
      COMMON /STAGE/ISTAGE,K1,K2,NBUG
      COMMON  /ACTIV/ AS(3,20),NPTR,NXTPTR,NS(3)
C
C
      NUM=NS(NPTR)
      INDX=AS(NPTR,1)
      R(1)=F(INDX)
      IF(NUM.LE.1) GO TO 8
      DO 15 I=2,NUM
      I1=I-1
      R(I1)=F(INDX)-F(I)
   15 CONTINUE
C     IF STAGE=1 TEST CONDITIONS TO SWITCH TO STAGE 2
C
C    A) TEST IF  LAMDA-1>=0,LAMDA>=0
C
    8 GO TO (10,100),ISTAGE
   10 SUMLGM=0.D0
      DO 20 I=1,NUM
      INDX=AS(NPTR,I)
      IF (GLM(INDX).LT.0.D0) GO TO 1000
      SUMLGM=SUMLGM+GLM(INDX)
      IF (ISTAGE.EQ.2) GO TO 31
   20 CONTINUE
      SUMDIF=SUMLGM-1.D0
      EDIF=.1D-2
      IF (SUMDIF.GT.EDIF) GO TO 1000
CC    GO TO 31
C
C    B) TEST IF ||H||=LAMDA
C
      DIFHL=HMAX-XLAMDA
      IF (DIFHL.GT.EDIF) GO TO 1000
C
C    C) TEST IF A(1,S1)=A(2,S2)=A(3,S3)
C
   25 IF (NS(1).NE.NS(2)) GO TO 1000
      IF (NS(2).NE.NS(3)) GO TO 1000
      DO 30 I=2,3
      DO 30 J=1,NUM
      IF (AS(I,J).NE.AS(1,J)) GO TO 1000
   30 CONTINUE
C
C    D) TEST IF LAMDA.J<=E2
C
   31 DO 32 J=1,N
      SUM(J)=0.D0
   32 CONTINUE
      DO 40 I=1,NUM
      NI=AS(NPTR,I)
      DO 35 J=1,N
      XJM(J)=XJ(NI,J)
      XJL(J)=GLM(NI)*XJM(J)
      SUM(J)=SUM(J)+XJL(J)
      XJM(J)=DABS(XJM(J))
   35 CONTINUE
```

```
      IF(ISTAGE.EQ.2) GO TO 200
      GO TO 71
      CALL MAX(XJM,N,XJMAX)
      XJMX(I)=-XJMAX
   40 CONTINUE
      SQSUM=0.D0
      DO 50 J=1,N
      SQSUM=SQSUM+SUM(J)*SUM(J)
   50 CONTINUE
      SQRTS=DSQRT(SQSUM)
C
C     DETERMINE E2
      MXLMDA=1.D0
      IF (NUM.GT.1) GO TO 60
      E2=.01D0*FOBJ/MXLMDA
      GO TO 70
   60 CALL MAX(XJMX,NUM,XMIN)
      E2=.5D0*XMIN
   70 IF(SQRTS.GT.E2) GO TO 1000
   71 ISTAGE=2
      GO TO 1200
C
C     IF STAGE=2 TEST CONDITIONS TO SWITCH TO STAGE 1
C
C     DETERMINE THE RESIDUAL
C
  100 NO=N+NUM
      K2=K2+1
      WRITE (IOUT,101) K2
  101 FORMAT(/1H0,'STAGE 2 ITERATION NO: ',I2)
      DO 105 I=1,NO
      IF (K2.LT.2) RESDUL(I)=0
      RPRE(I)=DABS(RESDUL(I))*.999D0
  105 CONTINUE
C
C
C     DETERMINE THE NEW ACTIVE SET
C
  125 CALL ACTIVE(FOBJ,F,M)
      RESDUL((N+2))=R(1)
      GO TO 10
  200 RESDUL(1)=SUMDIF
      DO 110 J=1,N
      JJ=J+1
      RESDUL(JJ)=SUM(J)
  110 CONTINUE
      IF (NUM.EQ.1) GO TO 121
      NUM1=NUM-1
      DO 120 I=1,NUM1
      RESDUL((N+1+I))=R(I)
  120 CONTINUE
  121 IF (K2.LT.3) GO TO 1200
      DO 210 I=1,NO
      RESDUL(I)=DABS(RESDUL(I))
      IF (NBUG.EQ.1) WRITE(IOUT,130)RPRE(I),RESDUL(I)
  130 FORMAT(' ','RPRE,RESDUL',2E15.5)
      IF(RPRE(I).LT.RESDUL(I)) GO TO 1000
  210 CONTINUE
      GO TO 1200
 1000 ISTAGE=1
      K2=0
 1200 RETURN
      END
```

```
C
C
      SUBROUTINE LINSCH (NOPROB,J0,FOBJ,FOBNXT,X,H,N)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(20),H(20),XL(20),FNXT(20)
C
C
      STEP=.5D0
      A0=0.D0
      FA=FOBJ
      FB=FOBNXT
      DO 5 I=1,N
      XL(I)=X(I)
    5 CONTINUE
      IF (FB.LE.FA) GO TO 50
      S=-STEP
      DO 10 I=1,N
      XL(I)=XL(I)+H(I)*S
   10 CONTINUE
      CALL FUNCTN(NOPROB,XL,N,FNXT,M,J0)
      FC=FNXT(J0)
      IF (FC.LE.FA) GO TO 40
C
C     BRACKET C A B
C
      A1=A0+S
      A2=A0
      A3=A0-S
      P1=FC
      P2=FA
      P3=FB
      GO TO 100
   40 FB=FC
      GO TO 51
   50 S=STEP
   51 A=A0
      B=A+S
   52 S=S*2
      IF (DABS(S).LE.1.D0) GO TO 60
      WRITE(IOUT,53)
   53 FORMAT(' ','STEPSIZE TOO LARGE')
      S=S/2
      GO TO 1000
   60 C=B+S
      DO 61 I=1,N
      XL(I)=XL(I)+H(I)*C
      IF (NBUG.EQ.1) WRITE(IOUT,555) XL(I),H(I),C
  555 FORMAT(' ','XL-H-C',3E15.5)
   61 CONTINUE
      CALL FUNCTN(NOPROB,XL,N,FNXT,M,J0)
      FC=FNXT(J0)
      IF (FC.GT.FB) GO TO 65
      A=B
      B=C
      FA=FB
      FB=FC
      GO TO 52
   65 D=.5*(B+C)
      DO 69 I=1,N
      XL(I)=XL(I)+H(I)*D
   69 CONTINUE
      CALL FUNCTN(NOPROB,XL,N,FNXT,M,J0)
      FD=FNXT(J0)
      IF (S.GE.0.D0) GO TO 80
```

```
C
C BRACKET C D B
C
      IF (FD.GE.FB) GO TO 75
      A1=C
      A2=D
      A3=B
      P1=FC
      P2=FD
      P3=FB
      GO TO 100
C
C     BRACKET D B A
C
   75 A1=D
      A2=B
      A3=A
      P1=FD
      P2=FB
      P3=FA
      GO TO 100
C
C     BRACKET BDC
C
   80 IF(FD.GE.FB) GO TO 85
      A1=B
      A2=D
      A3=C
      P1=FB
      P2=FD
      P3=FC
      GO TO 100
C
C
C
   85 A1=A
      A2=B
      A3=D
      P1=FA
      P2=FB
      P3=FD
C
C    QUAD INTERPOLATION
C
  100 H1=A2-A1
      H2=A3-A2
      DEN=H2*(P1-P2)+H1*(P3-P2)
      A4=A2+.5D0*(H2**2*(P1-P2)-H1**2*(P3-P2))/DEN
      IF (NBUG.EQ.1) WRITE(IOUT,99) A4
   99 FORMAT(' ','A4',E15.5)
      DO 110 I=1,N
      H(I)=A4*H(I)
  110 CONTINUE
 1000 RETURN
      END
```

```
C
C
C     THIS SUBROUTINE DETERMINES THE FUNCTION VALUES
C
C
      SUBROUTINE FUNCTN(NOPROB,X,N,F,M,J0)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(20),X(20)
C
C
      GO TO (10,20,30,40,50,60,70),NOPROB
   10 M=3
      IF(J0.EQ.0) GO TO 11
      GO TO (11,12,13),J0
   11 F(1)=X(1)*X(1)+X(2)*X(2)+X(1)*X(2)
      IF(J0.GT.0) GO TO 1000
   12 F(2)=DSIN(X(1))
      IF(J0.GT.0) GO TO 1000
   13 F(3)=DCOS(X(2))
      GO TO 1000
C
   20 M=2
      IF (J0.EQ.0) GO TO 21
      GOTO (21,22),J0
   21 IF(DABS(X(1)).LT.1.D-15) X(1)=0.D0
      F(1)=     (10.D0*(X(2)-X(1)*X(1)))
      IF(J0.GT.0) GOTO 1000
   22 F(2)=     (1.D0-X(1))
      GO TO 1000
C
   30 M=5
      IF(J0.EQ.0) GO TO 31
      GO TO (31,32,33,34,35),J0
   31 F(1)= (X(1)-10.)**2+5.*(X(2)-12)**2+X(3)**4+3*(X(4)-11)**2
     ¢+10*X(5)**6+7*X(6)**2+X(7)**4-4*X(6)*X(7)-10*X(6)-8*X(7)+1000
      IF (J0.GT.0) GO TO 110
   32 F(2)=(-2)*X(1)**2-3*X(2)**4-X(3)-4*X(4)**2-5*X(5)+127
      IF (J0.GT.0) GO TO 110
   33 F(3)=(-7)*X(1)-3*X(2)-10*X(3)**2-X(4)+X(5)+282
      IF (J0.GT.0) GO TO 110
   34 F(4)=(-23)*X(1)-X(2)**2-6*X(6)**2+8*X(7)+196
      IF(J0.GT.0) GO TO 110
   35 F(5)=(-4)*X(1)**2-X(2)**2+3*X(1)*X(2)-2*X(3)**2-5*X(6)+11*X(7)
  110 DO 112 I=2,5
      F(I)=(F(1)-10*F(I))
  112 CONTINUE
      F(1)=(F(1))
      GO TO 1000
C
   40 M=4
      IF(J0.EQ.0) GO TO 41
      GO TO (41,42,43,44),J0
   41 F(1)=X(1)*X(1)+X(2)*X(2)+2*X(3)*X(3)+X(4)*X(4)-5*X(1)-5*X(2)
     C-21*X(3)+7*X(4)+100
      IF(J0.GT.0) GO TO 140
   42 F(2)=-X(1)*X(1)-X(2)*X(2)-X(3)*X(3)-X(4)*X(4)-X(1)+X(2)-X(3)+8+

     CX(4)
      IF(J0.GT.0) GO TO 140
   43 F(3)=F(2)-X(2)*X(2)-X(4)*X(4)+2*X(1)-X(2)+X(3)+2
      IF(J0.GT.0) GO TO 140
   44 F(4)=F(2)+X(4)*X(4)-X(1)+X(3)-3
  140 DO 142 I=2,4
      F(I)=F(1)-10*F(I)
      F(I)=(F(I))
  142 CONTINUE
      F(1)=(F(1))
      GO TO 1000
```

```
C
   50 F(1)=100*(X(2)-X(1)**2)**2
      F(2)=(1-X(1))**2
      M=2
      N=2
      GO TO 1000
   60 M=3
      IF (J0.EQ.0) GO TO 61
      GO TO (61,62,63),J0
   61 F(1)=X(1)**4+X(2)**2
C2 61 F(1)=X(2)**4+X(1)**2
      IF (J0.GT.0) GO TO 1000
   62 F(2)=(2-X(1))**2+(2-X(2))**2
      IF(J0.GT.0) GO TO 1000
   63 F(3)=2*DEXP(-X(1)+X(2))
      GO TO 1000
C
   70 GO TO 1000
 1000 RETURN
      END
```

```
C
C
C     THIS SUBROUTINE DETERMINES THE DERIVATIVES
C
C
      SUBROUTINE DERIV(NOPROB,X,N,XJ,M)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(20),X(20),XJ(20,20)
C
C
      GO TO (10,20,30,40,50,60,70),NOPROB
C
   10 XJ(1,1)=2.D0*X(1)+X(2)
      XJ(1,2)=X(1)+2.D0*X(2)
      XJ(2,1)=DCOS(X(1))
      XJ(2,2)=0.D0
      XJ(3,1)=0.D0
      XJ(3,2)=-DSIN(X(2))
      M=3
      GO TO 1000
C
   20 XJ(1,1)=     (-20.D0*X(1))
      XJ(1,2)=10.D0
      XJ(2,1)=-1.D0
      XJ(2,2)=0.D0
      M=2
      GO TO 1000
C
   30 DO 15 I=1,5
      XJ(I,1)=2*X(1)-20
      XJ(I,2)=10*X(2)-120
      XJ(I,3)=4*X(3)**3
      XJ(I,4)=6*X(4)-66
      XJ(I,5)=60*X(5)**5
      XJ(I,6)=14*X(6)-4*X(7)-10
      XJ(I,7)=4*X(7)**3-4*X(6)-8
   15 CONTINUE
      XJ(2,1)=XJ(2,1)+40*X(1)
      XJ(2,2)=XJ(2,2)+120*X(2)**3
      XJ(2,3)=XJ(2,3)+10
      XJ(2,4)=XJ(2,4)+80*X(4)
      XJ(2,5)=XJ(2,5)+50
      XJ(3,1)=XJ(3,1)+70
      XJ(3,2)=XJ(3,2)+30
      XJ(3,3)=XJ(3,3)+200*X(3)
      XJ(3,4)=XJ(3,4)+10
      XJ(3,5)=XJ(3,5)-10
      XJ(4,1)=XJ(4,1)+230
      XJ(4,2)=XJ(4,2)+20*X(2)
      XJ(4,6)=XJ(4,6)+120*X(6)
      XJ(4,7)=XJ(4,7)-80
      XJ(5,1)=XJ(5,1)+80*X(1)-30*X(2)
      XJ(5,2)=XJ(5,2)+20*X(2)-30*X(1)
      XJ(5,3)=XJ(5,3)+40*X(3)
      XJ(5,6)=XJ(5,6)+50
      XJ(5,7)=XJ(5,7)-110
C     DO 100 I=1,M
C     DO 100 J=1,N
C     XJ(I,J)=DABS(XJ(I,J))
C 100 CONTINUE
C
      GO TO 1000
```

```
   40 N=4
      XJ(1,1)=2*X(1)-5
      XJ(1,2)=2*X(2)-5
      XJ(1,3)=4*X(3)-21
      XJ(1,4)=2*X(4)+7
      XJ(2,1)=XJ(1,1)+20*X(1)+10
      XJ(2,2)=XJ(1,2)+20*X(2)-10
      XJ(2,3)=XJ(1,3)+20*X(3)+10
      XJ(2,4)=XJ(1,4)+20*X(4)-10
      XJ(3,1)=XJ(2,1)-20
      XJ(3,2)=XJ(2,2)+20*X(2)+10
      XJ(3,3)=XJ(2,3)-10
      XJ(3,4)=XJ(2,4)+20*X(4)
      XJ(4,1)=XJ(2,1)+10
      XJ(4,2)=XJ(2,2)
      XJ(4,3)=XJ(2,3)-10
      XJ(4,4)=XJ(1,4)-10
      GO TO 1000
C
   50 XJ(1,1)=-400*X(1)*(X(2)-X(1)**2)
      XJ(1,2)=200*(X(2)-X(1)**2)
      XJ(2,1)=-2+2*X(1)
      XJ(2,2)=0
      GO TO 1000
C
   60 XJ(1,1)=4*X(1)**3
C2 60 XJ(1,2)=4*X(2)**3
C2    XJ(1,1)=2*X(1)
      XJ(1,2)=2*X(2)
      XJ(2,1)=-4+2*X(1)
      XJ(2,2)=-4+2*X(2)
      XJ(3,1)=-2*DEXP(-X(1)+X(2))
      XJ(3,2)=-XJ(3,1)
      GO TO 1000
C
   70 GO TO 1000
C
 1000 RETURN
      END
```

```
C
C
C
C
C     THIS SUBROUTINE DETERMINES THE HESSIAN
C
C
      SUBROUTINE HESIAN(NOPROB,X,N,G,M)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(20),G(40,20),GG(20)
      COMMON /STAGE/ ISTAGE,K1,K2,NBUG
C
      GO TO (10,20,30,40,50,60,70),NOPROB
   10 IF (K2.GT.1)GOTO 15
      G(1,1)=2.D0
      G(1,2)=1.D0
      G(2,1)=1.D0
      G(2,2)=2.D0
      DO 12 I=3,6
      DO 12 J=1,2
      G(I,J)=0.D0
   12 CONTINUE
      IF (NBUG.EQ.1) WRITE(6,13)(X(I),I=1,2)
   13 FORMAT(1H0,2HX=,2(E15.5))
   15 G(3,1)=-DSIN(X(1))
      G(6,2)=-DCOS(X(2))
C
      GO TO 1000
C
   20 IF(K2.GT.1) GO TO 1000
      MN=M*N
      DO 25 I=1,MN
      DO 25 J=1,N
      G(I,J)=0.D0
   25 CONTINUE
      G(1,1)=-20.D0
      GO TO 1000
C
   30 GG(1)=2.D0
      GG(2)=10.D0
      GG(3)=12*X(3)**2
      GG(4)=6.D0
      GG(5)=300*X(5)**4
      GG(6)=14.D0
      GG(7)=12*X(7)**2
      DO 32 I=1,M
      DO 32 K=1,N
      IK=(I-1)*N+K
      DO 32 J=1,N
      G(IK,J)=0.D0
      IF (K.EQ.J) G(IK,J)=GG(J)
      IF(J.EQ.6.AND.K.EQ.7) G(IK,J)=-4.D0
      IF(J.EQ.7.AND.K.EQ.6) G(IK,J)=-4.D0
   32 CONTINUE
      G(8,1)=42.D0
      G(9,2)=G(9,2)+360*X(2)**2

      G(11,4)=86.D0
      G(17,3)=G(17,3)+200.D0
      G(23,2)=30.D0
      G(27,6)=134.D0
      G(29,1)=82.D0
      G(29,2)=-30.D0
      G(30,1)=-30.D0
      G(30,2)=30.D0
      G(31,3)=G(31,3)+40.D0
      MTN=M*N
C     DO 999 I=1,MTN
C     DO 999 J=1,N
C     G(I,J)=DABS(G(I,J))
C 999 CONTINUE
```

```
      GO TO 1000
   40 N=4
      MN=M*N
      DO 42  I=1,MN
      DO 42  J=1,N
      G(I,J)=0
   42 CONTINUE
      G(1,1)=2
      G(2,2)=2
      G(3,3)=4
      G(4,4)=2
      G(5,1)=22
      G(6,2)=22
      G(7,3)=24
      G(8,4)=22
      G(9,1)=22
      G(10,2)=42
      G(11,3)=24
      G(12,4)=42
      G(13,1)=22
      G(14,2)=22
      G(15,3)=24
      G(16,4)=2
C
      GO TO 1000
C
   50 G(1,1)=-400*(X(2)-3*X(1)**2)
      G(1,2)=-400*X(1)
      G(2,1)=G(1,2)
      G(2,2)=200
      G(3,1)=2
      G(3,2)=0
      G(4,1)=0
      G(4,2)=0
      GO TO 1000
C
C2 60 G(2,2)=12*X(1)**2
```

```
C
C
C
C
C       THIS SUBROUTINE DETERMINES THE SUM OF ACTIVE HESSIAN
C
C
        SUBROUTINE UPDATE(G,M,N,GLM,HG)
C
C
        IMPLICIT REAL*8(A-H,O-Z)
        DIMENSION GLM(20),HG(20,20),G(40,20)
        COMMON  /ACTIV/ AS(3,20),NPTR,NXTPTR,NS(3)
C
        DO 10 I=1,N
        DO 10 J=1,N
        HG(I,J)=0.D0
   10   CONTINUE
C
        IK=0
        NAS=NS(NPTR)
        DO 30 I=1,M
CCC     NI=AS(NPTR,I)
        NI=I
        DO 20 K=1,N
        NI1=(NI-1)*N
        N1=NI1+K
        DO 20 J=1,N
        HG(K,J)=GLM(NI)*G(N1,J)+HG(K,J)
   20   CONTINUE
        IF(GLM(NI).EQ.0) IK=IK+1
   30   CONTINUE
C
        IF (IK.NE.NAS) GO TO 1000
C       DO 35 I=1,N
C       HG(I,I)=1.D0
C  35   CONTINUE
C
 1000   RETURN
        END
```

```
C
C
C
C
C
C     DETERMINES THE MAX VALUE
C
C
      SUBROUTINE MAX(FUNC,NO,FMAX)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION FUNC(20)
C
      FMAX=DABS(FUNC(1))
      DO 10 I=2,NO
      IF (FMAX.GE.DABS(FUNC(I))) GO TO 10
      FMAX=FUNC(I)
   10 CONTINUE
C
      RETURN
      END
```

```
C
C
C
C     DETERMINES THE ACTIVE FUNCTIONS
C
C
C
      SUBROUTINE ACTIVE(FOBJ,F,M)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(20)
C
      COMMON  /ACTIV/ AS(3,20),NPTR,NXTPTR,NS(3)
C
      NPTR=NXTPTR
      NXTPTR=MOD(NXTPTR,3)+1
      NS(NPTR)=0
      EDIFF=.01D0*FOBJ
      DO 10 I=1,M
      FDIFF=DABS(F(I)-FOBJ)
      IF (FDIFF.GT.EDIFF)GO TO 10
      NS(NPTR)=NS(NPTR)+1
      AS(NPTR,NS(NPTR))=I
   10 CONTINUE
C
      RETURN
      END
```

```
C
C
C     DETERMINES THE MATRICES FOR THE LINEAR OR QUADRATIC
C     LINEAR PROGRAM
C
C
      SUBROUTINE COEFF(F,M,XJ,N,XLAMDA,MN,GLM,X,NOPROB)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(20),XJ(20,20),XA(30,20),GLM(20),X(20)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40),HG(20,20),G(40,20)
      DIMENSION W(40),Z(40),MBSIS(80)
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
      COMMON /STAGE/ ISTAGE,K1,K2,NBUG
C
C     DETERMINE Q(*)
C
      MN=M+N*4+4
      MN1=1+N
      MN2=M+2*N+2
      MN3=2+N
      MN4=MN3+N
      N1=N+1
C
      DO 10 I=1,N
      Q(I)=0.D0
      I1=MN-I+1
      Q(I1)=XLAMDA
      I2=I+MN1
      Q(I2)=0.D0
      I3=MN2+I
      Q(I3)=XLAMDA
   10 CONTINUE
      Q(MN1)=1.D0
      Q(MN4)=-1.D0
      DO 20 I=1,M
      IN3=MN4+I
      Q(IN3)=-F(I)
   20 CONTINUE
C     Q((MN-N-1))=XLAMDA
      Q((MN-N-1))=0.D0
      Q(MN)=0.D0
      Q((MN-N))=XLAMDA
C
C     INITIALIZE XA(*,*)
C
      DO 25 I=1,MN2
      DO 25 J=1,MN4
      XA(I,J)=0.D0
   25 CONTINUE
C
C      DETERMINE XA(*,*)
C
      DO 40 J=1,N
      DO 30 I=1,M
      XA(I,J)=-XJ(I,J)
      JA1=MN1+J
      XA(I,JA1)=XJ(I,J)
   30 CONTINUE
      JA2=M+J
      XA(JA2,J)=1.D0
      JA3=JA2+N+1
      XA(JA3,J)=-1.D0
      XA(JA2,JA1)=-1.D0
      XA(JA3,JA1)=1.D0
   40 CONTINUE
      XA((JA2+1),MN1)=0
      XA((JA3+1),MN1)=0
      XA((JA2+1),MN4)=0
      XA((JA3+1),MN4)=0
      DO 50 I=1,M
      XA(I,MN1)=1.D0
```

```
      XA(I,MN4)=-1.D0
   50 CONTINUE
C
C     DETERMINE AM(*,*)
C
      DO 55 I=1,MN4
      DO 55 J=1,MN4
      AM(I,J)=0.D0
   55 CONTINUE
      GO TO(56,51),ISTAGE
C
C     DETERMINE HESSIAN IF CALL FROM STAGE 2
C
   51 K2=K2
      CALL HESIAN(NOPROB,X,N,G,M)
      CALL UPDATE(G,M,N,GLM,HG)
C
      DO 53 I=1,N
      I2=MN1+I
      DO 53 J=1,N
      AM(I,J)=HG(I,J)
      AM(I2,J)=-HG(I,J)
      J2=MN1+J
      AM(I2,J2)=HG(I,J)
      AM(I,J2)=-HG(I,J)
   53 CONTINUE
C
C     HESSIAN =0 IF CALL IS FROM STAGE 1
C
   56 MN5=MN4+1
      DO 60 I=MN5,MN
      DO 60 J=MN5,MN
      AM(I,J)=0.D0
   60 CONTINUE
      II=0
      DO 75 I=MN5,MN
      II=II+1
      DO 75 J=1,MN4
      AM(I,J)=XA(II,J)
      AM(J,I)=-XA(II,J)
   75 CONTINUE
      IF (NBUG.EQ.1) WRITE(6,61)
   61 FORMAT(//15X,8HVECTOR Q)
      IF (NBUG.EQ.1) WRITE(6,80)(Q(I),I=1,MN)
      DO 70 I=1,MN
      IF (NBUG.EQ.1) WRITE(6,80) (AM(I,J),J=1,MN)
   80 FORMAT(1H0,20(F6.2))
   70 CONTINUE
      RETURN
      END
```

```
C
      SUBROUTINE HVAL(H,N,GLM,FAPR,F,M,XJ)
C
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION H(20),GLM(20),F(20),XJ(20,20),FAPRX(20)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
C
C
      MN=1+N
C
      DO 10 I=1,N
      I1=MN+I
      H(I)=Z(I)-Z(I1)
      WRITE(6,11) H(I)
   11 FORMAT(' ',2HH=,10(E12.4))
   10 CONTINUE
      DO 20 I=1,M
      I2=2*MN+I
      GLM(I)=Z(I2)
   20 CONTINUE
C
C     DETERMINE APPROX. VALUE OF FUNCTION PREDICTED BY LP.
C
      DO 40 I=1,M
      DELTAF=0.D0
      DO 30 J=1,N
      DELTAF=XJ(I,J)*H(J)+DELTAF
   30 CONTINUE
      FAPRX(I)=F(I)+DELTAF
   40 CONTINUE
      CALL MAX(FAPRX,M,FMAX)
      FAPR=FMAX
      RETURN
      END
```

```
      SUBROUTINE LEMKE(N,IFLAG)
C                   ALGORITHM 431
C
C        A COMPUTER ROUTINE FOR QUADRATIC AND LINEAR PROGRAMMING PROBLEMS
C
C               COMMUNICATIONS OF THE ACM
C
C           VOL. 15 SEPT. 1972 PP. 818-820
C
C          AUTHOR - ARUNACHALAM RAVINDRAN
C
C          MODIFIED BY - PENSRI TEERAVARAPAUG
C
C          LANGUAGE - A.N.S.I
C                     STANDARD FORTRAN
C
C          INSTALLATION - OKLAHOMA STATE UNIVERSITY
C
C          DATE - DECEMBER 1974
C
C REMARKS
C SINCE THIS PROGRAM IS COMPLETE IN ALL RESPECTS,IT CAN BE
C RUN AS IT IS WITHOUT ANY ADDITIONAL MODIFICATION OR
C INSTRUCTION.IN SUCH CASE FOLLOW THE INPUT FORMAT AS GIVEN
C
C PROGRAM FOR SOLVING LINEAR AND QUADRATIC PROGRAMMING
C PROBLEMS IN THE FORM W=M*Z+Q, Q.Z=0, W AND Z NONNEGATIVE
C BY LEMKE/S ALGORITHM.
C
C MAIN PROGRAM WHICH CALLS THE SIX SUBROUTINES-MATRX,
C INITL,NEWBS,SORT,PIVOT AND PRINT IN PROPER ORDER.
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
C
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
C
C DESCRIPTION OF PARAMETERS IN COMMON
C   AM        A TWO DIMENSIONAL ARRAY CONTAINING THE
C             ELEMENTS OF MATRX M.
C   Q         A SINGLY SUBSCRIPTED ARRAY CONTAINING THE
C             ELEMENTS OF VECTOR Q.
C   L1        AN INTEGER VARIABLE INDICATING THE NUMBER OF
C             ITERATIONS TAKEN FOR EACH PROBLEM.
C   B         A TWO DIMENSIONAL ARRAY CONTAINING THE
C             ELEMENTS OF THE INVERSE OF THE CURRENT BASIS.
C   W         A SINGLY SUBSCRIPTED ARRAY CONTAINING THE VALUES
C             OF W VARIABLES IN EACH SOLUTION.
C   Z         A SINGLY SUBSCRIPTED ARRAY CONTAINING THE VALUES
C             OF Z VARIABLES IN EACH SOLUTION.
C   NL1       AN INTEGER VARIABLE TAKING VALUE 1 OR 2 DEPEND-
C             ING ON WHETHER VARIABLE W OR Z LEAVES THE BASIS
C   NE1       SIMILAR TO NL1 BUT INDICATES VARIABLE ENTERING
C   NL2       AN INTEGER VARIABLE INDICATING WHAT COMPONENT
C             OF W OR Z VARIABLE LEAVES THE BASIS.
C   NE2       SIMILAR TO NL2 BUT INDICATES VARIABLE ENTERING
C   A         A SINGLY SUBSCRIPTED ARRAY CONTAINING THE
C             ELEMENTS OF THE TRANSFORMED COLUMN THAT IS
C             ENTERING THE BASIS.
C   IR        AN INTEGER VARIABLE DENOTING THE PIVOT ROW AT
C             EACH ITERATION. ALSO USED TO INDICATE TERMINA-
C             TION OF A PROBLEM BY GIVING IT A VALUE OF 1000.
C   MBSIS   A SINGLY SUBSCRIPTED ARRAY-INDICATOR FOR THE
C             BASIC VARIABLES. TWO INDICATORS ARE USED FOR
C             EACH BASIC VARIABLE-ONE INDICATING WHETHER
C             IT IS A W OR Z AND ANOTHER INDICATING WHAT
C             COMPONENT OF W OR Z.
C
      IOUT=6
      IN=5
CC
CCREAD IN THE VALUE OF VARIABLE IP INDICATING THE
```

```
CCNUMBER OF PROBLEMS TO BE SOLVED.
CC
CC      READ(IN,1030) IP
CC
CCVARIABLE NO INDICATES THE CURRENT PROBLEM BEING SOLVED
CC
      IP=1
      NO=0
 1000 NO=NO+1
      IF(NO-IP)1010,1010,1070
 1010 WRITE(IOUT,1020)
 1020 FORMAT (/1H0,10X,11HLEMKE CALL )
CC
CC READ IN THE SIZE OF THE MATRIX M
CC
CC      READ(IN,1030)N
CC      WRITE(IOUT,1030)N
CC 1030 FORMAT (I2)
C
C PROGRAM CALLING SEQUENCE
C
      CALL MATRX (N)
C
C PARAMETER N INDICATES THE PROBLEM SIZE
C
      CALL INITL (N)
C
C SINCE FOR ANY PROBLEM TERMINATION CAN OCCUR IN INITIA,
C NEWBAS OR SORT SUBROUTINE,THE VALUE OF IR IS MATCHED WITH
C 1000 TO CHECK WHETHER TO CONTINUE OR GO TO NEXT PROBLEM.
C
      IF(IR-1000)1040,1000,1040
 1040 CALL NEWBS (N)
      IF(IR-1000)1050,1000,1050
 1050 CALL SORT (N,IFLAG)
      IF(IR-1000)1060,1000,1060
 1060 CALL PIVOT (N)
      GO TO 1040
 1070 RETURN
      END
```

```
      SUBROUTINE MATRX (N)
C
C PURPOSE  - TO INITIALLIZE AND READ IN THE VARIOUS INPUT DATA
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
C
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
C
      IOUT=6
      IN=5
      RZERO=0.0
      RONE=1.0
CC
CC READ THE ELEMENTS OF M MATRX COLUMN BY COLUMN
CC
CC      DO 2010 J=1,N
CC        READ(IN,2000)  (AM(I,J),I=1,N)
CC 2000   FORMAT (7F10.5)
CC 2010   WRITE(IOUT,2000)(AM(I,J),I=1,N)
CC
CC READ THE ELEMENTS OF Q VECTOR
CC
CC      READ(IN,2000)  (Q(I),I=1,N)
CC      WRITE(IOUT,2000)(Q(I),I=1,N)
C
C IN ITERATION 1,BASIS INVERSE IS AN IDENTITY MATRIX.
C
      DO 2030 J=1,N
        DO 2020 I=1,N
 2020     B(J,I)=RZERO
 2030   B(J,J)=RONE
      RETURN
      END
```

```
      SUBROUTINE INITL (N)
C
C PURPOSE TO FIND THE INITIAL ALMOST COMPLEMENTARY SOLUTION.
C          BY ADDING AN ARTIFICIAL VARIABLE Z0.
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
C
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
      IOUT=6
      RZERO=0.0
      TNONE=-1.0
C
C SET Z0 EQUAL TO THE MOST NEGATIVE Q(I)
C
      I=1
      J=2
 3000 IF(Q(I)-Q(J))3010,3010,3020
 3010   GO TO 3030
 3020 I=J
 3030 J=J+1
      IF(J-N)3000,3000,3040
C
C UPDATE Q VECTOR
C
 3040 IR=I
      T1=-Q(IR)
      IF(T1)3120,3120,3050
 3050 DO 3060 I=1,N
        Q(I)=Q(I)+T1
 3060   CONTINUE
      Q(IR)=T1
C
C UPDATE BASIS INVERSE AND INDICATOR VECTOR
C OF BASIC VARIABLES.
C
      DO 3070 J=1,N
        B(J,IR)=TNONE
        W(J)=Q(J)
        Z(J)=RZERO
        MBSIS(J)=1
        L=N+J
        MBSIS(L)=J
 3070   CONTINUE
      NL1=1
      L=N+IR
      NL2=IR
      MBSIS(IR)=3
      MBSIS(L)=0
      W(IR)=RZERO
      Z0=Q(IR)
      L1=1
C
C PRINT THE INITIAL ALMOST COMPLEMENTARY SOLUTION
C
C     WRITE(IOUT,3080)
C3080 FORMAT (3(/),5X,29HINITIAL ALMOST COMPLEMENTARY ,
C    *   8HSOLUTION)
C     DO 3100 I=1,N
C       WRITE(IOUT,3090)I,W(I)
C3090   FORMAT (10X,2HW(,I4,2H)=,F15.5)
C3100   CONTINUE
C     WRITE(IOUT,3110)Z0
C3110 FORMAT (10X,3HZ0=,F15.5)
      RETURN
 3120 WRITE(IOUT,3130)
 3130 FORMAT (///5X,36HPROBLEM HAS A TRIVIAL COMPLEMENTARY ,
     *    23HSOLUTION WITH W=Q, Z=0.)
      CALL PRINT(N)
      IR=1000
      RETURN
      END
```

```
      SUBROUTINE NEWBS (N)
C
C PURPOSE - TO FIND THE NEW BASIS COLUMN TO ENTER IN
C           TERMS OF THE CURRENT BASIS.
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
C
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
C
      IOUT=6
      RZERO=0.0
C
C IF NL1 IS NEITHER 1 NOR 2 THEN THE VARIABLE Z0 LEAVES THE
C BASIS INDICATING TERMINATION WITH A COMPLEMENTARY SOLUTION
C
      IF(NL1-1)4000,4030,4000
 4000 IF(NL1-2)4010,4060,4010
 4010 WRITE(IOUT,4020)
 4020 FORMAT (///5X,22HCOMPLEMENTARY SOLUTION)
      CALL PRINT(N)
      IR=1000
      RETURN
 4030 NE1=2
      NE2=NL2
C
C UPDATE NEW BASIC COLUMN BY MULTIPLYING BY BASIS INVERSE.
C
      DO 4050 I=1,N
        T1=RZERO
        DO 4040 J=1,N
       IF (DABS(B(I,J)).LT.1.0D-15) B(I,J)=0.D0
       IF(DABS(AM(J,NE2)).LT.1.D0-15) AM(J,NE2)=0.D0
 4040     T1=T1-B(I,J)*AM(J,NE2)
        A(I)=T1
 4050   CONTINUE
      RETURN
 4060 NE1=1
      NE2=NL2
      DO 4070 I=1,N
        A(I)=B(I,NE2)
 4070   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE SORT (N,IFLAG)
C
C PURPOSE - TO FIND THE PIVOT ROW FOR NEXT ITERATION BY THE
C             USE OF (SIMPLEX-TYPE) MINIMUM RATIO RULE.
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
C
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
C
      IOUT=6
      I=1
 5000 IF(A(I))5010,5010,5030
 5010 I=I+1
      IF(I-N)5020,5020,5100
 5020 GO TO 5000
 5030 T1=Q(I)/A(I)
      IR=I
 5040 I=I+1
      IF(I-N)5050,5050,5090
 5050 IF(A(I))5060,5060,5070
 5060 GO TO 5040
 5070 T2=Q(I)/A(I)
      IF(T2-T1)5080,5040,5040
 5080 IR=I
      T1=T2
      GO TO 5040
 5090 RETURN
C
C FAILURE OF THE RATIO RULE INDICATES TERMINATION WITH
C NO COMPLEMENTARY SOLUTION.
C
 5100 WRITE(IOUT,5110)
 5110 FORMAT (///5X,37HPROBLEM HAS NO COMPLEMENTARY SOLUTION)
      CALL PRINT(N)
      IFLAG=1
      IR=1000
      RETURN
      END
```

```
      SUBROUTINE PIVOT (N)
C
C PURPOSE - TO PERFORM THE PIVOT OPERATION BY UPDATING THE
C            INVERSE OF THE BASIS AND Q VECTOR.
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
C
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
C
      DO 6000 I=1,N
 6000   B(IR,I)=B(IR,I)/A(IR)
      Q(IR)=Q(IR)/A(IR)
      DO 6030 I=1,N
        IF(I-IR)6010,6030,6010
 6010   Q(I)=Q(I)-Q(IR)*A(I)
        DO 6020 J=1,N
          B(I,J)=B(I,J)-B(IR,J)*A(I)
 6020     CONTINUE
 6030   CONTINUE
C
C UPDATE THE INDICATOR VECTOR OF BASIC VARIABLES
C
      NL1=MBSIS(IR)
      L=N+IR
      NL2=MBSIS(L)
      MBSIS(IR)=NE1
      MBSIS(L)=NE2
      L1=L1+1
      RETURN
      END
```

```
      SUBROUTINE PRINT (N)
C
C PURPOSE - TO PRINT THE CURRENT SOLUTION TO COMPLEMENTARY
C              PROBLEM AND THE ITERATION NUMBER.
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(40,40),Q(40),B(40,40),A(40)
      DIMENSION W(40),Z(40),MBSIS(80)
C
      COMMON /LEM/AM,B,Q,A,W,Z,MBSIS,L1,NL1,NL2,NE1,NE2,IR
C
      IOUT=6
      RZERO=0.0
      WRITE(IOUT,7000)L1
 7000 FORMAT (10X,13HITERATION NO.,I4)
      I=N+1
      J=1
 7010 K1=MBSIS(I)
      K2=MBSIS(J)
      IF(Q(J))7020,7030,7030
 7020 Q(J)=RZERO
 7030 IF(K2-1)7040,7060,7040
C7040 WRITE(IOUT,7050)K1,Q(J)
C7050 FORMAT (10X,2HZ(,I4,2H)=,F15.5)
 7040 IF(K1.EQ.0) GO TO 7080
      Z(K1)=Q(J)
      GO TO 7080
C7060 WRITE(IOUT,7070)K1,Q(J)
C7070 FORMAT (10X,2HW(,I4,2H)=,F15.5)
 7060 IF(K1.EQ.0) GO TO 7080
      W(K1)=Q(J)
 7080 I=I+1
      J=J+1
      IF(J-N)7010,7010,7090
 7090 RETURN
      END
```

VITA

Rosemary Fernandes

Candidate for the Degree of

Master of Science

Thesis: A SURVEY OF NON-SMOOTH OPTIMISATION METHODS AND AN EVALUATION OF A METHOD FOR MINIMAX OPTIMISATION

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Kerela, India, August 6, 1955, the daughter of Mr and Mrs P. J. Thomas.

Education: Graduate from Convent of Good Shepherd July 1972; received Bachelor of Technology in Civil Engineering 1977 from Indian Institute of Technology, Madras, India; received Master of Engineering in Structure Engineering in 1979 from Asian Institute of Technology, Bangkok, Thailand; completed requirements for the Master of Science degree in Computer Science at Oklahoma State University Stillwater, Oklahoma in December 1985.

Professional Experience: Research Assistant, International Ferrocement Information Center, Asian Institute of Technology, September 1978, to April 1979; Assistant to Chief Structural Engineer, Binnie dan Rakaan, Malaysia, September 1980, to September 1981.