A COMPARISON OF THE COMPUTATIONAL PERFORMANCE OF THREE QUADRATIC PROGRAMMING ALGORITHMS

By

FOUAD MUSTAPHA KHALILI

Bachelor of Science
in Civil Engineering
Oklahoma State University
Stillwater, Oklahoma
1982

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1984

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
December, 1987

A COMPARISON OF THE COMPUTATIONAL

PERFORMANCE OF THREE QUADRATIC

PROGRAMMING ALGORITHMS

Thesis Approved:

J P Chandler
Thesis Advisor

G. E. Hedrick

Norman N. Durham
Dean of the Graduate College

PREFACE

The main objective of this study is to compare the computational performance of three quadratic programming algorithms. A quadratic programming problem is one in which the objective function to be minimized is quadratic and the constraint functions are linear. The three algorithms are Wolfe's reduced gradient method (implemented in the MINOS package), Lemke's complementary pivot method, and Fletcher's active set method. Fletcher's method was shown to be superior to the other two methods. In this paper, a random-problems generator is used. In addition, a translator program has been written which tranforms a given input data into MPS and SPECS files which are needed for the MINOS package. In a recent study, it was shown that Lemke's algorithm terminated with an infeasible solution in a convex quadratic programming problem. This claim was investigated to know the reason for such an abnormal behavior. This investigation is a secondary objective of the study.

I would like to express my gratitude to my major advisor Dr. John P. Chandler for his motivation, guidance, and insightful suggestions. I would like also to extend my thanks to the other two members of the committee, Dr. George E. Hedrick and Dr. K. M. George for their help and assistance.

My parents deserve my deepest gratitude for their love, patience, guidance, and all the values they taught me. Finally, I thank God who is the real source of guidance and help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A quadratic programming problem (QPP) is a one in which the objective function to be minimized contains quadratic and linear terms and the constraints are linear. Perhaps the most general way to pose this problem is:

$$\underset{x}{\text{minimize}} \quad f(x) = (1/2)x^T A x - b^T x \tag{1.a}$$

$$\text{subject to} \quad C^T x \geq d \tag{1.b}$$

$$u \geq x \geq 1 \tag{1.c}$$

Where x, b, u, and l are all n x 1, A is n x n, d is m-2n x 1, and C^T is m-2n x n. Sometimes, however, in this paper we will pose the problem in the following form:

$$\underset{x}{\text{minimize}} \quad f(x) = (1/2)x^T A x - b^T x \tag{2.a}$$

$$\text{subject to} \quad c^T x \geq d \tag{2.b}$$

$$x \geq 0$$

Going from form (1) to form (2) can be readily done; it is only a matter of convenience that form (2) is used, as will become obvious later.

In this study, we compare the computational performance of three well-known algorithms. Many comparisons were done earlier between different algorithms that solve the quadratic programming problem.

Braitsch (20) made a comparison between four different algorithms, namely, Dantzig's algorithm (33), Beale's algorithm (8), Wolfe's simplex method algorithm (116), and a modification of Wolfe's algorithm due to Braitsch. Moore and Whinston (70) compared between two categories of simplicial methods. The first category was based on the work of Dantzig, Van de panne and Whinston (110). The second category consisted of Wolfe's method. Van de panne and Whinston (111) compared Beale's and Dantzig's algorithms. Ravindran and Lee (87) compared Wolfe's method, Lemke's complementary pivot method (62), Zangwill's convex simplex method (121), the quadratic differential algorithm of Wilde and Beightler (114), and SUMT (37). The three algorithms that are compared here are chosen for different reasons. In the study done by Ravindran and Lee, it was shown that Lemke's method out-performed the other four algorithms in terms of number of iterations and execution time. In addition, Lemke's algorithm is designed specifically for quadratic programming. Fletcher's algorithm (40) is an efficient one and, as pointed by Fletcher (43), is preferable to other quadratic programming methods. The MINOS package is widely used and solves general nonlinear programming problems. However, Murtagh and Saunders (73) claim that MINOS should be competitive with other algorithms designed specifically for quadratic programming.

The three algorithms, although popular and widely used, have never been compared before. This paper attempts to contribute to the area of computational experience in quadratic programming since relatively little is known in this area compared to the theoretical activity.

There is a secondary objective in this paper which is to investigate a claim raised by Chiang (26) in which a case was given where Lemke's algorithm gave an infeasible solution.

CHAPTER II

BACKGROUND AND LITERATURE REVIEW

Quadratic Programming Applications

The quadratic programming problem was studied a long time ago since it represented the simplest case in going from the linear programming field to the nonlinear programming field. The quadratic programming problem received a great deal of attention because of its wide field of applications. Quadratic programming models have been used in areas such as structural optimization (118, 58), industries (21, 69), weapon selection and target analysis in the military (21), governmental, agriculture, and economic planning (54, 64, 98, 96), capital budgeting (61), portfolio selection (66), optimal design and utilization of electrical and communication networks (35), transition probabilities (100), aircraft design (31), population control (76), and management and decision sciences (68). Moreover, quadratic programming can be used to solve constrained regression problems (21), 0-1 integer programs (85), and two person nonzero sum games (65). As pointed out by Betts (14), some algorithms that are designed to optimize general nonlinear programming problem may pose a series of quadratic programming problems to approximate the behavior of the actual functions. In fact, the application of quadratic programming to approximate problems with nonlinear objective functions and linear constraints could give satisfactory results. Using quadratic functions to approximate a nonlinear function, especially near the minimum point where the behavior of the two functions is similar, is a well-known technique in solving unconstrained optimization problems. It is important to point out that the recursive quadratic programming methods are very promising approaches to solving the general nonlinear programming problems. These techniques have been studied by many researchers including Wilson (115), Biggs (15, 16), Fletcher (41, 42), Han (52, 53), Tapia (99), Powell (78, 79, 81, 82), Murray and Wright (72), Schittkowski (93, 94), Bartholomew-Biggs (6), Tone (103), Fukushima (45), and Powell and Yuan (83). For a brief review of these methods, the interested reader is refered to Bartholomew-Biggs (5). The general scheme of these methods could be summarized as follows. Given an estimate of the solution, a search direction could be obtained by solving a quadratic programming subproblem which is an approximation to the original problem. A new estimate is then obtained by moving along the calculated direction. The step-size of this movement is calculated by some technique. This process of moving from one estimate to another is repeated until the optimal point of the original problem is reached. In addition, optimization problems where quadratic terms appear in the constraints can be reformulated into a quadratic programming problem as Townsley (104) and Chen (25) have shown. Many problems, such as transportation, can be optimized with multiple objective programming which can be formulated using quadratic programming (68).

We now give an example to show how to use quadratic programming. Consider the problem of diminishing returns to scale, which is a well-known problem in economics. In this problem, the returns less the cost of production, which is an increasing function of quantity, is to be maximized. This problem can be posed as:

$$\text{Max.} \quad x^T p - x^T (c + \lambda x)$$

$$\text{Subject to} \quad A\, x \geq b$$

Here p denotes price, $c + \lambda x$ denotes the production cost to produce x units, and $A\, x \geq b$ represents restrictions on resources. For example, suppose that a certain company produces item z and it sells it for \$20.00. Suppose, also, that the company can not produce more than 200 of this item and that producing the first z costs \$1.00, and every additional z costs \$.00025. This problem could be mathematically written as:

$$\underset{x_z}{\text{Max.}} \quad 20\, x_z - (1 + .00025\, x_z)\, x_z$$

$$\text{S. T.} \quad x_z < 200$$

where x_z is the number of z items produced.

Quadratic Programming as a Linear Programming Extension

Early treatment of quadratic programming was based on linear programming techniques. Beale (7, 8, 9, 10) was the first to present an algorithm for solving quadratic programming problems. His approach was an extension of linear programming. Later, Wolfe (116) developed the simplex method for quadratic programming by solving the Kuhn-Tucker system as was suggested earlier by Barankin and Dorfman (4) and by Markowitz (67). In fact, earlier than this date, Frank and

Wolfe (44) proposed an algorithm to solve the quadratic programming problem using the Kuhn-Tucker system. In 1963, Dantzig (33) gave a variant of Wolfe's simplex algorithm. Van de panne (108) introduced, independently, the same algorithm, which he called the non-artificial simplex method. In the same year, Shetty (95) introduced his algorithm. A similar algorithm was given by Jagannathan (57). In 1964, Van de panne and Whinston (112) introduced their version of the simplex method. In the same year, Candler and Townsley gave another algorithm (24). The same authors (105) suggested a parametric linear programming approach in 1972. The work of solving the quadratic programming problem by solving the Kuhn-Tucker system was later called the linear complementarity problem (LCP). Lemke (62,63) developed a complementary pivot algorithm for solving the linear complementarity problem. In 1967, Graves (51) suggested a method he called the principal pivoting simplex algorithm. Cottle and Dantzig (28, 29) gave the principal pivot method. Tucker (106) used a least-distance approach to solve the quadratic programming problem. Eaves (36) extended Lemke's algorithm to calculate stationary points for general quadratic programming problems. Todd (102) gave an algorithm for generalized complementary pivoting. Ahn (1) gave some iterative methods to solve the linear complementarity problem. Goncalves (48) and Land and Morton (60) developed two different versions of Beale's method. Rusin (91) gave his revised simplex method for quadratic programming which reduces to the simplex method for linear programming when the objective function is linear. Goncalves (47, 49) developed the primal-dual method for quadratic programming. In 1980, Sacher (92) gave a decomposition algorithm which used Lemke's method. Another decomposition method was given by Whinston (113).

Other Approaches For Solving the Quadratic Programming Problem

There are several approaches other than those mentioned in the previous section for solving the quadratic programming problem. A combinatorial approach has been used by Theil and Van de panne (101), Boot (17, 18), Parsons (77), and Van de panne (110). In this approach, the idea is to solve a sequence of equality constrained problems. A similar but more systematic approach is the active set method. Fletcher (40, 43) uses this approach and a good discussion is given there. In 1960, Houthakker (56) introduced his capacity method where a restricted problem is obtained by adding a constraint of the form $\sum_{i=1}^{n} x_i \leq u$ and then solved. u is then increased and the problem is solved again. A one-direction search technique was developed by Hildreth (55) and D'Espo (32). In fact, all methods of feasible directions can be applied to solve the quadratic programming problem. A feasible directions algorithm is one which solves a nonlinear optimization problem by moving from one feasible point to another improved point along a certain direction of search d. In fact, Beale's method is an implementation of a convex simplex method of Zangwill (121). It could be considered as an active set method, as Fletcher (43) has shown. Some deformation methods were also used by authors such as Zahl (119, 120) and Bove (19). The idea of this method is to continuously deform an augmented objective function that is obtained by distorting the feasible region in such a way that an arbitrary initial optimum is obtained which is a solution to this deformed problem, until the problem is finally changed to the original one and a solution is obtained. Goldfarb (46) gave two methods which might be considered extensions of Newton's method for minimizing an unconstrained quadratic function.

All the methods that are discussed so far, except Fletcher's and Beale's algorithms, solve the convex quadratic programming problem, that is the case when the quadratic matrix is positive definite or positive semi-definite. When the quadratic matrix is indefinite, we have a general quadratic programming problem. Cutting plane methods were used to solve this problem in which the problem is posed as a minimization of a linear function subject to constraints in the form of a linear complementarity problem. Tui (107), Ritter (88, 89), Cottle and Mylander (30), Burdet (22), Balas (2), and Balas and Burdet (3) used this approach. There are several other approaches; these include Coffman, Majthay, and Whinston (27), Cabot and Francis (23), Mueller (71), Mylander (75), Taha (97), Van de panne (109), Goncalves (50), Keller (59), Zwart (122), Beneveniste (11, 12), Powell (80), and Betts (13, 14).

CHAPTER III

METHODOLOGY AND DESCRIPTION OF THE ALGORITHMS

Fletcher's Active Set Method

In this method, an equality problem (EP) is derived from the quadratic programming problem by keeping a basis of active constraints which are treated as equalities and disregarding the other constraints temporarily. Initially, the set of active constraints is chosen to provide a unique minimum. To meet this requirement, it is sufficient that A is strictly positive definite. On the other hand, if A is indefinite then it is sufficient to choose any n independent constraints. We start minimizing the quadratic function over this active constraint surface. Two possibilities exist here. It may be that a constraint is encountered which prevents the minimum of the current basis being reached. In this case, this constraint is added to the basis and the minimization process is continued. The second probability is that a minimum to the current equality problem has been found. In this case, the corresponding Lagrange multipliers are calculated, and if they are all negative the solution is optimal. Otherwise, the constraint with maximum Lagrange multiplier is dropped from the basis and minimization is continued with this new basis. The algorithm is now described with more details.

Suppose we need to find the minimum point of solution for the following problem:

$$\underset{x}{\text{Minimize}} \quad (1/2)\, x^T A\, x \; - b^T x \tag{3.a}$$

$$\text{S.T.} \qquad C^T x = d \tag{3.b}$$

where T superscript means transposition and C is a k x n matrix where $k \leq n$.

The Lagrangian function L of this problem is:

$$L(x, \lambda) = (1/2)\; x^T A\, x - b^T x + \lambda^T (C\, x^T - d) \tag{4}$$

where λ is the Lagrange multipliers vector.

Differentiating with respect to x and λ, respectively, and setting the result to zero gives the conditions for a stationary point:

$$\frac{\partial L}{\partial x} = Ax - b + \lambda^T C^T = 0 \tag{5.a}$$

$$\frac{\partial L}{\partial \lambda} = C^T x - d = 0 \tag{5.b}$$

In matrix form:

$$\begin{bmatrix} A & C \\ C^T & 0 \end{bmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} b \\ d \end{pmatrix} \tag{6}$$

To find the solution for this linear equations system, the inverse of the coefficient matrix is obtained:

$$\begin{bmatrix} A & C \\ C^T & 0 \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} - A^{-1} C\, (C^T A^{-1} C)^{-1} C^T A^{-1} & A^{-1} C (C^T A^{-1} C)^{-1} - (C^T A^{-1} C)^{-1} \\ (C^T A^{-1} C)^{-1} C^T A^{-1} & -(C^T A^{-1} C)^{-1} \end{bmatrix} \tag{7}$$

the solution vector, $(\hat{x}, \hat{\lambda})$, is:

$$\hat{x} = (A^{-1} - A^{-1} C (C^T A^{-1} C)^{-1} C^T A^{-1})\, b + A^{-1} C (C^T A^{-1} C)^{-1} d \tag{8.a}$$

$$\hat{\lambda} = (C^T A^{-1} C)^{-1} C^T A^{-1} b - (C^T A^{-1} C)^{-1} d \tag{8.b}$$

Substituting the gradient vector $g = Ax - b$ in (8.b) and $x = (C^T)^{-1}d$ in (8.a) gives:

$$\hat{x} = (x - (A^{-1} - A^{-1}C(C^T A^{-1}C)^{-1}C^T A^{-1})g \quad (9.a)$$

$$\hat{\lambda} = -(C^T A^{-1}C)^{-1} C^T A^{-1}\hat{g} \quad (9.b)$$

Where $\hat{g} = A\hat{x} - b$.

In these equations, two operators keep appearing and they are of great importance in the algorithm. The first operator is:

$$C^* = (C^T A^{-1} C)^{-1} C^T A^{-1} \quad (10)$$

C^* is a k x n matrix and it becomes C^{-1} when k is equal to n.

The second operator is:

$$H = A^{-1} - A^{-1}C(C^T A^{-1}C)^{-1}C^T A^{-1} \quad (11)$$

H is of rank n-k. If H is positive semi-definite, then a strict minimum point of the equality problem exists. It is to be noticed that C^* and H always exist because they are just partitions of (5) and the inverse of (5) must exist if the solution to the equality problem is unique.

To update these two operators, it takes only $O(n^2)$ computer operations, which makes the process of moving from one equality problem to another efficient. The recurrence relations for updating the operators are given below:

(1) To add a constraint, compute

$$C^*_{k+1} = \begin{pmatrix} C^*_k \\ 0 \end{pmatrix} + \begin{pmatrix} -C^*_k c \\ 1 \end{pmatrix} v^T/v^T c \quad (12.a)$$

$$H_{k+1} = H_k - vv^T/v^T c \quad (12.b)$$

where c is the normal of the added constraint and $v = H_k c$.

(2) To remove a constraint, compute

$$\begin{pmatrix} C_k^* \\ 0 \end{pmatrix} = C_{k+1}^* - C_{k+1}^* Ac^*c^{*T} / c^{*T} Ac^* \qquad (13.a)$$

$$H_k = H_{k+1} + c^* c^{*T} / c^{*T} Ac^* \qquad (13.b)$$

where c^{*T} is the K + 1th row of C^*_{k+1}, i.e. the row corresponding to the constraint to be removed.

However, because of the possibility of dividing by zero, these formulae cannot always be used safely. To avoid this problem, we need to come up with recurrence relations that perform the updating when one constraint is exchanged for another in C_k. These relations are given below:

$$C_k^* \longleftarrow C_k^* - (C_k^* c - e_k) w^T/y - C_k^* Ac^* u^T/y \qquad (14.a)$$

$$H_k \longleftarrow H_k + c^* u^T/y - H_k c w^T/y \qquad (14.b)$$

where $e_k{}^T$ is the vector $(0, 0, \ldots, 0, 1)$ in E^k, and

$$w = H_k c (c^{*T} Ac^*) + c^*(c^T c^*) \qquad (15.a)$$

$$u = c^* (c^T H_k c) - H_k c \, (c^T c^*) \qquad (15.b)$$

and

$$y = (c^T c^*)^2 + c^{*T} Ac^* c^T H_k c \qquad (15.c)$$

It is possible here again that y is zero and a division failure could happen. Before discussing how to avoid such a problem, it is interesting to know that when k = n the exchange formulae reduce to:

$$C^* = C^{-1} \longleftarrow C^{-1} - (C^{-1}c - e_n) \, c^{*T}/c^T c^* \qquad (16)$$

$$H = 0$$

Whenever a constraint is dropped, the new direction of search becomes c^*, where c^* is the row of C^* corresponding to the constraint being dropped, and the new minimum point along c^* is at a distance $\hat{\lambda}/c^{*T}Ac^*$ where $-m$ is $c^{*T}g$. However, a constraint might prevent this minimum being reached. To see if this is the case, we need to find:

$$l = \min_i \,(d_i - c_i^T \;\hat{x})/c_i^T \;\; c^* \tag{17}$$

Where c_i is the normal of the i^{th} inactive constraint. Notice that $c_i^T\, c^*$ must be negative if every element in $\hat{\lambda}/c^{*T}Ac^*$ is positive and less than or equal to 1, in which case, no inactive constraint is to be added to the basis and the minimum point can be reached along c^*.

When the curvature along c^* (that is $c^{*T}Ac^*$) is negative, or positive but small, the exchange formulae do not work. To get more insight into this problem, consider Figure 1.

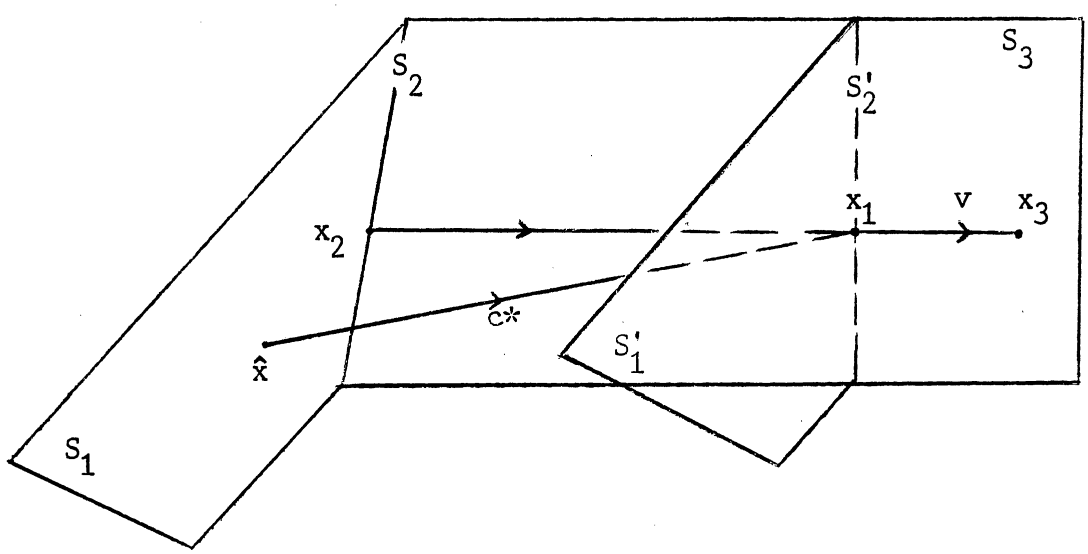

Figure 1. Changing the Basis of the Active Constraints.

In this figure, $\hat{x}$ is the current minimum point, c^* is the current direction of search, and S_1 is the current set of active constraints. Suppose that while searching along c^*, a new constraint with normal c is met at point x_1. It is important to recognize that x_1 is the minimum point of an equality problem with S_1' basis, where S_1' is parallel to S_1, and therefore, the operators for both bases are the same. The two bases are parallel in the sense that the constant term of the constraint of the normal c^* has been changed. Another important point that needs to be pointed out is that x_1 is also the minimum point of the equality problem of basis S_2' provided that the new constraint is independent of S_1. S_2' is S_1' plus the new constraint. Our concern, however, is to find the minimum of an equality problem of basis S_3 obtained by dropping the old constraint and adding the new constraint to basis S_1 or, equivalently, by removing the constraint that was obtained by changing the constant term of the old constraint from S_2'. To find this minimum, we proceed by adding the constraint corresponding to c to the current basis and then, in the next iteration, we assume that x_1, which is the minimum point of equality problem of S_2' basis, has been left by dropping the constraint corresponding to normal c^* and re-enter the previous code so that the operators for S_3 are not calculated. The direction of search in S_3 is:

$$v = c^* - Hc\ (c^Tc^*/c^THc) \tag{18}$$

and the curvature along this direction is:

$$v^TAv = c^{*T}Ac^* + (c^Tc^*)^2/c^T\,Hc = y/c^THc \tag{19}$$

After this description, the following conclusions can be derived.

If the new constraint is dependent or nearly dependent on the current basis, then the formulae for adding and dropping a constraint cannot be used; instead the exchange formulae must be used. If the constraint is dependent, then $c^T Hc = 0$ and using (19) y becomes $(c^T c*)^2$ which is strictly positive because $c^T c*$ is negative always. Using (19) again, it is clear that if $y \leq 0$, then $c^T Hc \leq 0$ because $(c^T c*)^2$ is positive and $c*^T Ac*$ is negative, and hence (13) can be used safely. If both, exchanging and adding, are safe then ly and $c^T Hcv^T g_1$ are calculated, where $g_1 = Ax_1 - b$. If ly is smaller than $c^T Hcv^T g_1$, then the adding formulae are used; otherwise, the exchange formulae are used. The reader is referred to Fletcher's paper for more discussion.

Lemke's Complementary Pivoting Method

A linear complementary problem is to find two vectors w and z such that:

$$w = Mz + q \tag{20.a}$$

$$w^T z = 0 \tag{20.b}$$

$$w \geq 0,\ z \geq 0 \tag{20.c}$$

The Kuhn-Tucker conditions of the quadratic programming could be written as:

$$Cx - y = d \tag{21.a}$$

$$-Ax + Cu + v = -b \tag{21.b}$$

$$x^t v = 0,\ u^t y = 0 \tag{21.c}$$

$$x,\ y,\ u,\ v \geq 0 \tag{21.d}$$

where u and v are the Langrangian multiplier vectors of the $C^T x \geq d$ and

$x > 0$ constraints, respectively. These conditions can be reduced to a complementary problem by letting

$$w = \begin{pmatrix} v \\ y \end{pmatrix} \quad , M = \begin{bmatrix} A & -C \\ C^T & 0 \end{bmatrix} \tag{22}$$

$$q = \begin{pmatrix} -b \\ -d \end{pmatrix} \text{ and } z = \begin{pmatrix} x \\ u \end{pmatrix}$$

where q is L x 1 and M is L x L.

Hence, Lemke's algorithm can be used to solve the quadratic programming problem. Before describing the algorithm, some definitions are introduced. A solution (w, z) to (20) is called a complementary basic feasible solution if (w, z) is a basic feasible solution to (20.a) and (20.c) and if one variable of the pair (w, z) is basic for i = 1,...L. System (20) can be solved readily if $q \geq 0$ by letting $w = q$ and $z = 0$ On the other hand, if $q \leq 0$, a new column 1 (i.e., a vector of ones) and an artificial variable z_o are introduced into the system to get:

$$w - Mz - 1z_o = q \tag{23.a}$$

$$w^T z = 0 \tag{23.b}$$

$$w \geq 0, \; z \geq 0 \tag{23.c}$$

Initially, the artificial variable $z_o = \max(-q_i : 1 < i < L)$, $z = 0$, $w = q + 1z_o$ constitutes the solution. Lemke's complementary pivoting algorithm tries to drive z_o out of the basis through a sequence of pivots that satisfies (23). We now introduce another important definition. An almost-complementary basic feasible solution is a feasible solution (w, z, z_o) to (23) that satisfies the following requirements.

(1) (w, z, z_o) is a basic feasible solution to (23.a) and (23.c).

(2) For some $i \in (1,\ldots,L)$ both w and z are nonbasic.

(3) z_o is basic.

(4) For $j = 1,\ldots L$ and $j \neq i$, either w_j or z_j is basic.

An adjacent almost complementary basic feasible solution $(w_d,\ A_d,\ z_o)$ is introduced by allowing either w_i or z_i to enter the basis and driving a basic variable other than z_o, that is, either z_j or w_j, from the basis. Therefore, every almost complementary basic feasible solution can have a maximum of two adjacent almost complementary basic feasible solutions.

Lemke's algorithm moves among adjacent almost complementary basic feasible solutions until one of two things happen:

(1) A complementary basic feasible solution is reached.

(2) Stop with a ray termination because the feasible region is unbounded.

A summary of the algorithm can now be given:

1) If $q \geq 0$, a solution is readily available. The solution is $w = q$ and $z = 0$. Stop.

2) If $q < 0$ form a tableau for system 4.a and 4.c. Let $q_i = \min (q_j : 1 \leq j \leq L)$, and pivot at row i and column z_o. In this tableau the basic variables z_o and w_j, where $j = 1,\ldots,L$ and $j \neq i$, are all non-negative. Let $y_i = z_j$.

3) Let u_i denote the column that has been just updated (i.e., column under y_i). If $u_i \leq 0$, go to Step 7.

4) Let q be the updated right-hand-side column. q has the values of the basic variables. Obtain the index r by the following ratio

test:

$$\frac{q_r}{u_{ri}} = \underset{1 \leq j \leq L}{\text{minimum}} \left\{ \frac{q_j}{u_{ji}} : u_{ji} > 0 \right\}$$

If the basic variable at row r is z_o, go to Step 6.

5) Pivot at row r and the y_i column so that y_i will enter the basis. The variable that has just left the basis is either w_l or z_l where $l \neq i$. If it is w_l then $y_i \longleftarrow z_l$, otherwise $y_i \longleftarrow w_l$. Go to Step 3.

6) Pivot at row z_o and the y_i column so that z_o will leave the basis, and a complementary solution is reached. Stop.

7) In this case, a ray termination takes place, where $R = [(w, z, z_o) + \delta u : \delta \geq 0)]$ is found such that every point in R is a solution to the problem. Here (w, z, z_o) is the current almost complementary basic feasible solution and u is a vector that has a 1 at the row corresponding to y_i, $-u_i$ at the rows of the current basis variables, and zero elsewhere. Stop.

If there is no degeneracy involved in the problem, the algorithm is guaranteed to find a Kuhn-Tucker point in a finite number of steps if any one of the following conditions is true:

1. A is positive semidefinite and b = 0.
2. A is positive definite.
3. All diagonal elements of A are strictly positive and all others are nonnegative.

The MINOS Package

The MINOS package solves a linearly constrained nonlinear program using Wolfe's reduced gradient method (117) in conjunction with Davidon's quasi-Newton algorithm (34). In this section, we give a summary of the procedure as described in Murtagh and Saunders (73).

Initialization Step:

(a) A feasible point x which satisfies $[B\ S\ N]x = d$ and $l \leq x \leq u$ is obtained. Here B, S, and N are the arrays corresponding to basic (x_B), superbasic (x_S) and nonbasic (X_N) variables, respectively.

(b) The corresponding $(1/2)\ x^T Ax$ value and gradient vector $g(x) = (g_B\, g_S\, g_N)$ are calculated.

(c) The number of superbasic variables, s, is obtained. Here $0 \leq s \leq 3n - m$, and $m \leq 3n$.

(d) Calculate the LU factorization of the $m - 2n \times m - 2n$ basis matrix B.

(e) Calculate the R^TR factorization of a quasi-Newton approximation to the $s \times s$ matrix $Z^T AZ$, Z is a matrix that is orthogonal to the matrix of constraint normals, i.e. $C^TZ = 0$.

(f) Calculate the vector v such that $B^Tv = g_B$.

(g) Calculate the reduced-gradient vector h, $h = g_S - S^Tv$.

Step 1. (Test for convergence.)

If $\|h\| >$ TOLRG go to step 3.

(Where TOLRG is a small positive convergence tolerance.)

Step 2. (Estimate Lagrange multipliers, add one superbasic.)

a. Calculate $\lambda = g_N - N^Tv$

b. Select $\lambda_{q_1} < -$ TOLDJ ($\lambda_{q_2} >$ TOLDJ), The largest elements of λ corresponding to variables at their lower (upper) bound.

(TOLDJ is a small positive convergence tolerance.)

If none, stop; an optimal point has been obtained.

c. Choose $q = q_1$ or $q = q_2$ corresponding to

$|\lambda_q| = \max(|\lambda_{q_1}|, |\lambda_{q_2}|)$

d. Add c_q as a new column of S.

e. Add λ_q as a new element of h.

f. Add a suitable new column to R.

g. Increase s by 1.

Step 3. (Compute the new direction of search $p = Zp_S$.)

a. Solve $R^T R p_S = -h$ for p_S.

b. Solve LU $p_B = -S\ p_S$ for p_B.

c. Set $p = [\ p_B\ p_S\ 0]^T$

Step 4. (Find l_{max})

a. Find $l_{max} \geq 0$, the greatest value of l for which $x + lp$ is feasible.

b. If $l_{max} = 0$, go to Step 7.

Step 5. (Do a line search.)

a. Find l, an approximation to l^*,

where $f(x + l^* p) = \text{MIN}\ f(x + \theta p), \quad 0 \leq \theta < l_{max}$

Where $f(x)$ is $(1/2)\ x^T Ax$.

b. Change x to $x + lp$ and set f and g to their values at the new x.

Step 6. (Compute the reduced gradient $\bar{h}$, $\bar{h} = Z^T g$.)

a. Solve $U^T L^T v = g_B$

b. Compute the new reduced gradient $\bar{h}$, $\bar{h} = g_S - S^T v$

c. Modify R to reflect some variable-metric recursion on $R^T R$, using l, p_S, and the change in reduced gradient, $\bar{h} - h$

d. set $h = \bar{h}$.

e. If $l < l_{max}$, go to Step 1 (no new constraint was encountered.)

Step 7. (Change the current basis if necessary; delete one superbasic.)

a. If a basic variable hit its bound ($0 \leq p \leq m - 2n$)

(i) Interchange the pth and the qth columns of

$$[B \quad x_B^T]^T \quad \text{and} \quad [S \quad x_S^T]^T$$

Respectively, where q is chosen to keep B nonsingular.

(ii) Modify L, U, R, and v to reflect this change in B.

(iii) Compute the new reduced gradient h,

$$h = g_S - S^T v$$

(iv) Go to c.

b. Otherwise, a superbasic variable hits its bound ($m - 2n < p \leq m - 2n + s$). Define $q = p - m + 2n$.

c. Make the qth variable in S nonbasic at the appropriate bound, thus:

(i) Delete the qth columns of

$$[S \quad x_S^T]^T \text{ and } [R \quad h^T]^T.$$

(ii) Restore R to traingular form.

d. Decrease s by 1 and go to Step 1.

Random Quadratic Programming Problem Generator

A computer program was written to generate quadratic programming problems randomly following the method of Rosen and Suzuki (90) which is also described by Ravindran and Lee (87). Some minor modifications, however, were made. For example, to ensure a positive definite matrix A, A was calculated by using $A \leftarrow A^T A$. In this method, we solve (2) for b and d after generating C, A, x, u and v randomly. The description of the generator is as follows:

Step 1. Randomly generate $x \geq 0$ and $u \geq 0$.

Step 2. Randomly generate A and C with specified percentages of zero elements.

Step 3. Compute b as follows:

a. If $x_i = 0 \Rightarrow b_i \geq C_j\ u - Ax$

b. If $x_i \gneq 0 \Rightarrow b_i = C_j\ u - Ax$

C_j is the ith row of C and A_i is the ith row of A.

Step 4. Compute d_i as follows:

a. If $u_i = 0 \Rightarrow d = C_i^T x$

b. If $u_i \geq 0 \Rightarrow d = C_i^T x$

C_i^T is the ith row of C^T.

CHAPTER IV

RESULTS AND DISCUSSION

General

In this paper, Ravindran's (86) computer program for Lemke's method, modified by Proll (84), is used. Fletcher's (38, 39) routine for his method is used in this paper. However, to invert a matrix, subroutine LINV2F from the IMSL library is used. In addition, to find the inner product of two vectors, subroutine INNERP, developed by the author, is used. The most recent version of MINOS (74), implemented in 1983, is used in this study. The modified Ravindran's routine, Fletcher's program, a sample of the input for the MINOS package and a program to generate this sample automatically are given in Appendices A, B and C, respectively. All of the programs were run on the 3081 IBM mainframe at Oklahoma State University using double precision computations. This study involves comparing the computational performances of the three methods for convex and general quadratic programming cases.

Test Problems Design

The effect of different factors were studied in this study, these factors are the following:

1) The number of active constraints at the optimal point.

2) The number of constraints.

3) The number of variables.

4) The percentage of zero elements in the quadratic array A.

The above mentioned factors are considered for the case of convex quadratic programming only. In the case of an indefinite matrix A, the main purpose was to investigate the reliability of the three algorithms, i.e. their abilities to solve a given problem correctly.

Test Criteria

Many test criteria could be used to evaluate the performance of any algorithm. In this study, the criteria used are:

1) Robustness

2) Number of iterations

3) CPU time

The first criterion is the most important one since a user wants to use an algorithm which will surely give the correct answers to the given degree of precision. In fact, it is generally accepted that the primary criterion in evaluating an algorithm is its reliability.

The number of iterations is the second important criterion. However, sometimes this criterion might be misleading because one can reduce the number of iterations by different time-consuming ways such as special heuristic calculations. To avoid such unfair comparisons a third criterion should be employed, namely, the CPU time. It should be mentioned here that depending solely on the CPU time in measuring the performance of an algorithm might be misleading, also. Considerations such as care in coding the algorithm could significantly affect the results. In addition, if the operating system is multiprogrammed the CPU time becomes longer and less reliable. Consequently, the number of iterations should be used together with the CPU time to get a better insight into the performances of the different algorithms.

Results and Analysis

In the first part of the study, we consider the convex quadratic programming problem. It should be mentioned here that convex quadratic programming problems have only one local minimum, which is therefore the global minimum. For Ravindran's routine and the MINOS package no special parameters are required to be input. For Fletcher's program, three different modes could be used. Mode 1 is used for any quadratic programming problem. Modes 2 and 3 can be used when A is strictly positive definite. In addition, if mode 3 is used then the user should provide a feasible point to the routine. In fact, there are two additional modes that can be used, namely modes 4 and 5, and these are used for general parametric programming and right-hand side parametric programming, respectively.

Table I shows the effect of changing the number of active constraints at the optimal point on the number of iterations and the execution time. A total of 690 problems were tested, i.e. 10 problems for each case. The average number of iterations of these 10 runs (rounded to the nearest integer) and the average of the execution time are shown in Table I. In Fletcher's algorithm, an application of formulae (12), (13), or (16) is counted as 1, whereas application of (14) is counted as 2. In all of the tested cases, neither of the programs failed to reach the optimal solution. They all gave the "exact" answers. Table I shows clearly that for Lemke's algorithm the number of iterations increases as the number of active constraints

TABLE I

MEAN ITERATION COUNT AND EXECUTION TIME FOR THE THREE ALGORITHMS FOR THE CONVEX PROGRAMMING CASE WITH NUMBER OF VARIABLES EQUAL TO NUMBER OF CONSTRAINTS AND DIFFERENT NUMBER OF ACTIVE CONSTRAINTS

No. of Con-straints	No. of Variables	No. of Active Constraints	Fletcher		Lemke		MINOS	
			Iter	Time	Iter	Time	Iter	Time
2	2	2	2	.08	5	.05	5	.21
4	4	2	4	.09	7	.06	10	.24
4	4	4	4	.09	11	.06	4	.21
8	8	2	6	.12	11	.06	24	.35
8	8	4	6	.13	15	.1	23	.34
8	8	6	7	.13	17	.11	15	.29
8	8	8	8	.14	19	.11	11	.28
10	10	1	3	.14	12	.12	30	.42
10	10	2	4	.14	13	.12	28	.4
10	10	3	7	.16	14	.13	28	.4
10	10	4	8	.16	15	.13	26	.38
10	10	5	8	.16	16	.14	25	.38
10	10	6	8	.16	17	.15	22	.35
10	10	8	9	.17	19	.15	21	.35
10	10	10	10	.18	21	.15	14	.31
15	15	1	7	.28	19	.25	56	.75
15	15	2	8	.28	21	.25	55	.74
15	15	5	15	.34	23	.28	46	.64
15	15	8	15	.34	28	.30	40	.59
15	15	10	12	.31	29	.32	40	.58
15	15	12	14	.33	32	.34	36	.55
15	15	15	15	.35	33	.34	23	.45
20	20	2	21	.62	27	.49	77	1.37

increases. The same pattern is followed by Fletcher's algorithm except for two cases, namely, for the cases where the number of active constraints are 5 and 8 and the size of the problem is 15 x 15. A reverse pattern is obtained for the MINOS package. In all cases, the number of iterations for Fletcher's algorithm is less than that obtained by Lemke's algorithm which, in turn, is always less than that of the MINOS package. The execution time for the MINOS package is always bigger than that of the other two algorithms. In fact, the number of iterations and the execution time are always worse than those of the other two algorithms in all the test problems that were conducted in this study as can be seen in the tables.

The execution times for Fletcher and Lemke are very close. In approximately 90 percent of the test cases in Table I Lemke gave a better execution time than Fletcher.

To see the effect of using mode 3 on the performance of Fletcher's algorithm part of the test problems of Table I were used. The results are given in Table II. 75 problems were tested, i.e. 5 problems for each case. The results show that when the number of active constraints is small, better number of iterations and execution time can be obtained than when mode 2 is used.

The effect of the number of zero quadratic terms in the objective function is shown in Table III. In this table, as well as Tables IV and V, the number of the active constraints was set equal to 2. In Table III, a total of 150 problems were tested. Table III shows clearly that a significant decrease is obtained in the number of iterations and the execution time for Fletcher's algorithm. Lemke's algorithm and the MINOS pacakge are generally not affected.

TABLE II

MEAN ITERATION COUNT AND EXECUTION TIME FOR FLETCHER'S ALGORITHM WHEN USING MODE 3

No. of Constraints	No. of Variables	No. of Active Constraints	Iter	Time
2	2	2	2	.08
4	4	2	6	.09
4	4	4	4	.09
8	8	2	3	.12
8	8	4	10	.14
8	8	6	10	.14
8	8	8	7	.13
10	10	1	1	.12
10	10	2	2	.13
10	10	3	4	.13
10	10	4	8	.15
10	10	5	8	.15
10	10	6	10	.15
10	10	8	10	.16
10	10	10	10	.17

TABLE III

MEAN ITERATION COUNT AND EXECUTION TIME FOR THE THREE ALGORITHMS FOR DIFFERENT PERCENTAGES OF ZERO ELEMENTS IN THE QUADRATIC MATRIX A

No. of Con-straints	No. of Vari-ables	Percentage of Zero Elements in A	Fletcher		Lemke		MINOS	
			Iter	Time	Iter	Time	Iter	Time
2	2	50	2	.08	5	.06	3	.21
4	4	50	2	.08	7	.06	9	.25
8	8	12.5	6	.13	14	.1	24	.36
8	8	50	4	.13	11	.1	22	.35
10	10	32	4	.14	15	.13	30	.44
10	10	50	4	.14	13	.13	30	.44
15	15	22	7	.25	25	.29	48	.66
15	15	30	5	.25	26	.29	52	.74
15	15	40	4	.25	19	.26	51	.72
20	20	50	4	.41	23	.46	71	1.16

TABLE IV

MEAN ITERATION COUNT AND EXECUTION TIME FOR THE THREE ALGORITHMS WITH n = 4 AND INCREASING NUMBER OF CONSTRAINTS

No. of Constraints	No. of Variables	Fletcher Iter	Fletcher Time	Lemke Iter	Lemke Time	MINOS Iter	MINOS Time
6	4	2	.09	9	.07	11	.26
8	4	4	.11	13	.08	19	.31
10	4	3	.12	13	.1	26	.35
15	4	4	.18	18	.16	36	.48
20	4	6	.27	25	.26	47	.68

TABLE V

MEAN ITERATION COUNT AND EXECUTION TIME FOR THE THREE ALGORITHMS WITH 4 CONSTRAINTS AND INCREASING NUMBER OF VARIABLES

No. of Constraints	No. of Variables	Fletcher Iter	Fletcher Time	Lemke Iter	Lemke Time	MINOS Iter	MINOS Time
4	6	3	.09	7	.06	9	.24
4	8	3	.09	7	.07	9	.24
4	10	3	.1	9	.08	10	.26
4	15	4	.11	9	.09	12	.28
4	20	4	.11	9	.1	12	.29

In Table IV, the effect of increasing the number of constraints is shown. A total of 75 problems were tested. As expected, the number of iterations and the execution time increase as the number of constraints increases. In all of the cases, the number of iterations for Fletcher is significantly less than that for Lemke and MINOS.

The effect of increasing the number of variables is shown in Table V. Again 75 problems were tested. The table shows that the number of variables does not have a very significant effect on the results. Fletcher's algorithm is still superior to the other two algorithms in terms of the number of iterations.

In part two of the study, the general quadratic programming case was tested. The results are given in Table VI. A total of 54 cases were tested. Fletcher's algorithm and the MINOS package always gave the correct answers. Lemke's algorithm failed to arrive at an optimal point in 70 percent of the tested cases. This is not an abnormal behavior of the method because it is not guaranteed to give an optimal solution in the general quadratic programming case. It is because of this reason that the claim raised by Chiang (26) is not true. In the problem he was trying to solve the matrix of quadratic terms was positive semi-definite and for such a case it is guaranteed to obtain a solution by Lemke's algorithm only if the linear terms in the objective function are all zeros.

TABLE VI

MEAN ITERATION COUNT AND EXECUTION TIME FOR THE THREE ALGORITHMS FOR GENERAL QUADRATIC PROGRAMMING CASE

No. of Constraints	No. of Variables	No. of Active Constraints	Fletcher		Lemke		MINOS	
			Iter	Time	Iter	Time	Iter	Time
2	2	2	2	.08	4	.05	2	.21
4	4	2	2	.08	--	*	4	.23
8	4	2	3	.1	--	*	5	.25
10	5	2	7	.12	15	.09	6	.27
15	6	2	12	.14	--	*	7	.27
15	10	5	12	.21	--	*	16	.37

*Indicates failure to arrive at a solution.

CHAPTER V

SUMMARY AND CONCLUSIONS

The results given in Tables I through VI all indicate that Fletcher's algorithm is a very efficient algorithm to solve the quadratic programming problem. In the cases tested, Fletcher's algorithm never needed more than 2*n iterations to reach an optimal point. Although Lemke's algorithm gave slightly better execution time, one should not forget that this method has a drawback in that it enlarges the size of the problem since it tries to solve the Kuhn-Tucker conditions. In addition, Lemke's method does not solve general quadratic programming problems. In fact, it does not solve positive semi-definite problems. Hence, it should have troubles on ill-conditioned positive definite (but almost semi-definite) problems. On the other hand, Fletcher's algorithm requires a lower and an upper bound on each variable to be input. This can be a disadvantage, but if bounds are known then not much extra work is needed by Fletcher's algorithm while the other algorithm will need more iterations and execution time. Another advantage of Fletcher's algorithm is its flexibility, since 5 modes are available for the user. Furthermore, mode 3 should be used whenever matrix A is known to be strictly positive definite and it is expected that few constraints are active at the optimal point, since few iterations will then be needed to arrive at the solution. Finally, it is to be mentioned here that the MINOS package is slower than the other programs and does some times face problems when the problem is poorly scaled, as it did in 2 cases in part 2 of the study (i.e. in General Quadratic Programming Problems).

Therefore, Fletcher's method is recommended as the best method, among the three methods tested in this thesis, for solving quadratic programming problems.

SELECTED REFERENCES

1. Ahn, B. H. "Iterative methods for linear complementarity problems with upper bounds on primary variables," Math. Prog., 26, (1983), pp. 295-315.

2. Balas, E. "Nonconvex quadratic programming via generalized polars," Manag. Sci. Research Report No. 278(R), G.S.I.A., Carnegie-Mellon Univ., 1973.

3. Balas, E. and C. A. Burdet. "Maximizing a convex quadratic function subject to linear constraints," Manag. Sci. Res. Report No. 299, G.S.I.A., Carnegie-Mellon University, Sept. 1972-July 1973.

4. Barankin, E., and R. Dorfman. "On quadratic programming," University of California publications and statistics, 2, (1958), pp. 285-318.

5. Bartholomew-Biggs, M. C. "Recursive quadratic programming methods for nonlinear constraints," In Nonlinear Optimization, M. J. D. Powell (Ed), 1981, Academic Press, N.Y., 1982, pp. 213-221.

6. Bartholomew-Biggs, M. C. "A recursive quadratic programming algorithm based on the augmented lagrangian function," Technical report No. 139, Numerical Optimization Centre, The Hatfield Polytechnic, 1983.

7. Beale, E. M. L. "On minimizing a convex function subject to linear inequalities," Journal of the Royal Statistical Society (B), 17, (1955), pp. 173-184.

8. Beale, E. M. L. "On quadratic programming," Naval Research Logistics Quarterly, 6, (1959), pp. 227-243.

9. Beale, E. M. L. "Note on a comparison of two methods of quadratic programming," Operations Research 14, (1966), pp. 442-443.

10. Beale, E. M. L. "Numerical methods," In Nonlinear Programming, Abadie (Ed), Wiley, N.Y. 1967.

11. Benveniste, R. "A quadratic programming algorithms using conjugate serch directions," Math. Prog. 16(1979), pp. 63-80.

12. Benveniste, R. "Quadratic programming using conjugate search directions," Ph.D. Thesis, University of London (1979).

13. Betts, J. T. "Algorithm 559, The stationary point of a quadratic function subject to linear constraints," ACM Transactions on Mathematical Software 6(3), 1980, pp. 391-397.

14. Betts, J. T. "A compact Algorithm for computing the stationary point of a quadratic formula subject to linear constraints," ACM Transactions on Mathematical Software 6(3), 1980, pp. 391-397.

15. Biggs, M. C. "Constrained minimization using recursive equality quadratic programming," In Numerical Methods for Nonlinear Optimization, F.A. Lootsma (Ed), Academic Press (London), 1972.

16. Biggs, M. C. "Constrained minimization using recursive quadratic programming: some alternative subproblem formulations," In Towards Global Optimization, L.C.W. Dixon and G. P. Szego (Eds), North-Holland Publishing Co. (Amsterdam) 1975.

17. Boot, J. C. G. "Notes on quadratic programming: the Kuhn-Tucker and Theil-Van de panne conditions, degeneracy and equality constraints," Management Science, 8, (1961), pp. 85-98.

18. Boot, J. C. G. Quadratic Programming Algorithms, Anomalies Applications. North-Holland Publishing Company, Amesterdam, 1964.

19. Bove, B. E. "The one-stage deformation method: an algorithm for quadratic programming," Econometrica, 38, (1970), pp. 225-230.

20. Braitsch, R. J. Jr. "A computer comparison of four quadratic programming algorithms," Management Science 11, (1972), pp. 632-643.

21. Bracken, J. and A. P. McCormick. Selected Applications of Nonlinear Programming, Wiley, New York, 1968.

22. Burdet, C. A. "General quadratic programming," Management Science Res. Report No. 272, G.S.I.A., Carnegie-Mellon Univ., Nov. 1971.

23. Cabot, A. V. and R. L. Francis. "Solving certain nonconvex minimization problems by ranking extreme points," Operations Research, 18, (1970), pp. 82-86.

24. Candler, W. and R. J. Townsley. "The maximization of a quadratic function of variables subject to linear inequalities," Management Science 10 (3), (1964), pp. 515-523.

25. Chen, J. "Feed formulations with a probabilistic constraint," Amer. J. Agric. Econ. 55, (1973), pp. 175-184.

26. Chiang, S.A. "Separable Programming Analysis of Spatial Competitive Market Models." M.S. Thesis, Department of Computer Science, Oklahoma State University, 1986.

27. Coffman, J., A. Majthay, and A. Whinston. "Local optimization for nonconvex quadratic programming," unpublished paper.

28. Cottle, R. W. "The principal pivoting method of quadratic programming," In Mathematics of the Decision Sciences (part 1), Dantzig and Veinott (Eds), Providence, R. I., American Math. Soc., 1968.

29. Cottle, R. W. and G. B. Dantzig. "Complementary pivot theory of mathematical programming," Linear Algebra and Applications, 1 (1968), pp. 103-125.

30. Cottle, R. W., and W. C. Mylander. "Ritter's cutting plane method for nonconvex quadratic programming," In Integer and Nonlinear programming, Abadie (Ed), Amesterdam, North-Holland publ. company., 1970.

31. Cutler, L. and D. S. Pass. "A computer program for quadratic mathematical models to be used for aircraft, design and other applications involving linear constraints," Report prepared for U.S.A.F. Project Rand, Rand Corp., Project No. R-516-PR, 1971.

32. D'Espo, D. "A convex programming procedure," Naval Research Logistics Quarterly, 6, (1959), pp. 33-42.

33. Dantzig, G. B. Linear Programming and Extensions. Princeton University Press, Princeton, N.J. 1963.

34. Davidon, W. C. "Variable metric method for minimization," AEC Research Development Report ANL-5990 (1959).

35. Dennis, J. B. Mathematical Programming and Electrical Networks, Wiley, N.Y. 1959.

36. Eaves, B. C. "The linear complementarity problem," Management Science, 17, (1971), pp. 612-634.

37. Fiacco, A. V. and G. P. McCormick. "The sequential unconstrained minimization technique for nonlinear programming," Management Science. 10(2), (1964), pp. 360-366.

38. Fletcher, R. "A Fortran Subroutine for quadratic programming," UKAEA Research Group Report, AERE R6370, 1970.

39. Fletcher, R. "The calculation of feasible points for linearly constrained optimization problems," UKAEA Research Group Report, AERE R6354, 1970.

40. Fletcher, R. "A general quadratic programming algorithm," J. Inst. Maths Applies, 7, (1971), pp. 76-91.

41. Fletcher, R. "An exact penalty function for nonlinear programming with inequalities," Math. Prog. 5, (1972), pp. 129-150.

42. Fletcher, R. "An ideal penalty function for constrained optimization," In Nonlinear Programming, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson (Eds), Academic Press (New York), 1975.

43. Fletcher, R. Practical Methods of Optimization, John Wiley and Sons, New York, 1981.

44. Frank, M. and P. Wolfe. "An algorithm for quadratic programming," Naval Research Logistics Quarterly 3, (1956), pp. 95-110.

45. Fukushima, M. "A successive quadratic programmin algorithm with global and superlinear properties," Math. Prog., 35, (1986), pp. 253-264.

46. Goldfarb, D. "Extensions of Newton's method and simplex methods for solving quadratic programs," In Conference on Numerical Methods for Nonlinear Optimization, F. A. Lootsma (Ed), University of Dundee, Academic Press, N. Y., 1971.

47. Goncalves, A. S. "A primal-dual method for quadratic programming," Revista da Fac., Ciencias Univ. Coimbra, vol. 47.

48. Goncalves, A. S. "A version of Beale's method avoiding the free variables," Proc. of the ACM National Conference, Chicago, 1971, pp. 433-441.

49. Goncalves, A. S. "A primal-dual method for quadratic programming with bounded variables," In Numerical Methods for Nonlinear Optimization, Lootsma (Ed), N.Y. Academic Press, 1972.

50. Goncalves, A. S. "A nonconvex quadratic programmin algorithm," In Mathematical Programming in Theory and Practice, Goncalves (Ed), Dordrecht (Holland), Reidel Publishing Co., 1973.

51. Graves, R. L. "A principal pivoting simplex algorithm for linear and quadratic programming," Operations Research, 15, (1967), pp. 482-494.

52. Han, S. P. "Superlinearly convergent variable metric algorithms for general nonlinear programming problems," Math. Prog., 11, (1976), pp. 263-282.

53. Han, S. P. "A globally convergent method for nonlinear programming," J. of Optimization Theory and Appls. 22, (1977), pp. 297-309.

54. Hazell, P. B. R. "A linear alternative to quadratic and semi-variance programming for farm planning under uncertainty," Amer. J. Agricultural Economics 53, (1971), pp. 53-62.

55. Hildreth, C. "A quadratic programming procedure," Naval Res. Logistics Quarterly 4, (1957), pp. 79-85.

56. Houthakker, H. S. "The capacity method of quadratic programming," Econometrica 28, (1960), pp. 62-87.

57. Jagannathan, R. "A simplex-type algorithm for linear and quadratic programming--a parametric procedure," Econometrica, 34, (1966), pp. 460-471.

58. Kaneko, T. "On some recent engineering applications of complementarity problems," Math. Prog., Study 17, (1982), pp. 111-125.

59. Keller, E. L. "The general quadratic optimization problem," Math. Prog., 5, (1973), pp. 311-337.

60. Land, A. H. and G. Morton. "An inverse-basis method for Beale's quadratic programming algorithm," Management Science 19, (1973), pp. 510-516.

61. Laughunn, D. J. "Quadratic binary programming with application to capital budgeting problems," Operations Res., 18, (3), (1970), pp. 454-461.

62. Lemke, C. E. "Bi-matrix equilibrium points and mathematical programming," Math. Science, 11, (1965), pp. 681-689.

63. Lemke, C. E. "On complementary pivot theory," In Mathematics of Decision Sciences, G. B. Dantzig and A. F. Veinott (Eds), 1968.

64. Louwes, S. L., J. C. A. Boot, and S. Wage, "A quadratic programming approach to the problem of optimal use of milk in Netherlands," J. Farm Econ., 45, (1963), pp. 309-317.

65. Mangasarian, O. L., and H. Stone. "Two person non-zero sum games and quadratic programming," J. Math. Anal. Appl., 9, (1964), pp. 348-355.

66. Markowitz, H. "Portfolio selection," J. Finance, 7, (1952), pp. 77-91.

67. Markowitz, H. "The optimization of quadratic formulas subject to linear constraints," Naval Res. Logistics Quarterly 3, (1956), pp. 111-133.

68. McCarl, B. A., H. Moskowitz and H. Furtan. "Quadratic programming applications," OMEGA, 5, (1977), pp. 43-55.

69. McMillan, C. Mathematical Programming, Wiley, New York, 2nd. Ed., 1975.

70. Moore, J. and A. Whinston. "Experimental methods on quadratic programming," Management Science, 13, (1966), pp. 58-76.

71. Mueller, R. K. "A method for solving the indefinite quadratic programming problem," Manag. Sci., 16, (1970), pp. 333-339.

72. Murray, W., and M. H. Wright. "Projected lagrangian methods based on the trajectories of penalty and barrier functions," Report SOL 78-23, Department of Operations Research, Stanford University, 1978.

73. Murtagh, B. A., and M. A. Saunders. "Large-scale linearly constrained optimization," Math. Prog., 14, (1978), pp. 41-72.

74. Murtagh, B., and M. Saunders. Modular In-Core Nonlinear Optimization System (MINOS), User's Manual, Stanford University, 1983.

75. Mylander, W. C. "Finite algorithms for solving quasi-convex quadratic problems," Operations Research, 20, (1972), pp. 167-173.

76. Nakamura, M. "Some programming problems in population projections," Operations Res. 21, (5), (1973), pp. 1048-1062.

77. Parsons, T. D. "A combinatorial approach to convex quadratic programming," Ph.D. Dissertation, Princeton University, Dept. of Math., 1966.

78. Powell, M. J. D. "A fast algorithm for nonlinearly constrained optimization calculations," In Numerical Analysis G. A. Watson (Ed), Dundee 1977, Lecture notes in Mathematics 630, Springer-Verlag (Berlin), 1978.

79. Powell, M. J. D. "The convergence of variable metric methods for nonlinear constrained optimization calculations," In Nonlinear Programming 3, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson (Eds), Academic Press (New York), 1978.

80. Powell, M. J. D. "An upper triangular matrix method for quadratic programming," Presented at the symposium: Nonlinear Programming 4, Madison, Wisconsin, July 1980.

81. Powell, M. J. D. "Variable metric methods for constrained optimization," In Mathematical Programming, The State of Art, A. Bachem, M. Grotschel, and B. Korte (Eds), Springer-Verlag (Berlin), 1983.

82. Powell, M. J. D. "The performance of two subroutines for constrained optimization on some difficult test problems," In Numerical Optimization, P. T. Boggs, R. H. Boyd, and R. B. Schnabel (Eds), 1984, (SIAM), Philadelphia, 1985.

83. Powell, M. J. D., and Y. Yuan. "A recursive quadratic programming algorithm that uses differentiable exact penalty functions," Math. Prog., 35, (1986), pp. 265-278.

84. Proll, L. A. "Remark on algorithm 431," Communications of the ACM, 17 (10), (1974), pp. 590.

85. Raghavachari, M. "On connections between zero-one integer programming and concave programming under linear constraints," Operations Res., 17, (4), (1969), pp. 680-684.

86. Ravindran, A. "Algorithm 431, a computer routine for quadratic and linear programming problems," Communications of the ACM, 15 (9), (1972), pp. 818-820.

87. Ravindran, A., and H. K. Lee. "Computer experiments on quadratic programming algorithms," European Journal of Operational Research, 8, (1981), pp. 166-174.

88. Ritter, K. "Stationary points of a quadratic maximum problems," Zeitschrift fur Wahrscheinlichkeitstheorie und verwandte Gebiete, 4, (1965), pp. 149-158.

89. Ritter, K. "A method for solving maximum problems with a nonconcave quadratic objective function," Zeitschritt fur Wahrscheinlichkeitstheorie und verwandte Gebiete, 4, (1966), pp. 340-351.

90. Rosen, J. B. and S. Suzuki. "Construction of nonlinear programming test problems," Communications of thhe ACM, 8 (2), (1965), pp. 113.

91. Rusin, M. H. "A revised simplex method for quadratic programming," SIAM Journal of Appl. Math, 20, (1971), pp. 143-160.

92. Sacher, R. S. "A decomposition algorithm for quadratic programming," Math. Prog. 18, (1980), pp. 16-30.

93. Schittkowski, K. "The nonlinear programming method of Wilson, Han, and Powell with an augmented lagrangian type line search function, part I: convergence analysis," Numerische Mathematik 38, (1981), pp. 83-114.

94. Schittkowski, K. "On the convergence of a sequential quadratic programming method with an augmented lagrangian line search function," Mathematische Operationsforschung und Statistik, Ser. Optimization, 14, (1983), pp. 197-216.

95. Shetty, C. M. "A simplified procedure for quadratic programming," Operations Research, 11, (1963), pp. 248-260.

96. Stoecker, A. L. "A quadratic programming model of United States agriculture in 1980: theory and application," Ph.D. Thesis, Iowa State University.

97. Taha, H. A. "Concave minimization over a convex polyhedron," Naval Research Logistics Quarterly, 20, (1973), pp. 533-548.

98. Takayama, T. "An application of spatial and temporal price equilibrium model to world energy modeling," Regional Science Assoc. Papers, 41, (1979), pp. 43-58.

99. Tapia, R. A. "Diagonalized multiplier methods and quasinewtion methods for constrained optimization," J. Optim. Theory and Appls., 22, (1977), pp. 135-194.

100. Theil, H. and G. Ray. "A quadratic programming approach to the estimation of transition probabilities," Management Science, 12, (1966), pp. 714-722.

101. Theil, H. and C. Van de panne. "Quadratic programming as an extension of conventional quadratic maximization," Management Science, 7, (1), (1960), pp. 1-20.

102. Todd, M. J. "A generalized complementary pivoting algorithm," Math. Prog. 6, (1974), pp. 243-263.

103. Tone, K. "Revisions of constraint approximations in the successive quadratic programming methods for nonlinear programming problems," Math. Prog., 26, (1983), pp. 144-152.

104. Townsley, R. "Derivation of optimal livestock rations using quadratic programming," J. Agric. Econ., 19, (1968), pp. 347-354.

105. Townsley, R. J. and W. Candler. "Quadratic as parametric linear programming," Naval Res. Log. Quarterly, 19, (1), (1972), pp. 183-189.

106. Tucker, A. W. "A least-distance approach to quadratic programming," In Math of Decision Sciences, G. B. Dantzig and A. F. Veinott (Eds), 1968.

107. Tui, H. "Concave programming under linear constraints," Soviet Mathematics, 1965, pp. 1437-1440.

108. Van de panne, C. "A non-artificial simplex method for quadratic programming," Report 22 of the Int'l Center for Management Science, 1962.

109. Van de panne, C. "A parametric method for general quadratic programming," Discussion papers, Series No. 28, The Univ. of Calgary, Department of Economics, Nov. 1973.

110. Van de panne, C. Methods for Linear and Quadratic Programming, North Holland Publishing Company, Amesterdam, 1974.

111. Van de panne, C., and A. Whinston. "A comparison of two methods for quadratic programming," Operations Research 14 (1966), pp. 422-441.

112. Van de panne, C., and A. Whinston. "The simplex and dual method for quadratic programming," Operations Research Quarterly 15, (1964), pp. 355-388.

113. Whinston, A. "A decomposition algorithm for quadratic programming," Cowles Foundation Duscussion, paper No. 172.

114. Wilde, D. J. and C. S. Beightler. Foundation of Optimization, Prentice-Hall, Englewood Cliffs, N.J., 1967.

115. Wilson, R. B. "A simplical algorithm for concave programming," Ph.D. dissertation, Graduate School of Business Administration, Harvard University, Boston 1963.

116. Wolfe, P. "The simplex method for quadratic programming," Econometrica, 27, (1959), pp. 382-398.

117. Wolfe, P. "Methods of nonlinear programming," In Recent Advances in Mathematical Programming, R. L. Graves and P. Wolfe (Eds), 1963.

118. Wood, M. J. "The February 1975 state of Build," Ministry of Works and Development Report (February 1975), Wellington, New Zealand.

119. Zahl, S. "A deformation method for quadratic programming," J. of the Royal Statistical Society, Series B, 26, (1964), pp. 141-160.

120. Zahl, S. "Supplement to a deformation method for quadratic programming," Journal of the Royal Statistical Society, Series B, 27, (1965), pp. 166-168.

121. Zangwill, W. I. "The convex simplex method," Management Science, 14, (3), (1967), pp. 221-238.

122. Zwart, P. B. "Global maximization of a convex function with linear inequality constraints," Operations Research, 22, (1974), pp. 602-609.

APPENDIX A

LISTING OF RAVINDRAN'S IMPLEMENTATION OF LEMKE'S ALGORITHM

```
//U10832A   JOB (10832,269-34-0589),'F. M. KHALILI',TIME=(,5),
// CLASS=2,MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=*
/*PASSWORD ?
/*JOBPARM ROOM=F,FORMS=9031
// EXEC FORTVCLG,IMSL=DP,REGION.GO=1500K
//FORT.SYSIN DD *
C************************************************************C
C*                                                          *C
C*  MODIFIED RAVINDRAN'S IMPLEMENTATION OF LEMKE'S ALGORITHM *C
C*                                                          *C
C************************************************************C
C*                                                          *C
C*   MODIFIED BY : FOUAD M. KHALILI                         *C
C*   DATE        : NOV. 20, 1987                            *C
C*                                                          *C
C************************************************************C
C*
C*
C*
       IMPLICIT REAL*8(A-H,O-Z)
       PARAMETER(N=100)
       PARAMETER(M=200)
       DIMENSION
     1 C(N),Q(N,N),A(N,N),RES1(N),RES2(N),ATRANS(N,N),BMAT(M,M),
     2 B(N),X(N),U(N),AM(M,M),QV(M),W(M),Z(M),AV(M),MBSIS(2*M)
       DIMENSION QI(15,15),D(15),WK(20),ZZ(15,15)
       COMMON    AM,AV,BMAT,W,Z,QV,L1,NL1,NL2,NE1,NE2,IR,MBSIS
       IN = 5
       IOUT = 6
       TYPE = 1.0D0
       SEED = 50.0D0
       NOFROW = 15
       NOFCOL = 15
       NOACTV = 2
       NOZERO = 5
C** GENERATE X AND U VECTORS
       DO 100 I = 1,NOFCOL
          CALL GENRTE(SEED,RANDOM)
          X(I) = RANDOM
100    CONTINUE
       DO 110 I = 1,NOFROW
          CALL GENRTE(SEED,RANDOM)
          U(I) = RANDOM
110    CONTINUE
       DO 120 I = 1,NOFROW-NOACTV
          U(I) = 0.0D0
120    CONTINUE
C** GENERATE MATRIX A (OR CTRANS IN FLETCHER'S PAPER)
       DO 200 I = 1,NOFROW
       DO 200 J = 1,NOFCOL
          CALL GENRTE(SEED,RANDOM)
          IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
          A(I,J) = RANDOM
200    CONTINUE
       DO 1700 I = 1,NOFCOL
       DO 1700 J = NOFCOL+1,NOFROW+NOFCOL
          AM(I,J) = -A(J-NOFCOL,I)
1700   CONTINUE
       DO 1800 I = NOFCOL+1,NOFROW+NOFCOL
       DO 1800 J = 1,NOFCOL
```

```
          AM(I,J) = A(I-NOFCOL,J)
1800    CONTINUE
C** GENERATE MATRIX Q (OR A IN FLETCHER'S PAPER)
        DO 300 I = 1,NOFCOL
        DO 300 J = 1,NOFCOL
           IF (I.GT.J) GO TO 300
           CALL GENRTE(SEED,RANDOM)
           IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
           ATRANS(I,J) = RANDOM
300     CONTINUE
        DO 1000 I = 1,NOFCOL
        DO 1000 J = 1,NOFCOL
           IF (I.LE.J) GO TO 1000
           ATRANS(I,J) = ATRANS(J,I)
1000    CONTINUE
C** TYPE = 0.=> Q IS INDEFINITE
C** TYPE = 1.=> Q IS POSITIVE DEFINITE
        IF (TYPE.EQ.0.0D0) GO TO 10
        CALL MULT(ATRANS,ATRANS,NOFCOL,NOFCOL,NOFCOL,Q,N,N,N,N)
        DO 1200 I = 1,NOFCOL
        DO 1200 J = 1,NOFCOL
           IF (I.EQ.J)Q(I,J) = Q(I,J) + 1.0D0
1200    CONTINU
        GO TO 40
10      DO 800 I = 1,NOFCOL
        DO 800 J = 1,NOFCOL
           Q(I,J) = ATRANS(I,J)
800     CONTINUE
40      DO 810 I = 1,NOZERO
        DO 810 J = NOFCOL-NOZERO+1,NOFCOL
           Q(I,J) = 0.0D0
810     CONTINUE
        DO 860 I = NOFCOL-NOZERO+1,NOFCOL
        DO 860 J = 1,NOZERO
           Q(I,J) = 0.0D0
860     CONTINUE
        DO 1600 I = 1,NOFCOL
        DO 1600 J = 1,NOFCOL
           AM(I,J) = 2.0D0*Q(I,J)
1600    CONTINUE
C** COMPUTE VECTOR C (OR B IN FLETCHER'S PAPER)
        DO 700 I = 1,NOFCOL
        DO 700 J = 1,NOFROW
           ATRANS(I,J) = A(J,I)
700     CONTINUE
        CALL MULT(ATRANS,U,NOFCOL,NOFROW,1,RES1,N,N,N,1)
        CALL MULT(Q,X,NOFCOL,NOFCOL,1,RES2,N,N,N,1)
        DO 400 I = 1,NOFCOL
           C(I) = RES1(I) - 2.0D0*RES2(I)
400     CONTINUE
C** COMPUTE VECTOR B ( OR D IN FLETCHER'S PAPER)
        CALL MULT(A,X,NOFROW,NOFCOL,1,B,N,N,N,1)
        DO 900 I = 1,NOFCOL
           IF (X(I).GT.0.0D0) GO TO 900
           CALL GENRTE(SEED,RANDOM)
           C(I) = C(I) + RANDOM
900     CONTINUE
        DO 910 I = 1,NOFROW
           IF (U(I).GT.0.0D0) GO TO 910
           CALL GENRTE(SEED,RANDOM)
```

```
         B(I) = B(I) - RANDOM
910     CONTINUE
        DO 1900 I = 1,NOFCOL
           QV(I) = C(I)
1900    CONTINUE
        DO 2000 I = NOFCOL+1,NOFROW+NOFCOL
           QV(I) = -B(I-NOFCOL)
2000    CONTINUE
        CALL LEMKES(NOFROW+NOFCOL)
        STOP
        END
C*
C*
C*************************************************************C
C*                                                           *C
C*  SUBROUTINE MULT : MULTIPLIES TWO MATRICES RLEFT AND RIGHT.*C
C*  ARGUMENTS :                                              *C
C*      RLEFT : THE FIRST MATRIX                             *C
C*      RIGHT : THE SECOND MATRIX                            *C
C*      LEFTR : ROW SIZE OF THE FIRST MATRIX                 *C
C*      LEFTC : COLUMN SIZE OF THE FIRST MATRIX              *C
C*      IRIHTC: COLUMN SIZE OF THE SECOND MATRIX             *C
C*      ID1   : ROW DIMENSION OF THE FIRST MATRIX            *C
C*      ID2   : COLUMN DIMENSION OF THE FIRST MATRIX         *C
C*      ID3   : ROW DIMENSION OF THE SECOND MATRIX           *C
C*      ID4   : COLUMN DIMENSION OF THE SECOND MATRIX        *C
C*      RESULT: MULTIPLICATION RESULT                        *C
C* INPUT      :                                              *C
C*      RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,ID1,ID2,ID3,ID4       *C
C* OUTPUT     :                                              *C
C*      RESULT                                               *C
C*                                                           *C
C*************************************************************C
C*
C*
       SUBROUTINE MULT(RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,RESULT,ID1,ID2,
     1 ID3,ID4)
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION RLEFT(ID1,ID2),RIGHT(ID3,ID4),RESULT(ID1,ID4)
       DO 100 I = 1,LEFTR
       DO 100 J = 1,IRIHTC
          RESULT(I,J) = 0.0D0
100    CONTINUE
       DO 200 I = 1,LEFTR
       DO 300 J = 1,IRIHTC
       DO 400 K = 1,LEFTC
          RESULT(I,J) = RESULT(I,J) + RLEFT(I,K)*RIGHT(K,J)
400    CONTINUE
300    CONTINUE
200    CONTINUE
       RETURN
       END
C*
C*
C************************************************************C
C*                                                          *C
C*  SUBROUTINE GENRTE : GENERATES A REAL NUMBER RANDOMLY    *C
C*  ARGUMENTS :                                             *C
C*    SEED    :  THE SEED FOR THE GENERATOR                 *C
C*    RANDOM  :  THE GENERATED NUMBER                       *C
```

```
C*   INPUT :                                                     *C
C*     SEED                                                      *C
C*   OUTPUT:                                                     *C
C*     SEED,RANDOM                                               *C
C*                                                               *C
C****************************************************************C
C*
C*
      SUBROUTINE GENRTE(SEED,RANDOM)
      IMPLICIT REAL*8(A-H,O-Z)
      X = 3373.0D0
      Y = 6925.0D0
      WORD = 32768.0D0
      TMAX = 24.0D0
      ONE = 1.0D0
      SEED = DMOD((X*SEED + Y),WORD)
      RANDOM = INT(TMAX*(SEED/WORD) + ONE)
      RETURN
      END
C*
C*
C*
C*****************************************************************C
C*                                                               *C
C* PROGRAM FOR SOLVING LINEAR AND QUADRATIC PROGRAMMING          *C
C* PROBLEMS IN THE FORM W=M*Z+Q, Q.Z=0, W AND Z NONNEGATIVE      *C
C* BY LEMKE'S ALGORITHM.                                         *C
C*                                                               *C
C* THE SUBROUTINE CALLS SIX SUBROUTINES. THESE ARE : MATRX,      *C
C* INITL,NEWBS,SORT,PIVOT AND PRINT IN PROPER ORDER.             *C
C* INPUT :                                                       *C
C*   N : THE SIZE OF ARRAY AM                                    *C
C*                                                               *C
C* DESCRIPTION OF PARAMETERS IN COMMON                           *C
C*  AM       A TWO DIMENSIONAL ARRAY CONTAINING THE              *C
C*           ELEMENTS OF MATRX M.                                *C
C*  Q        A SINGLY SUBSCRIPTED ARRAY CONTAINING THE           *C
C*           ELEMENTS OF VECTOR Q.                               *C
C*  L1       AN INTEGER VARIABLE INDICATING THE NUMBER OF        *C
C*           ITERATIONS TAKEN FOR EACH PROBLEM.                  *C
C*  B        A TWO DIMENSIONAL ARRAY CONTAINING THE              *C
C*           ELEMENTS OF THE INVERSE OF THE CURRENT BASIS.       *C
C*  W        A SINGLY SUBSCRIPTED ARRAY CONTAINING THE VALUES    *C
C*           OF W VARIABLES IN EACH SOLUTION.                    *C
C*  Z        A SINGLY SUBSCRIPTED ARRAY CONTAINING THE VALUES    *C
C*           OF Z VARIABLES IN EACH SOLUTION.                    *C
C*  NL1      AN INTEGER VARIABLE TAKING VALUE 1 OR 2 DEPEND-     *C
C*           ING ON WHETHER VARIABLE W OR Z LEAVES THE BASIS     *C
C*  NE1      SIMILAR TO NL1 BUT INDICATES VARIABLE ENTERING      *C
C*  NL2      AN INTEGER VARIABLE INDICATING WHAT COMPONENT       *C
C*           OF W OR Z VARIABLE LEAVES THE BASIS.                *C
C*  NE2      SIMILAR TO NL2 BUT INDICATES VARIABLE ENTERING      *C
C*  A        A SINGLY SUBSCRIPTED ARRAY CONTAINING THE           *C
C*           ELEMENTS OF THE TRANSFORMED COLUMN THAT IS          *C
C*           ENTERING THE BASIS.                                 *C
C*  IR       AN INTEGER VARIABLE DENOTING THE PIVOT ROW AT       *C
C*           EACH ITERATION. ALSO USED TO INDICATE TERMINA-      *C
C*           TION OF A PROBLEM BY GIVING IT A VALUE OF 1000.     *C
C*  MBSIS  A SINGLY SUBSCRIPTED ARRAY-INDICATOR FOR THE          *C
C*           BASIC VARIABLES. TWO INDICATORS ARE USED FOR        *C
```

```
C*          EACH BASIC VARIABLE-ONE INDICATING WHETHER          *C
C*          IT IS A W OR Z AND ANOTHER INDICATING WHAT          *C
C*          COMPONENT OF W OR Z.                                *C
C*                                                              *C
C****************************************************************C
C*
C*
      SUBROUTINE LEMKES(N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(200,200),Q(200),B(200,200),A(200)
      DIMENSION W(200),Z(200),MBSIS(400)
C
      COMMON AM,A,B,W,Z,Q,L1,NL1,NL2,NE1,NE2,IR,MBSIS
      IOUT=6
      IN=5
C
      IP = 1
C
C VARIABLE NO INDICATES THE CURRENT PROBLEM BEING SOLVED
C
      NO=0
 1000 NO=NO+1
      IF(NO-IP)1010,1010,1070
 1010 WRITE(IOUT,1020)NO
 1020 FORMAT (1H1,10X,11HPROBLEM NO.,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)
      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(200,200),Q(200),B(200,200),A(200)
      DIMENSION W(200),Z(200),MBSIS(400)
C
      COMMON AM,A,B,W,Z,Q,L1,NL1,NL2,NE1,NE2,IR,MBSIS
C
      IOUT=6
      IN=5
      RZERO=0.0D0
```

```
      RONE=1.0D0
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(200,200),Q(200),B(200,200),A(200)
      DIMENSION W(200),Z(200),MBSIS(400)
C
      COMMON AM,A,B,W,Z,Q,L1,NL1,NL2,NE1,NE2,IR,MBSIS
C
      IOUT=6
      RZERO=0.0D0
      TNONE=-1.0D0
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
        IZR = IR
      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
      WRITE(IOUT,3080)
 3080 FORMAT (3(/),5X,29HINITIAL ALMOST COMPLEMENTARY ,
     *   8HSOLUTION)
      DO 3100 I=1,N
        WRITE(IOUT,3090)I,W(I)
 3090   FORMAT (10X,2HW(,I4,2H)=,D20.7)
 3100   CONTINUE
      WRITE(IOUT,3110)Z0
 3110 FORMAT (10X,3HZO=,D20.7)
      RETURN
 3120 WRITE(IOUT,3130)
 3130 FORMAT (5X,36HPROBLEM HAS A TRIVIAL COMPLEMENTARY ,
     *   23HSOLUTION WITH W=Q, Z=0.)
      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(200,200),Q(200),B(200,200),A(200)
      DIMENSION W(200),Z(200),MBSIS(400)
C
      COMMON AM,A,B,W,Z,Q,L1,NL1,NL2,NE1,NE2,IR,MBSIS
C
      IOUT=6
      RZERO=0.0D0
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
 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)
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(200,200),Q(200),B(200,200),A(200)
      DIMENSION W(200),Z(200),MBSIS(400)
C
      COMMON AM,A,B,W,Z,Q,L1,NL1,NL2,NE1,NE2,IR,MBSIS
C
      AMAX = ABS(A(1))
      DO 10 I = 2,N
          IF (AMAX.GE.ABS(A(I))) GO TO 10
          AMAX = ABS(A(I))
   10 CONTINUE
      NB = 15
      TOL = AMAX*2.0D0**(-NB)
C** IN ANY ACTUAL IMPLEMENTATION NB SHOULD BE REPLACED BY B-11
C** WHERE B IS THE NO. OF BITS IN THE FLOATING POINT MANTISSA
      IOUT=6
      I=1
 5000 IF(A(I).GT.TOL) GO TO 5030
 5010 I=I+1
      IF(I-N)5020,5020,5130
 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).GT.TOL) GO TO 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
 5130 IF (Q(IZR).GT.TOL) GO TO 5100
       WRITE(IOUT,5140)
 5140  FORMAT(5X,'COMPLEMENTARY SOLUTION')
       CALL PRINT(N)
       IR = 1000
       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)
      WRITE(IOUT,5120)L1
 5120 FORMAT (10X,13HITERATION NO.,I4)
      IR=1000
```

```
      RETURN
      END
      SUBROUTINE PIVOT (N)
C
C PURPOSE - TO PERFORM THE PIVOT OPERATION BY UPDATING THE
C           INVERSE OF THE BASIS AND 0 VECTOR.
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AM(200,200),Q(200),B(200,200),A(200)
      DIMENSION W(200),Z(200),MBSIS(400)
C
      COMMON AM,A,B,W,Z,Q,L1,NL1,NL2,NE1,NE2,IR,MBSIS
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(200,200),Q(200),B(200,200),A(200)
      DIMENSION W(200),Z(200),MBSIS(400)
C
      COMMON AM,A,B,W,Z,Q,L1,NL1,NL2,NE1,NE2,IR,MBSIS
C
      IOUT=6
      RZERO=0.0D0
      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
 7040 WRITE(IOUT,7050)K1,Q(J)
 7050 FORMAT (10X,2HZ(,I4,2H)=,D20.7)
      GO TO 7080
 7060 WRITE(IOUT,7070)K1,Q(J)
```

```
 7070 FORMAT (10X,2HW(,I4,2H)=,D20.7)
 7080 I=I+1
      J=J+1
      IF(J-N)7010,7010,7090
7090  RETURN
      END
//
```

APPENDIX B

FLETCHER'S ALGORITHM LISTING

```
//U10832A   JOB (10832,269-34-0589),'F. M. KHALILI',TIME=(1,0),
// CLASS=2,MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=*
/*PASSWORD ?
/*JOBPARM ROOM=F,FORMS=9031
// EXEC FORTVCLG,IMSL=DP,REGION.GO=5000K
//FORT.SYSIN DD *
C**********************************************************C
C** THIS IS THE LISTING FOR FLETCHER'S ALGORITHM.         **C
C**********************************************************C
C*                                                        *C
C*  MODIFIED BY : FOUAD M. KHALILI                        *C
C*  DATE : NOV. 20, 1987                                  *C
C**********************************************************C
C*
C*
       PARAMETER(N=200)
       PARAMETER(M=700)
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION
     1 C(N),Q(N,N),A(N,N),RES1(N),RES2(N),ATRANS(N,N),
     2 B(N),X(M),U(N),BDL(N),BDU(N),H(N,N),LT(N)
       IN = 5
       IOUT = 6
       TYPE = 1.0D0
       MODE = 2
       IF (TYPE.EQ.0.0D0) MODE = 1
       SEED = 78.0D0
       NOFROW = 15
       NOFCOL = 10
       NOACTV = 2
       NOZERO =  0
C** GENERATE X AND U VECTORS
       DO 100 I = 1,NOFCOL
          CALL GENRTE(SEED,RANDOM)
          X(I) = RANDOM
100    CONTINUE
       DO 110 I = 1,NOFROW
          CALL GENRTE(SEED,RANDOM)
          U(I) = RANDOM
110    CONTINUE
       DO 120 I = 1,NOFROW-NOACTV
          U(I) = 0.0D0
120    CONTINUE
C** GENERATE MATRIX A (OR CTRANS IN FLETCHER'S PAPER)
       DO 200 I = 1,NOFROW
       DO 200 J = 1,NOFCOL
          CALL GENRTE(SEED,RANDOM)
          IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
          A(I,J) = RANDOM
200    CONTINUE
C** GENERATE MATRIX Q (OR A IN FLETCHER'S PAPER)
       DO 300 I = 1,NOFCOL
       DO 300 J = 1,NOFCOL
          IF (I.GT.J) GO TO 300
          CALL GENRTE(SEED,RANDOM)
          IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
          ATRANS(I,J) = RANDOM
300    CONTINUE
       DO 1000 I = 1,NOFCOL
       DO 1000 J = 1,NOFCOL
```

```
          IF (I.LE.J) GO TO 1000
          ATRANS(I,J) = ATRANS(J,I)
1000   CONTINUE
C** TYPE = 0.=> Q IS INDEFINITE
C** TYPE = 1.=> Q IS POSITIVE DEFINITE
       IF (TYPE.EQ.0.0D0) GO TO 10
       CALL MULT(ATRANS,ATRANS,NOFCOL,NOFCOL,NOFCOL,Q,N,N,N,N)
       DO 1200 I = 1,NOFCOL
       DO 1200 J = 1,NOFCOL
          IF (I.EQ.J)Q(I,J) = Q(I,J) + 1.0D0
1200   CONTINUE
       GO TO 40
10     DO 800 I = 1,NOFCOL
       DO 800 J = 1,NOFCOL
          Q(I,J) = ATRANS(I,J)
800    CONTINUE
40     DO 810 I = 1,NOZERO
       DO 810 J = NOFCOL-NOZERO+1,NOFCOL
          Q(I,J) = 0.0D0
810    CONTINUE
       DO 860 I = NOFCOL-NOZERO+1,NOFCOL
       DO 860 J = 1,NOZERO
          Q(I,J) = 0.0D0
860    CONTINUE
C** COMPUTE VECTOR C (OR B IN FLETCHER'S PAPER)
       DO 700 I = 1,NOFCOL
       DO 700 J = 1,NOFROW
          ATRANS(I,J) = A(J,I)
700    CONTINUE
       CALL MULT(ATRANS,U,NOFCOL,NOFROW,1,RES1,N,N,N,1)
       CALL MULT(Q,X,NOFCOL,NOFCOL,1,RES2,N,N,N,1)
       DO 400 I = 1,NOFCOL
          C(I) = RES1(I) - 2.0D0*RES2(I)
400    CONTINUE
C** COMPUTE VECTOR B ( OR D IN FLETCHER'S PAPER)
       CALL MULT(A,X,NOFROW,NOFCOL,1,B,N,N,N,1)
       DO 900 I = 1,NOFCOL
          IF (X(I).GT.0.0D0) GO TO 900
          CALL GENRTE(SEED,RANDOM)
          C(I) = C(I) + RANDOM
900    CONTINUE
       DO 910 I = 1,NOFROW
          IF (U(I).GT.0.0D0) GO TO 910
          CALL GENRTE(SEED,RANDOM)
          B(I) = B(I) - RANDOM
910    CONTINUE
       DO 1110 I = 1,M
          X(I) =1.0D0
1110   CONTINUE
       DO 1100 I = 1,N
          BDU(I) = 24.0D0
          BDL(I) = 0.0D0
1100   CONTINUE
       IH = N
       IC = N
       IA = N
       K = 0
       KE = 0
       DO 140 I = 1,NOFCOL
       C(I) = -C(I)
```

```
      DO 140 J = 1,NOFROW
         ATRANS(I,J) = A(J,I)
140   CONTINUE
      DO 160 I = 1,NOFCOL
      DO 160 J = 1,NOFCOL
         Q(I,J) = 2.0D0*Q(I,J)
160   CONTINUE
      ICOUNT = 0
      CALL ACTIVE(NOFCOL,NOFROW+2*NOFCOL,Q,IA,C,ATRANS,IC,B,BDL,BDU,
     1 X,K,KE,H,IH,LT,MODE,ICOUNT)
      WRITE(IOUT,1400)ICOUNT
1400  FORMAT(2X,' NUMBER OF ITERATIONS FOR FLETCHER METHOD = ',I5)
      WRITE(IOUT,222)
222   FORMAT (1X,'THE SOLUTION VECTOR FOR THE PROBLEM IS : ')
      DO 1500 I = 1,NOFCOL
      WRITE(IOUT,111)I,X(I)
111   FORMAT(2X,' X(',I3,') = ',D20.7)
1500  CONTINUE
      STOP
      END
C*
C*
C*************************************************************C
C*                                                           *C
C*   SUBROUTINE MULT : MULTIPLIES TWO MATRICES RLEFT AND RIGHT.*C
C*   ARGUMENTS :                                             *C
C*       RLEFT : THE FIRST MATRIX                            *C
C*       RIGHT : THE SECOND MATRIX                           *C
C*       LEFTR : ROW SIZE OF THE FIRST MATRIX                *C
C*       LEFTC : COLUMN SIZE OF THE FIRST MATRIX             *C
C*       IRIHTC: COLUMN SIZE OF THE SECOND MATRIX            *C
C*       ID1   : ROW DIMENSION OF THE FIRST MATRIX           *C
C*       ID2   : COLUMN DIMENSION OF THE FIRST MATRIX        *C
C*       ID3   : ROW DIMENSION OF THE SECOND MATRIX          *C
C*       ID4   : COLUMN DIMENSION OF THE SECOND MATRIX       *C
C*       RESULT: MULTIPLICATION RESULT                       *C
C* INPUT       :                                             *C
C*       RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,ID1,ID2,ID3,ID4      *C
C* OUTPUT      :                                             *C
C*       RESULT                                              *C
C*                                                           *C
C*************************************************************C
C*
C*
      SUBROUTINE MULT(RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,RESULT,ID1,ID2,
     1 ID3,ID4)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION RLEFT(ID1,ID2),RIGHT(ID3,ID4),RESULT(ID1,ID4)
      DO 100 I = 1,LEFTR
      DO 100 J = 1,IRIHTC
         RESULT(I,J) = 0.0D0
100   CONTINUE
      DO 200 I = 1,LEFTR
      DO 300 J = 1,IRIHTC
      DO 400 K = 1,LEFTC
         RESULT(I,J) = RESULT(I,J) + RLEFT(I,K)*RIGHT(K,J)
400   CONTINUE
300   CONTINUE
200   CONTINUE
      RETURN
```

```
      END
C*
C*
C************************************************************C
C*                                                          *C
C*  SUBROUTINE GENRTE : GENERATES A REAL NUMBER RANDOMLY    *C
C*  ARGUMENTS :                                             *C
C*    SEED    :  THE SEED FOR THE GENERATOR                 *C
C*    RANDOM  :  THE GENERATED NUMBER                       *C
C*  INPUT :                                                 *C
C*    SEED                                                  *C
C*  OUTPUT:                                                 *C
C*    SEED,RANDOM                                           *C
C*                                                          *C
C************************************************************C
C*
C*
      SUBROUTINE GENRTE(SEED,RANDOM)
      IMPLICIT REAL*8(A-H,O-Z)
      X = 3373.0D0
      Y = 6925.0D0
      WORD = 32768.0D0
      TMAX = 24.0D0
      ONE = 1.0D0
      SEED = DMOD((X*SEED + Y),WORD)
      RANDOM = INT(TMAX*(SEED/WORD) + ONE)
      RETURN
      END
C*************************************************************C
C*                                                           *C
C*  SUBROUTINE ACTIVE SOLVES THE GENERAL QUADRATIC PROGRAMMING *C
C*  PROBLEM USING FLETCHER'S ACTIVE SET METHOD. THE METHOD IS  *C
C*  GIVEN BY R. FLETCHER ("A GENERAL QUADRATIC PROGRAMMING     *C
C*  ALGORITHM", J. INST. MATH. APPLCS.,7,(1971),PP. 76-91.)    *C
C*  PROGRAM SOURCE : UNITED KINGDOM ATOMIC ENERGY AUTHORITY,   *C
C*  RESEARCH GROUP REPORT, AERE - R 6370, "A FORTRAN SUBROUTINE*C
C*  FOR GENERAL QUADRATIC PROGRAMMING",R. FLETCHER, (1970).    *C
C*************************************************************C
C*                                                           *C
C*  MODIFIED BY : FOUAD M. KHALILI                           *C
C*  DATE : NOV. 20,1987.                                     *C
C*                                                           *C
C*************************************************************C
C*                                                           *C
C*  THE CALLING SEQUENCE FOR ACTIVE IS                       *C
C* CALL ACTIVE(N,M,A,IA,B,C,IC,D,BDL,BDU,X,K,KE,H,IH,LT,MODE, *C
C* ICOUNT)                                                   *C
C*  THE ARGUMENTS WILL BE DESCRIBED AS FOLLOWS :             *C
C*  N     : THE NUMBER OF VARIABLES.                         *C
C*  M     : THE TOTAL NUMBER OF CONSTRAINTS.                 *C
C*  A     : THE COEFFICIENTS OF THE QUADRATIC TERMS IN THE   *C
C*          QUADRATIC FUNCTION 1/2*X(TRANS)*A*X - B(TRANS)*X . *C
C*          A SHOULD BE A SYMMETRIC NXN MATRIX, NOTE ALSO THAT *C
C*          A FACTOR OF 1/2 OCCURS IN THE DEFINITION OF THE  *C
C*          FUNCTION.                                        *C
C*  IA    : THE FIRST DIMENSION OF A IN THE DIMENSION STATEMENT *C
C*          WHICH ALLOCATES SPACE TO A.                      *C
C*  B     : THE COEFFICIENTS OF THE LINEAR TERMS IN THE QUAD- *C
C*          RATIC FUNCTION GIVEN ABOVE. B SHOULD HAVE N ELEMENTS*C
C*  C     : THE CONSTRAINTS MATRIX : EACH COLUMN OF C CONTAINS *C
```

```
C*            THE COEFFICIENTS OF CONSTRAINT C(TRANS)*X >= D .     *C
C*            THERE ARE M-2N COLUMNS OF C, AND N ROWS.             *C
C*   IC     : THE FIRST DIMENSION OF C IN THE DIMENSION STATEMENT *C
C*            WHICH ALLOCATES SPACE TO C.                          *C
C*   D      : THE RIGHT-HAND SIDES OF THE CONSTRAINTS CORRESPOND-  *C
C*            ING TO C, THERE ARE M-2N ELEMENTS IN D.              *C
C*   BDL    : LOWER BOUNDS ON THE VARIABLES. BDL HAS N ELEMENTS,   *C
C*            THE ITH BEING THE BOUND ON THE ITH VARIABLE.         *C
C*   BDU    : UPPER BOUNDS ON THE VARIABLES, N ELEMENTS AGAIN.     *C
C*   X      : THE ESTIMATE OF THE SOLUTION VECTOR, AND WORKING     *C
C*            SPACE. X(1), X(2), ..., X(N) CONTAINS THE VALUE OF   *C
C*            VECTOR X WHICH MINIMIZES THE OBJECTIVE FUNCTION.     *C
C*            THERE SHOULD BE AT LEAST 2N+M OR 7N ELEMENTS IN X,   *C
C*            WHICH EVER IS GREATER, THE REMAINDER BEING USED FOR *C
C*            WORKING SPACE. ON ENTRY, WHEN MODE 1 OR 2 IS BEING   *C
C*            USED, THEN X(1), X(2), ..., X(N) MIGHT BE USED TO    *C
C*            DETERMINE WHICH BOUNDS TO INCLUDE FOR THE FIRST      *C
C*            TRIAL BASIS, AND SHOULD BE SET ACCORDINGLY. ON ENTRY*C
C*            WITH MODE 3, THE FIRST N ELEMENTS OF VECTOR X SHOULD*C
C*            BE SET TO A FEASIBLE POINT. NOTHING NEED TO BE SET   *C
C*            ON ENTRY WITH MODES 4 AND 5. FINALLY, THE GRADIENT   *C
C*            OF THE OBJECTIVE FUNCTION , A*X - B, WILL BE FOUND   *C
C*            IN X(6N+1), X(N6+2), ..., X(7N) ON EXIT. THIS CAN BE*C
C*            USED TO COMPUTE THE MINIMUM VALUE OF THE FUNCTION IF*C
C*            REQUIRED, USING F(X) = 1/2*X(TRANS)*(A*X - 2*B).     *C
C*   K      : THE NUMBER OF CONSTRAINTS IN THE BASIS. ON ENTRY IN *C
C*            MODES 1 AND 2, K SHOULD BE SET EQUAL TO THE NUMBER   *C
C*            OF CONSTRAINTS (EQUALITIES AND OTHER INEQUALITIES    *C
C*            OF TYPE C(TRANS)*X >= D) WHICH ARE TO APPEAR IN THE *C
C*            TRAIL VERTEX FOR SUBROUTINE VERTEX. WITH NO A-PRIORI*C
C*            KNOWLEDGE SET K = KE. IF K IS SET NOT EQUAL TO ZERO,*C
C*            THEN LT MUST ALSO BE SET APPROPRIATELY. ON ENTRY IN *C
C*            MODE 3, K MUST BE SET EQUAL TO ZERO. ON ENTRY IN     *C
C*            MODES 4 AND 5, K SHOULD CONTAIN THE NUMBER OF CONST-*C
C*            RAINTS TO APPEAR IN THE EP(EQUALITY PROBLEM); THIS   *C
C*            WILL USUALLY BE THE VALUE WHICH WAS LEFT ON EXIT     *C
C*            FROM PREVIOUS CALL OF ACTIVE. ON EXIT, K WILL ALWAYS*C
C*            CONTAIN THE NUMBER OF CONSTRAINTS IN THE FINAL       *C
C*            BASIS . IF NO FEASIBLE POINT EXISTS, THEN K IS SET *C
C*            EQUAL TO ZERO AND A DIAGNOSTIC IS PRINTED.           *C
C*   KE     : THE TOTAL NUMBER OF EQUALITY CONSTRAINTS IN THE      *C
C*            PROBLEM. SET KE = 0 IF THERE ARE NONE. KE MUST BE    *C
C*            LESS THAN OR EQUAL TO K.                             *C
C*   H      : WORKING SPACE. H IS 2NX2N MATRIX. ON ENTRY, NOTHING *C
C*            NEED BE SET EXCEPT IN MODE 5, WHEN IT MUST CONTAIN   *C
C*            THE CORRECT OPERATORS. THESE WILL USUALLY BE LEFT    *C
C*            BY A PREVIOUS CALL TO ACTIVE AND SHOULD NOT BE       *C
C*            CHANGED. ON EXIT, THE LEADING NXN PARTITION CONTAINS*C
C*            THE OPERATOR HAND PARTITION BELOW THIS ( ROWS N+1 TO*C
C*            N+K) CONTAINS THE OPERATOR C*. THE LATTER OPERATOR   *C
C*            CAN BE USED TO CALCULATE LAGRANGE MULTIPLIERS OF THE*C
C*            EP CORRESPONDING TO THE FINAL BASIS, IF REQUIRED.    *C
C*   IH     : THE FIRST DIMENSION OF H IN THE DIMENSION STATEMENT *C
C*            WHICH ALLOCATES SPACE TO H.                          *C
C*   LT     : INTEGER WORKING SPACE. THE CONSTRAINTS ARE NUMBERED *C
C*            AS FOLLOWS. LOWER BOUNDS FROM 1 TO N, UPPER BOUNDS   *C
C*            FROM N+1 TO 2N, OTHERS FROM 2N+1 TO M. ON EXIT,      *C
C*            LT(1), LT(2), ..., LT(K) STORE THE INDEX NUMBERS OF *C
C*            THE ACTIVE CONSTRAINTS. ON ENTRY, LT(1), LT(2), ...,*C
C*            LT(KE) MUST ALWAYS CONTAIN THE INDEX NUMBERS OF THE *C
```

```
C*          EQUALITY CONSTRAINTS. IN MODES 1 AND 2, LT(KE+1),    *C
C*          LT(KE+2), ..., LT(K) MUST ALSO CONTAIN THE INDEX     *C
C*          NUMBERS OF ANY OTHER CONSTRAINTS TO APPEAR IN THE    *C
C*          TRIAL VERTEX FOR SUBROUTINE VERTEX. IN MODES 4 AND   *C
C*          5, LT(KE+1), LT(KE+2), ..., LT(K) MUST CONTAIN THE   *C
C*          INDEX NUMBERS OF CONSTRAINTS OTHER THAN EQUALITIES   *C
C*          WHICH ARE TO APPEAR IN THE EP. HOWEVER, IN MODES 4   *C
C*          AND 5, LT WILL USUALLY HAVE BEEN SET FROM A PREVIOUS*C
C*          CALL OF ACTIVE AND SHOULD NOT BE CHANGED. LT MUST    *C
C*          HAVE AT LEAST 2N+M ELEMENTS, THE REMAINDER BEING     *C
C*          USED AS WORKING SPACE.                               *C
C*   MODE : AN INTEGER BETWEEN 1 AND 5 INDICATING THE MODE OF    *C
C*          USE OF THE SUBROUTINE.                               *C
C*          1 : FOR GENERAL QUADRATIC PROGRAMMING CASE.          *C
C*          2 : FOR A STRICTLY CONVEX OBJECTIVE FUNCTION CASE.   *C
C*          3 : SAME AS IN MODE 2 EXCEPT THAT A FEASIBLE POINT   *C
C*              MUST BE PROVIDED BY THE USER SO THAT THERE IS NO*C
C*              NEED TO CALL SUBROUTINE VERTEX.                  *C
C*          4 : FOR GENERAL PARAMETRIC PROGRAMMING.              *C
C*          5 : FOR RIGHT-HAND SIDE PARAMETRIC PROGRAMMING.      *C
C*   ICOUNT:THE NUMBER OF ITERATIONS THAT WAS REQUIRED TO FIND   *C
C*          THE OPTIMAL POINT.                                   *C
C*                                                               *C
C****************************************************************C
C*                                                               *C
C*   SUBROUTINES CALLED BY SUBROUTINE ACTIVE ARE :               *C
C*     VERTEX : TO FIND A VERTEX POINT (SEE DESCRIPTION BELOW). *C
C*     INNERP : TO COMPUTE THE INNER PRODUCT OF TWO VECTORS.     *C
C*     LINV2F : TO FIND THE INVERSE OF A MATRIX. THIS IS AN IMSL*C
C*              LIBRARY SUBROUTINE.                              *C
C*                                                               *C
C****************************************************************C
C*
C*
C*
      SUBROUTINE ACTIVE(N,M,A,IA,B,C,IC,D,BDL,BDU,X,K,KE,H,IH,LT,MODE,
     1 ICOUNT)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(IA,*),B(*),C(IC,*),D(*),BDL(*),BDU(*),X(*),
     1          H(IH,*),LT(*),WKAREA(11000),TEMP(200,200)
      LOGICAL RETEST,PASSIV,POSTIV
      RETEST = .FALSE.
      IOUT = 6
      IN = 5
      IX = 700
      IDGT = 5
      NN = N + N
      N3 = NN + N
      N4 = NN + NN
      N5 = N4 + N
      N6 = N5 + N
      IF (MODE.GE.3) GO TO 99
C**   CALL FEASIBLE VERTEX ROUTINE
  8   CALL VERTEX(N,M,C,IC,D,BDL,BDU,X,K,KE,H,IH,LT)
      IF (K.EQ.0) RETURN
      IF (MODE.EQ.2.AND..NOT.RETEST) GO TO 100
C** INITIAL OPERATORS H=0 AND CSTAR=C(-1) FROM VERTEX
      DO 60 I = 1,N
      DO 60 J = 1,N
         H(N+I,J) = H(I,J)
```

```
         H(I,J) = 0.0D0
 60    CONTINUE
       GO TO 120
 99    DO 1 I=1,M
          LT(NN+I) = 1
  1    CONTINUE
C** CONSTRAINTS INDEXED AS FOLLOWS :
C**                     EQUALITY = -1
C**                     ACTIVE   =  0
C**                     INACTIVE =  1
       IF (K.EQ.0) GO TO 100
       DO 2 I = 1,K
       J = 0
       IF (I.LE.KE) J = -1
  2    LT(NN+LT(I)) = J
100    IF (MODE.EQ.5.AND..NOT.RETEST) GO TO 109
C**  SET UP MATRIX AND RHS OF EQUATIONS GOVERNING EQUALITY PROBLEM
       DO 101 I = 1,N
       X(N+I) = B(I)
       DO 101 J = 1,N
101    H(I,J) = A(I,J)
       IF((MODE.EQ.2.OR.MODE.EQ.3).AND..NOT.RETEST) GO TO 200
       IF (K.EQ.0) GO TO 107
       DO 102 I = 1,K
          LI = LT(I)
          IF (LI.GT.NN) GO TO 105
          DO 103 J = 1,N
             H(J,N+I) = 0.0D0
             H(N+I,J) = 0.0D0
103       CONTINUE
          IF (LI.GT.N) GO TO 104
          H(N+I,LI) = 1.0D0
          H(LI,N+I) = 1.0D0
          X(NN+I) = BDL(LI)
          GO TO 108
104       LI = LI - N
          H(N+I,LI) = -1.0D0
          H(LI,N+I) = -1.0D0
          X(NN+I) = -BDU(LI)
          GO TO 108
105       LI = LI - NN
          DO 106 J = 1,N
             H(N+I,J) = C(J,LI)
             H(J,N+I) = C(J,LI)
106       CONTINUE
          X(NN+I) = D(LI)
108       DO 102 J = 1,K
             H(N+I,N+J) = 0.
102    CONTINUE
107    NK = N + K
C** INVERT MATRIX GIVING OPERATORS H AND CSTAR
C**    CALL INVERT(H,NK,IH)
       CALL LINV2F(H,NK,IH,TEMP,IDGT,WKAREA,IER)
       DO 5100 I = 1,NK
       DO 5100 J = 1,NK
          H(I,J) = TEMP(I,J)
 5100  CONTINUE
       GO TO 118
C** SET UP RHS ONLY
109    DO 113 I = 1,N
```

```
          X(N+I) = B(I)
113    CONTINUE
       DO 115 I = 1,K
          LI = LT(I)
          IF (LI.GT.NN) GO TO 117
          IF (LI.GT.N) GO TO 116
          X(NN+I) = BDL(LI)
          GO TO 115
116       X(NN+I) = -BDU(LI-N)
          GO TO 115
117       X(NN+I) = D(LI-NN)
115    CONTINUE
C** SOLVE FOR SOLUTION POINT X
       NK = N + K
118    DO 119 I = 1,N
          CALL INNERP(H,X,IH,IH,IX,1,X(I),NK,1,2,N+1,N+2,0,0,I,1)
119    CONTINUE
C** CHECK FEASIBILITY, IF NOT EXIT TO 8
       DO 110 I = 1,M
          IF (LT(NN+I).LE.0) GO TO 110
          IF (I.GT.N) GO TO 111
          Z = X(I) - BDL(I)
          GO TO 114
111       IF (I.GT.NN) GO TO 112
          Z = BDU(I-N) - X(I-N)
          GO TO 114
112       J = I - NN
          CALL INNERP(C,X,IC,IC,IX,1,Z,N,1,2,1,2,0,0,J,1)
          Z =  Z - D(J)
114       IF (Z.LT.0.0D0) GO TO 8
110    CONTINUE
120    CONTINUE
C** CALCULATE GRADIENT G ANDLAGRANGE MULTIPLIERS -CSTAR.G,
C** FIND LARGEST MULTIPLIER, EXIT IF NOT POSITIVE
       DO 121 I = 1,N
          CALL INNERP(A,X,IA,IA,IX,1,X(N6+I),N,1,2,1,2,1,0,I,1)
          X(N6+I) =  X(N6+I) - B(I)
121    CONTINUE
       IF (K.EQ.0) RETURN
123    Z = -1.0D75
       DO 122 I = 1,K
          IF (LT(NN+LT(I)).EQ.-1) GO TO 122
          CALL INNERP(H,X,IH,IH,IX,1,ZZ,N,1,2,N6+1,N6+2,1,0,N+I,1)
          ZZ = -ZZ
          IF (ZZ.LE.Z) GO TO 122
          Z = ZZ
          II = I
122    CONTINUE
       IF (Z.GT.0.0D0) GO TO 130
       IF (RETEST.OR.MODE.GE.4) GO TO 137
       RETEST = .TRUE.
       GO TO 100
137    IF (Z.NE.0.0D0) RETURN
       WRITE(IOUT,1003)
1003   FORMAT ('0SOLUTION MAY BE A DEGENERATE LOCAL MINIMUM')
       RETURN
C** SET DIRECTION OF SEARCH AS CORRESPONDING ROW OF CSTAR
130    DO 131 I = 1,N
          X(NN+I) = H(N+II,I)
131    CONTINUE
```

```
136     DO 132 I = 1,N
           CALL INNERP(A,X,IA,IA,IX,1,X(N+I),N,1,2,NN+1,NN+2,1,0,I,1)
132     CONTINUE
        CALL INNERP(X,X,IX,1,IX,1,CAC,N,NN+1,NN+2,N+1,N+2,0,0,1,1)
        IF (CAC.GT.0.0D0) GO TO 134
        POSTIV = .FALSE.
        Y = 1.0D0
        GO TO 135
134     POSTIV = .TRUE.
        Y = Z/CAC
135     DO 133 I =1,N
           X(N5+I) = X(NN+I)*Y
133     CONTINUE
        PASSIV = .TRUE.
139     ALPHA = 1.0D75
        NK = N + K
C** LINEAR SEARCH ALONG DIRECTION OF SEARCH,PASSIV INDICATES
C** A CONSTRAINT HAS BEEN REMOVED TO GET SEARCH DIRECTION,
C** POSTIV INDICATES POSITIVE CURVATURE ALONG DIRECTION
        DO 140 I = 1,M
           IF (LT(NN+I).LE.0) GO TO 140
           IF (I.GT.N) GO TO 141
           IF (X(N5+I).GE.0.0D0)GO TO 140
           CC = (BDL(I) - X(I))/X(N5+I)
           GO TO 143
141        IF (I.GT.NN) GO TO 142
           IF (X(N4+I).LE.0.0D0) GO TO 140
           CC = (BDU(I-N) - X(I-N))/X(N4+I)
           GO TO 143
142        J = I - NN
           CALL INNERP(C,X,IC,IC,IX,1,ZZ,N,1,2,N5+1,N5+2,0,0,J,1)
           IF (ZZ.GE.0.0D0) GO TO 140
           CALL INNERP(C,X,IC,IC,IX,1,CC,N,1,2,1,2,0,0,J,1)
           CC = D(J) - CC
           CC = CC/ZZ
143        IF (CC.GE.ALPHA) GO TO 140
           ALPHA = CC
           IAL = I
140     CONTINUE
        IF (PASSIV) LT(NN+LT(II)) = 1
C** IF MINIMUM FOUND, GO TO 170
        IF(POSTIV.AND.ALPHA.GE.1.0D0) GO TO 170
C** CALCULATE H.C AND CSTAR.C
        DO 144 I = 1,N
           X(I) = X(I) + ALPHA*X(N5+I)
144     CONTINUE
        ALPHA = ALPHA*Y
        J = 1
        IF (K.EQ.N) J = N + 1
        IF (IAL.GT.N) GO TO 146
        DO 145 I = J,NK
           X(N3+I) = H(I,IAL)
145     CONTINUE
        CHC = X(N3+IAL)
        GO TO 151
146     IB = IAL - N
        IF (IB.GT.N) GO TO 148
        DO 147 I = J,NK
           X(N3+I) = -H(I,IB)
147     CONTINUE
```

```
      CHC = -X(N3+IB)
      GO TO 151
148   IB = IB - N
      DO 149 I = 1,N
         X(N5+I) = C(I,IB)
149   CONTINUE
      DO 150 I = J,NK
         CALL INNERP(H,X,IH,IH,IX,1,X(N3+I),N,1,2,N5+1,N5+2,1,0,I,1)
150   CONTINUE
      IF(K.NE.N)
     1 CALL INNERP(X,X,IX,1,IX,1,CHC,N,N5+1,N5+2,N3+1,N3+2,0,0,1,1)
151   LT(NN+IAL) = 0
      IF (K.EQ.N) GO TO 180
      IF (PASSIV) GO TO 160
C** APPLY FORMULA FOR ADDING A CONSTRAINT
156   IF (K.EQ.0) GO TO 157
      DO 152 I = 1,K
      ALPHA = X(N4+I)/CHC
      NI = N + I
      DO 152 J = 1,N
         H(NI,J) = H(NI,J) - ALPHA*X(N3+J)
152   CONTINUE
157   K = K + 1
      LT(K) = IAL
      DO 158 J = 1,N
         H(N+K,J) = X(N3+J)/CHC
158   CONTINUE
      IF(K.LT.N) GO TO 154
      DO 153 I = 1,N
      DO 153 J = 1,N
         H(I,J) = 0.0D0
153   CONTINUE
      GO TO 159
154   DO 155 I = 1,N
         ALPHA = X(N3+I)/CHC
         DO 155 J = 1,I
         H(I,J) = H(I,J) - ALPHA*X(N3+J)
         H(J,I) = H(I,J)
155   CONTINUE
159   ICOUNT = ICOUNT + 1
      IF(.NOT.PASSIV) GO TO 167
C** REMOVAL OF A CONSTRAINT HAS BEEN DEFERRED, SET UP AS IF
C** THE CONSTRAINT IS BEING REMOVED FROM AUGMENTED BASIS
      DO 164 I = 1,N
         CALL INNERP(A,X,IA,IA,IX,1,X(N6+I),N,1,2,1,2,1,0,I,1)
         X(N6+I) =  X(N6+I) - B(I)
         X(NN+I) = H(N+II,I)
164   CONTINUE
      CALL INNERP(X,X,IX,1,IX,1,Z,N,N6+1,N6+2,NN+1,NN+2,0,0,1,1)
      Z = -Z
      IF (Z.EQ.0.0D0) GO TO 178
      GO TO 136
160   CC = X(N4+II)
      Y = CHC*CAC + CC**2.0D0
      CALL INNERP(X,X,IX,1,IX,1,GHC,N,N6+1,N6+2,N3+1,N3+2,0,0,1,1)
      IF (ALPHA*Y.LT.CHC*(Z - ALPHA*CAC) + GHC*CC) GO TO 156
C** APPLY FORMULA FOR EXCHANGING NEW CONSTRAINT
C** WITH PASSIVE CONSTRAINT
      ICOUNT = ICOUNT + 2
      DO 161 I = 1,K
```

```
          NI = N + I
          CALL INNERP(H,X,IH,IH,IX,1,X(N5+I),N,1,2,N+1,N+2,1,0,NI,1)
161    CONTINUE
       DO 162 I = 1,N
          X(N+I) = (CHC*X(NN+I) - CC*X(N3+I))/Y
          X(N6+I) = (CAC*X(N3+I) + CC*X(NN+I))/Y
162    CONTINUE
       DO 163 I = 1,N
       DO 163 J = 1,I
          H(I,J) = H(I,J) + X(N+I)*X(NN+J) - X(N6+I)*X(N3+J)
          H(J,I) = H(I,J)
163    CONTINUE
       X(N4+II) = X(N4+II) - 1.0D0
       DO 166 I = 1,K
          NI = N + I
          DO 166 J = 1,N
             H(NI,J) = H(NI,J) - X(N4+I)*X(N6+J) - X(N5+I)*X(N+J)
166    CONTINUE
       LT(II) = IAL
167    IF(K.EQ.N) GO TO 120
C** CALCULATE G, NEW SEARCH DIRECTION IS -H.G
       DO 168 I = 1,N
          CALL INNERP(A,X,IA,IA,IX,1,X(N+I),N,1,2,1,2,1,0,I,1)
          X(N+I) =  X(N+I) - B(I)
168    CONTINUE
       Z = 0.0D0
       DO 169 I = 1,N
          CALL INNERP(H,X,IH,IH,IX,1,X(N5+I),N,1,2,N+1,N+2,1,0,I,1)
          X(N5+I) = -X(N5+I)
          IF (X(N5+I).NE.0.0D0) Z = 1.0D0
169    CONTINUE
       PASSIV = .FALSE.
       IF (Z.EQ.0.0D0) GO TO 120
       POSTIV = .TRUE.
       GO TO 139
170    DO 171 I = 1,N
          X(I) = X(I) + X(N5+I)
171    CONTINUE
C** X IS NOW THE MINIMUM POINT IN THE BASIS
C** UPDATE THE OPERATORS IF A CONSTRAINT HAD BEEN REMOVED
       IF (.NOT.PASSIV) GO TO 120
       ICOUNT = ICOUNT + 1
178    DO 172 I = 1,N
          ALPHA = X(NN+I)/CAC
          DO 172 J = 1,I
          H(I,J) = H(I,J) + ALPHA*X(NN+J)
          H(J,I) = H(I,J)
172    CONTINUE
       IF (K.GT.1) GO TO 177
       K = 0
       GO TO 120
177    IF (II.EQ.K) GO TO 175
       DO 174 I = 1,N
174    H(N+II,I) = H(N+K,I)
       LT(II) = LT(K)
175    K = K - 1
       DO 173 I = 1,K
          NI = N + I
          CALL INNERP(H,X,IH,IH,IX,1,X(N3+I),N,1,2,N+1,N+2,1,0,NI,1)
173    CONTINUE
```

```
      DO 176 I =1,K
      ALPHA = X(N3+I)/CAC
      NI = N + I
      DO 176 J = 1,N
         H(NI,J) = H(NI,J) - ALPHA*X(NN+J)
176   CONTINUE
      GO TO 120
180   Z = 1.0D0/X(N4+II)
C** APPLY SIMPLEX FORMULA TO EXCHANGE CONSTRAINTS
      ICOUNT = ICOUNT + 1
      DO 181 I = 1,N
         NI = N + I
         IF (I.NE.II) GO TO 182
         DO 183 J = 1,N
            H(NI,J) = H(NI,J)*Z
183      CONTINUE
         GO TO 181
182      ZZ = Z*X(N4+I)
         DO 184 J = 1,N
            H(NI,J) = H(NI,J) - ZZ*X(NN+J)
184      CONTINUE
181   CONTINUE
      LT(II) = IAL
      GO TO 120
200   K = 0
      IF (KE.NE.0) WRITE(IOUT,1002)
1002  FORMAT('0KE MUST BE 0 IN MODES 2 AND 3')
      KE = 0
      DO 202 I = 1,M
         LT(NN+I) = 1
202   CONTINUE
C**   CALL INVERT(H,N,IH)
      CALL LINV2F(H,N,IH,TEMP,IDGT,WKAREA,IER)
      DO 5200 I = 1,N
      DO 5200 J = 1,N
         H(I,J) = TEMP(I,J)
5200  CONTINUE
C** START WITH EMPTY BASIS FROM FEASIBLE POINT
C** SEARCH DIRECTION IS -A(-1).B
      GO TO 167
      END
C*
C*
C*****************************************************************C
C*                                                               *C
C*  SUBROUTINE INNERP : CALCULATE THE INNERPRODUCT OF TWO VECTORS *C
C*  IT MULTIPLIES THE TWO VECTORS THAT ARE EXTRACTED FROM ARRAYS  *C
C*  E & F. THE ELEMENTS ARE AT LOCATIONS I+(II-1)*(J-I) AND THE   *C
C*  ELEMENTS OF THE SECOND VECTOR ARE BEING STORED AT LOCATIONS   *C
C*  K+(II-1)*(L-K), WHERE II = 1,N.                               *C
C*  INPUT :                                                       *C
C*     E,F,IDIM1,IDIM2,IDIM3,IDIM4,N,I,J,K,L,I1,I2,N1,N2          *C
C*  OUTPUT:
C*     SUM
C*  ARGUMENTS :
C*     E,F,I,J,K,L DEFINED ABOVE.
C*     IDIM1 : ROW DIMENSION OF THE FIRST ARRAY FROM WHICH THE    *C
C*             FIRST VECTOR IS BEING EXTRACTED.                   *C
C*     IDIM2 : COLUMN DIMENSION OF THE FIRST ARRAY.               *C
C*     IDIM3 : ROW DIMENSION OF THE SECOND ARRAY FROM WHICH THE   *C
```

```
C*              SECOND VECTOR IS BEING EXTRACTED.                     *C
C*      IDIM4 : COLUMN DIMENSION OF THE SECOND VECTOR.                *C
C*      N     : NUMBER OF ELEMENTS TO BE MULTIPLIED.                  *C
C*      I1    : IF = 0 => EXTRACT ELEMENTS OF COLUMN N1 FROM E FOR    *C
C*              THE FIRST VECTOR; ELSE IF = 1 => EXTRACT ELEMENTS     *C
C*              OF ROW N1.                                            *C
C*      I2    : IF = 0 => EXTRACT ELEMENTS OF COLUMN N2 FROM F FOR    *C
C*              THE SECOND VECTOR; ELSE IF = 1 => EXTRACT ELEMENTS    *C
C*              OF ROW N2.                                            *C
C*      N1,N2  DEFINED ABOVE                                          *C
C*      SUM   : THE PRODUCT OF MULTIPLICATION                         *C
C*                                                                    *C
C* *******************************************************************C
C*
C*
      SUBROUTINE INNERP(E,F,IDIM1,IDIM2,IDIM3,IDIM4,SUM,N,I,J,K,L,
     1 I1,I2,N1,N2)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION E(IDIM1,IDIM2),F(IDIM3,IDIM4)
      SUM = 0.0D0
      DO 10  II = 1,N
      IF(I1.EQ.0) GO TO 100
      IF(I2.EQ.0) GO TO 200
      SUM = SUM + E(N1,I+(II-1)*(J-I))*F(N2,K+(II-1)*(L-K))
      GO TO 10
100   IF(I2.EQ.1) GO TO 300
      SUM = SUM + E(I+(II-1)*(J-I),N1)*F(K+(II-1)*(L-K),N2)
      GO TO 10
200   SUM = SUM + E(N1,I+(II-1)*(J-I))*F(K+(II-1)*(L-K),N2)
      GO TO 10
300   SUM = SUM + E(I+(II-1)*(J-I),N1)*F(N2,K+(II-1)*(L-K))
10    CONTINUE
      IF(DABS(SUM).LE.1.D-15) SUM = 0.0D0
      RETURN
      END
C*****************************************************************C
C*                                                               *C
C*  SUBROUTINE VERTEX FINDS A FEASIBLE VERTEX FOR A LINEARLY     *C
C*  CONSTRAINED FEASIBLE SOLUTION SPACE.                         *C
C*  PROGRAM SOURCE : UNITED KINGDOM ATOMIC ENERGY AUTHORITY,     *C
C*  RESEARCH GROUP REPORT, AERE - R 6354, "THE CALCULATION OF    *C
C*  FEASIBLE POINTS FOR LINEARLY CONSTRAINED OPTIMIZATION        *C
C*  PROBLEMS", R. FLETCHER, (1970).                              *C
C*****************************************************************C
C*                                                               *C
C*  MODIFIED BY : FOUAD M. KHALILI                               *C
C*  DATE : NOV. 20,1987.                                         *C
C*                                                               *C
C*****************************************************************C
C*                                                               *C
C*  THE CALLING SEQUENCE FOR ACTIVE IS                           *C
C* CALL VERTEX(N,M,C,IC,D,BDL,BDU,X,K,KE,H,IH,LT)                *C
C*  THE ARGUMENTS WILL BE DESCRIBED AS FOLLOWS :                 *C
C*  N     : THE NUMBER OF VARIABLES.                             *C
C*  M     : THE TOTAL NUMBER OF CONSTRAINTS.                     *C
C*  C     : THE CONSTRAINTS MATRIX : EACH COLUMN OF C CONTAINS   *C
C*          THE COEFFICIENTS OF CONSTRAINT C(TRANS)*X >= D .     *C
C*          THERE ARE M-2N COLUMNS OF C, AND N ROWS.             *C
C*  IC    : THE FIRST DIMENSION OF C IN THE DIMENSION STATEMENT  *C
C*          WHICH ALLOCATES SPACE TO C.                          *C
```

```
C*  D     : THE RIGHT-HAND SIDES OF THE CONSTRAINTS CORRESPOND- *C
C*          ING TO C, THERE ARE M-2N ELEMENTS IN D.            *C
C*  BDL   : LOWER BOUNDS ON THE VARIABLES. BDL HAS N ELEMENTS,  *C
C*          THE ITH BEING THE BOUND ON THE ITH VARIABLE.        *C
C*  BDU   : UPPER BOUNDS ON THE VARIABLES, N ELEMENTS AGAIN.    *C
C*  X     : POSITION OF THE VERTEX AND THE SPACE. ON EXIT,      *C
C*          X(1), X(2), ..., X(N) CONTAINS THE POSITION OF THE  *C
C*          FEASIBLE VERTEX. THERE SHOULD BE 2N+M ELEMENTS IN   *C
C*          X, THE REMAINING ELEMENTS N+M BEING USED AS WORKING*C
C*          SPACE. ON ENTRY, X(I) MIGHT BE USED TO DECIDE       *C
C*          WHETHER TO INCLUDE AN UPPER BOUND OR A LOWER BOUND  *C
C*          IN THE BASIS. THE CHOICE IS MADE BY INCLUDING THE   *C
C*          BOUND WHICH IS NEAREST TO X(I), OR THE LOWER BOUND  *C
C*          IN CASES OF EQUALITY. THE USER CAN THUS DETERMINE   *C
C*          THE CHOICE BY SETTING X SUITABLY ON ENTRY.          *C
C*  K     : THE TOTAL NUMBER OF DESIGNATED CONSTRAINTS TO APPEAR*C
C*          IN THE BASIS. SET K = 0 IF NONE. IF K <> 0 THEN LT  *C
C*          MUST BE SET AS INDICATED. ON EXIT K = N MEANS THAT  *C
C*          A FEASIBLE VERTEX HAS BEEN FOUND AND K = 0 MEANS    *C
C*          THAT NO FEASIBLE POINT EXISTS. A DIAGNOSTIC IS      *C
C*          PRINTED IN THE LATTER CASE.                         *C
C*  KE    : THE TOTAL NUMBER OF EQUALITY CONSTRAINTS IN THE     *C
C*          PROBLEM. SET KE = 0 IF THERE ARE NONE. KE MUST BE   *C
C*          LESS THAN OR EQUAL TO K.                            *C
C*  H     : WORKING SPACE. H IS NXN+K MATRIX, WHERE K IS THE    *C
C*          NUMBER OF INITIALLY DESIGNATED CONSTRAINTS. THE     *C
C*          LEADING NXN PARTITION OF H STORES THE INVERSE MATRIX*C
C*          CORRESPONDIND TO THE NORMALS OF CONSTRAINTS IN THE  *C
C*          BASIS.                                              *C
C*  IH    : THE FIRST DIMENSION OF H IN THE DIMENSION STATEMENT *C
C*          WHICH ALLOCATES SPACE TO H.                         *C
C*  LT    : INTEGER WORKING SPACE. THE CONSTRAINTS ARE NUMBERED *C
C*          AS FOLLOWS. LOWER BOUNDS FROM 1 TO N, UPPER BOUNDS  *C
C*          FROM N+1 TO 2N, OTHERS FROM 2N+1 TO M. LT(1), LT(2),*C
C*          ... , LT(N) STORE THE INDEX NUMBERS OF CONSTRAINTS  *C
C*          IN THE BASIS. THE MATRIX OF NORMALS OF ACTIVE       *C
C*          CONSTRAINTS C IS THUS THE COLUMNS OF (I,I,C)        *C
C*          CORRESPONDING TO LT(1), LT(2), ..., LT(N), (IN THAT *C
C*          ORDER) AND IS THEREFORE NOT STORED EXPLICITLY.      *C
C*          H ABOVE IS C(-1) DEFINED IN THIS WAY. ON ENTRY,     *C
C*          LT(1), LT(2), ..., LT(KE) MUST CONTAIN THE INDEX    *C
C*          NUMBERS OF EQUALITY CONSTRAINTS, AND LT(KE+1),      *C
C*          LT(KE+2), ..., LT(K) THE INDEX NUMBERS OF ANY       *C
C*          REMAINING DESIGNATED INEQUALITY CONSTRAINTS. LT MUST*C
C*          HAVE AT LEAST 2N+M ELEMENTS, THE REMAINDER BEING    *C
C*          USED AS WORKING SPACE.                              *C
C*                                                              *C
C***************************************************************C
C*                                                              *C
C*  SUBROUTINES CALLED BY SUBROUTINE VERTEX ARE :               *C
C*    INNERP : TO COMPUTE THE INNER PRODUCT OF TWO VECTORS.     *C
C*    LINV2F : TO FIND THE INVERSE OF A MATRIX. THIS IS AN IMSL*C
C*             LIBRARY SUBROUTINE.                              *C
C*                                                              *C
C***************************************************************C
C*
C*
C*
      SUBROUTINE VERTEX(N,M,C,IC,D,BDL,BDU,X,K,KE,H,IH,LT)
      IMPLICIT REAL*8(A-H,O-Z)
```

```
        DIMENSION C(IC,*),D(*),BDL(*),BDU(*),X(*),H(IH,*),LT(*),
      1 TEMP(200,200),WKAREA(11000)
        IN = 5
        IOUT = 6
        IX = 700
        IDGT = 5
        NN = N + N
        N3 = NN + N
        DO 1 I = 1,M
1       LT(NN+I) = 1
C** CONSTRAINTS INDEXED AS FOLLOWS :
C**                         EQUALITY = -1
C**                         ACTIVE   =  0
C**                         INACTIVE =  1
C**                         VIOLATED =  2
        IF (K.NE.0) GO TO 10
C**NO DESIGNATED CONSTRAINTS, VERTEX CHOSEN FROM UPPER AND
C** LOWER BOUNDS, INVERSE MATRIX TRIVIAL
        DO 4  I = 1,N
        DO 5  J = 1,N
           H(I,J) = 0.0D0
5       CONTINUE
        IF (X(I)-BDL(I).GT.BDU(I)-X(I)) GO TO 6
        LT(I) = I
        H(I,I) = 1.0D0
        GO TO 4
6       LT(I) = N + I
        H(I,I) = -1.0D0
4       LT(NN+LT(I)) = 0
        K = N
        GO TO 40
C** SET UP NORMALS V OF THE K DESIGNATED CONSTRAINTS IN BASIS
10      DO 11 I = 1,K
           J = 0
           IF (I.LE.KE) J = -1
           LT(NN+LT(I)) = J
           LI = LT(I)
           NI = N + I
           IF (LI.GT.NN) GO TO 14
           DO 12 J = 1,N
              H(J,NI) = 0.0D0
12         CONTINUE
           IF (LI.GT.N) GO TO 13
           H(LI,NI) = 1.0D0
           GO TO 11
13         H(LI-N,NI) = -1.0D0
           GO TO 11
14         LI = LI - NN
           DO 15 J = 1,N
              H(J,NI) = C(J,LI)
15         CONTINUE
11      CONTINUE
        IF (K.NE.N) GO TO 19
        DO 16 J = 1,N
           NJ = N + J
           DO 16 I = 1,N
              H(I,J) = H(I,NJ)
16      CONTINUE
C**     CALL INVERT(H,N,IH)
        CALL LINV2F(H,N,IH,TEMP,IDGT,WKAREA,IER)
```

```
      DO 5300 I = 1,N
      DO 5300 J = 1,N
         H(I,J) = TEMP(I,J)
5300  CONTINUE
      GO TO 40
19    CONTINUE
C** FORM M = (VTRANSPOSE.V)(-1)
      DO 20 I = 1,K
      DO 20 J = 1,K
         CALL INNERP(H,H,IH,IH,IH,IH,H(I,J),N,1,2,1,2,0,0,N+I,N+J)
         H(J,I) = H(I,J)
20    CONTINUE
      IF (K.EQ.1) H(1,1) = 1.0D0/H(1,1)
C**   IF (K.NE.1) CALL INVERT(H,K,IH)
      IF (K.NE.1) CALL LINV2F(H,K,IH,TEMP,IDGT,WKAREA,IER)
      DO 5400 I = 1,K
      DO 5400 J = 1,K
         H(I,J) = TEMP(I,J)
5400  CONTINUE
C** CALCULATE GENERALIZED INVERSE OF V, VPLUS = M.VTRANSPOSE
      DO 21 I = 1,K
         DO 22 J = 1,K
            X(N+J) = H(I,J)
22       CONTINUE
         DO 21 J = 1,N
           CALL INNERP(X,H,IX,1,IH,IH,H(I,J),K,N+1,N+2,N+1,N+2,0,1,1,J)
21    CONTINUE
C** SET UP DIAGONAL ELEMENTS OF THE PROJECTION MATRIX P = V.PLUS
      DO 23 I = 1,N
         CALL INNERP(H,H,IH,IH,IH,IH,X(N+I),K,1,2,N+1,N+2,0,1,I,I)
23    CONTINUE
      DO 24 I = 1,N
         LT(N+I) = 0
24    CONTINUE
      KV = K
C** ADD BOUND E(I) CORRESPONDING TO THE SMALLEST DIAG(P)
29    Z = 1.0D0
      DO 25 I = 1,N
         IF (LT(N+I).EQ.1) GO TO 25
         IF (X(N+I).GE.Z) GO TO 25
         Z = X(N+I)
         II = I
25    CONTINUE
      Y = 1.0D0
      IF (X(II)-BDL(II).GT.BDU(II)-X(II)) Y = -1.0D0
C** CALCULATE VECTORS VPLUS.E(I) AND U = E(I) - V.VPLUS.E(I)
      IF (Y.NE.1.0D0) GO TO 27
      DO 26 I = 1,K
         X(NN+I) = H(I,II)
26    CONTINUE
      GO TO 30
27    DO 28 I = 1,K
         X(NN+I) = -H(I,II)
28    CONTINUE
30    CONTINUE
      DO 31 I = 1,N
         IF(LT(N+I).EQ.1) GO TO 31
         CALL INNERP(H,X,IH,IH,IX,1,X(N3+I),KV,N+1,N+2,NN+1,NN+2,1,0,I,
     1   I)
         X(N3+I) = -X(N3+I)
```

```
31     CONTINUE
       DO 32 I = 1,N
          H(I,II) = 0.0D0
32     CONTINUE
       LT(N+II) = 1
       Z = 1.0D0 + X(N3+II)*Y
C** UPDATE VPLUS AND DIAG(P)
       DO 33 I = 1,N
          IF (LT(N+I).EQ.1) GO TO 33
          ALPHA = X(N3+I)/Z
          H(K+1,I) = ALPHA
          DO 34 J = 1,K
             H(J,I) = H(J,I) - X(NN+J)*ALPHA
34        CONTINUE
33     CONTINUE
       DO 35 I = 1,N
          IF (LT(N+I).EQ.1) GO TO 35
          X(N+I) = X(N+I) + X(N3+I)**20D0/Z
35     CONTINUE
       K = K + 1
       H(K,II) = Y
       IF(Y.NE.1.0D0) II = II + N
       LT(NN+II) = 0.0D0
       LT(K) = II
       IF (K.NE.N) GO TO 29
C** SET UP RHS OF CONSTRAINTS IN BASIS
40     DO 41 I = 1,N
          LI = LT(I)
          IF (LI.GT.N) GO TO 42
          X(N+I) = BDL(LI)
          GO TO 41
42        IF (LI.GT.NN) GO TO 43
          X(N+I) = -BDU(LI-N)
          GO TO 41
43        X(N+I) = D(LI-NN)
41     CONTINUE
C** CALCULATE POSITION OF VERTEX
       DO 44 I = 1,N
          CALL INNERP(H,X,IH,IH,IX,1,X(I),N,1,2,N+1,N+2,0,0,I,1)
44     CONTINUE
C** CALCULATE THE CONSTRAINT RESIDUALS, THE NUMBER OF VIOLATED
C** CONSTRAINTS, AND THE SUM OF THEIR NORMALS
50     KV = 0
       DO 51 I = 1,N
          X(N+I) = 0.0D0
51     CONTINUE
       DO 52 I = 1,M
          IF (LT(NN+I).LE.0) GO TO 52
          IF (I.GT.N) GO TO 53
          Z = X(I) - BDL(I)
          GO TO 55
53        IF (I.GT.NN) GO TO 54
          Z = BDU(I-N) - X(I-N)
          GO TO 55
54        J = I - NN
          CALL INNERP(C,X,IC,IC,IX,1,Z,N,1,2,1,2,0,0,J,1)
          Z =  Z - D(J)
55        X(NN+I) = Z
          IF (Z.GE.0.0D0) GO TO 52
          KV = KV + 1
```

```
          LT(NN+I) = 2
          IF (I.GT.N) GO TO 56
          X(N+I) = X(N+I) + 1.0D0
          GO TO 52
56        IF (I.GT.NN) GO TO 57
          X(I) = X(I) - 1.0D0
          GO TO 52
57        DO 58 II = 1,N
58        X(N+II) = X(N+II) + C(II,J)
52      CONTINUE
        IF (KV.NE.0) GO TO 63
        RETURN
C** POSSIBLE DIRECTIONS OF SEARCH OBTAINABLE BY REMOVING A
C** CONSTRAINT ARE ROWS OF H, CALCULATE THE OPTIMUM DIRECTION
63      Z = 0.0D0
        DO 64 I = 1,N
          IF (LT(NN+LT(I)).EQ.-1) GO TO 64
          CALL INNERP(H,X,IH,IH,IX,1,Y,N,1,2,N+1,N+2,1,0,I,1)
          IF (Y.LE.Z) GO TO 64
          Z = Y
          II = I
64      CONTINUE
        IF (Z.GT.0.0D0) GO TO 70
        WRITE(IOUT,1000)
1000    FORMAT('0NO FEASIBLE POINT')
        K = 0
        RETURN
C** SEARCH FOR THE NEAREST OF THE FURTHEST VIOLATED CONSTRAINT
C** AND THE NEAREST NONVIOLATED NONBASIC CONSTRAINT
70      ALPHA = 1.0D75
        BETA = 0.0D0
        DO 71 I = 1,N
          X(N+I) = H(II,I)
71      CONTINUE
        DO 72 I = 1,M
          IF (LT(NN+I).LE.0) GO TO 72
          IF (I.GT.N) GO TO 73
          Z = -X(N+I)
          GO TO 75
73        IF (I.GT.NN) GO TO 74
          Z = X(I)
          GO TO 75
74        JJ = I - NN
          CALL INNERP(X,C,IX,1,IC,IC,Z,N,N+1,N+2,1,2,0,0,1,JJ)
          Z = -Z
75        IF (LT(NN+I).EQ.2) GO TO 76
          IF (Z.LE.0.0D0) GO TO 72
          Z = X(NN+I)/Z
          IF (Z.GE.ALPHA) GO TO 72
          ALPHA = Z
          IAL = I
          GO TO 72
76        LT(NN+I) = 1
          IF (Z.GE.0.0D0) GO TO 72
          Z = X(NN+I)/Z
          IF (Z.LE.BETA) GO TO 72
          BETA = Z
          IB = I
72      CONTINUE
        IF (ALPHA.GT.BETA) GO TO 80
```

```
      IB = IAL
      BETA = ALPHA
C** EXCHANGE WITH THE CONSTRAINT BEING REMOVED FROM THE BASIS,
C** USING SIMPLEX FORMULA FOR NEW H
80    LT(NN+LT(II)) = 1
      LT(NN+IB) = 0
      LT(II) = IB
      IF (IB.GT.N) GO TO 82
      DO 81 I = 1,N
         X(NN+I) = H(I,IB)
81    CONTINUE
      GO TO 90
82    IB = IB - N
      IF (IB.GT.N) GO TO 84
      DO 83 I = 1,N
         X(NN+I) = -H(I,IB)
83    CONTINUE
      GO TO 90
84    IB = IB - N
      DO 85 I = 1,N
         X(N3+I) = C(I,IB)
85    CONTINUE
      DO 86 I = 1,N
         CALL INNERP(H,X,IH,IH,IX,1,X(NN+I),N,1,2,N3+1,N3+2,1,0,I,1)
86    CONTINUE
90    Z = 1.0D0/X(NN+II)
      DO 91 I = 1,N
         X(I) = X(I) + BETA*X(N+I)
         IF (I.NE.II) GO TO 92
         DO 93 J = 1,N
            H(I,J) = H(I,J)*Z
93       CONTINUE
         GO TO 91
92       ZZ = Z*X(NN+I)
         DO 94 J = 1,N
            H(I,J) = H(I,J) - ZZ*X(N+J)
94       CONTINUE
91    CONTINUE
      GO TO 50
      END
//
```

APPENDIX C

A SAMPLE OF THE INPUT FOR THE MINOS PACKAGE AND LISTING OF THE GENERATOR OF SUCH A SAMPLE

```
//U10832A   JOB (10832,269-34-0589),'F. M. KHALILI',TIME=(,5),
// CLASS=2,MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=*
/*PASSWORD ?
/*JOBPARM ROOM=F,FORMS=9031
// EXEC FORTVCLG,REGION.GO=1500K
//FORT.SYSIN DD *
C*******************************************************C
C*                                                     *C
C*  THIS PROGRAM CREATS THE TWO FILES REQUIRED BY MINOS. *C
C*  THE TWO FILES ARE CALLED SPECS AND MPS.            *C
C*                                                     *C
C*******************************************************C
C*                                                     *C
C*  AUTHOR : FOUAD M. KHALILI                          *C
C*  DATE   : NOV. 20,1987                              *C
C*                                                     *C
C*******************************************************C
       PARAMETER(N=50)
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION
     1 C(N),Q(N,N),A(N,N),RES1(N),RES2(N),ATRANS(N,N),
     2 B(N),X(N),U(N)
         OPEN(12,STATUS='OLD',ACCESS='SEQUENTIAL')
       IN = 5
       IOUT = 6
       TYPE = 0.0D0
       SEED = 50.0D0
       NOFROW = 4
       NOFCOL = 4
       NOACTV = 2
       NOZERO = 0
C** GENERATE X AND U VECTORS
       DO 100 I = 1,NOFCOL
          CALL GENRTE(SEED,RANDOM)
          X(I) = RANDOM
100    CONTINUE
       DO 110 I = 1,NOFROW
          CALL GENRTE(SEED,RANDOM)
          U(I) = RANDOM
110    CONTINUE
       DO 120 I = 1,NOFROW-NOACTV
          U(I) = 0.0D0
120    CONTINUE
C** GENERATE MATRIX A (OR CTRANS IN FLETCHER'S PAPER)
       DO 200 I = 1,NOFROW
       DO 200 J = 1,NOFCOL
          CALL GENRTE(SEED,RANDOM)
          IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
          A(I,J) = RANDOM
200    CONTINUE
C** GENERATE MATRIX Q (OR A IN FLETCHER'S PAPER)
       DO 300 I = 1,NOFCOL
       DO 300 J = 1,NOFCOL
          IF (I.GT.J) GO TO 300
          CALL GENRTE(SEED,RANDOM)
          IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
          ATRANS(I,J) = RANDOM
300    CONTINUE
       DO 1000 I = 1,NOFCOL
       DO 1000 J = 1,NOFCOL
```

```
          IF (I.LE.J) GO TO 1000
          ATRANS(I,J) = ATRANS(J,I)
1000   CONTINUE
C** TYPE = 0.=> Q IS INDEFINITE
C** TYPE = 1.=> Q IS POSITIVE DEFINITE
       IF (TYPE.EQ.0.) GO TO 10
       CALL MULT(ATRANS,ATRANS,NOFCOL,NOFCOL,NOFCOL,Q,N,N,N,N)
       DO 1200 I = 1,NOFCOL
       DO 1200 J = 1,NOFCOL
          IF (I.EQ.J)Q(I,J) = Q(I,J) + 1.0D0
1200   CONTINUE
       GO TO 40
10     DO 800 I = 1,NOFCOL
       DO 800 J = 1,NOFCOL
          Q(I,J) = ATRANS(I,J)
800    CONTINUE
40     DO 810 I = 1,NOZERO
       DO 810 J = NOFCOL-NOZERO+1,NOFCOL
          Q(I,J) = 0.0D0
810    CONTINUE
       DO 860 I = NOFCOL-NOZERO+1,NOFCOL
       DO 860 J = 1,NOZERO
          Q(I,J) = 0.0D0
860    CONTINUE
C** COMPUTE VECTOR C (OR B IN FLETCHER'S PAPER)
       DO 700 I = 1,NOFCOL
       DO 700 J = 1,NOFROW
          ATRANS(I,J) = A(J,I)
700    CONTINUE
       CALL MULT(ATRANS,U,NOFCOL,NOFROW,1,RES1,N,N,N,1)
       CALL MULT(Q,X,NOFCOL,NOFCOL,1,RES2,N,N,N,1)
       DO 400 I = 1,NOFCOL
          C(I) = RES1(I) - 2.0D0*RES2(I)
400    CONTINUE
C** COMPUTE VECTOR B ( OR D IN FLETCHER'S PAPER)
       CALL MULT(A,X,NOFROW,NOFCOL,1,B,N,N,N,1)
       DO 900 I = 1,NOFCOL
          IF (X(I).GT.0.0D0) GO TO 900
          CALL GENRTE(SEED,RANDOM)
          C(I) = C(I) + RANDOM
900    CONTINUE
       DO 910 I = 1,NOFROW
          IF (U(I).GT.0.0D0) GO TO 910
          CALL GENRTE(SEED,RANDOM)
          B(I) = B(I) - RANDOM
910    CONTINUE
       DO 440 I = 1,NOFCOL
       DO 440 J = 1,NOFCOL
          Q(I,J) = 2.0D0*Q(I,J)
440    CONTINUE
C**  FORM THE SPECS FILE
       IOUT = 12
       WRITE(IOUT,510)
510    FORMAT(2X,'BEGIN QP')
       WRITE(IOUT,520)NOFCOL
520    FORMAT(5X,'NONLINEAR VARIABLES',5X,I3)
       WRITE(IOUT,530)NOFCOL+1
530    FORMAT(5X,'SUPERBASICS LIMIT',7X,I3)
       WRITE(IOUT,540)
540    FORMAT(5X,'SUMMARY FILE                9')
```

```
      WRITE(IOUT,550)
550   FORMAT(5X,'SUMMARY FREQUENCY          1')
      II = 3*NOFROW + 10*NOFCOL
      WRITE(IOUT,560)II
560   FORMAT(5X,'ITERATIONS LIMIT',7X,I4)
      WRITE(IOUT,570)
570   FORMAT(2X,'END QP')
C** FORM THE MPS FILE
      WRITE(IOUT,580)
580   FORMAT('NAME          QP')
      WRITE(IOUT,590)
590   FORMAT('ROWS')
      DO 2100 I = 1,NOFROW
      IF(I.LE.9) GO TO 2200
      WRITE(IOUT,610)I
610   FORMAT(1X,'G',2X,'ROW',I2)
      GO TO 2100
2200  WRITE(IOUT,620)I
620   FORMAT(1X,'G',2X,'ROW',I1)
2100  CONTINUE
      WRITE(IOUT,630)
630   FORMAT(1X,'N  C')
      WRITE(IOUT,640)
640   FORMAT('COLUMNS')
      DO 2300 I = 1,NOFCOL
      DO 2400 J = 1,NOFROW
      IF(I.LE.9) GO TO 2500
      IF(J.LE.9) GO TO 2600
      WRITE(IOUT,650)I,J,A(J,I)
650   FORMAT(4X,'X',I2,7X,'ROW',I2,5X,D12.6)
      GO TO 2400
2600  WRITE(IOUT,660)I,J,A(J,I)
660   FORMAT(4X,'X',I2,7X,'ROW',I1,6X,D12.6)
      GO TO 2400
2500  IF(J.LE.9) GO TO 2700
      WRITE(IOUT,670)I,J,A(J,I)
670   FORMAT(4X,'X',I1,8X,'ROW',I2,5X,D12.6)
      GO TO 2400
2700  WRITE(IOUT,680)I,J,A(J,I)
680   FORMAT(4X,'X',I1,8X,'ROW',I1,6X,D12.6)
2400  CONTINUE
      IF(I.LE.9) GO TO 2800
      WRITE(IOUT,690)I,C(I)
690   FORMAT(4X,'X',I2,7X,'C',9X,D12.6)
      GO TO 2300
2800  WRITE(IOUT,710)I,C(I)
710   FORMAT(4X,'X',I1,8X,'C',9X,D12.6)
2300  CONTINUE
      WRITE(IOUT,720)
720   FORMAT('RHS')
      DO 2900 I = 1,NOFROW
      IF(I.LE.9) GO TO 3000
      WRITE(IOUT,730)I,B(I)
730   FORMAT(4X,'B',9X,'ROW',I2,5X,D12.6)
      GO TO 2900
3000  WRITE(IOUT,740)I,B(I)
740   FORMAT(4X,'B',9X,'ROW',I1,6X,D12.6)
2900  CONTINUE
      WRITE(IOUT,750)
750   FORMAT('ENDATA')
```

```
      CLOSE(12)
      STOP
      END
C*
C*
C***********************************************************C
C*                                                         *C
C*  SUBROUTINE MULT : MULTIPLIES TWO MATRICES RLEFT AND RIGHT.*C
C*  ARGUMENTS :                                            *C
C*      RLEFT : THE FIRST MATRIX                           *C
C*      RIGHT : THE SECOND MATRIX                          *C
C*      LEFTR : ROW SIZE OF THE FIRST MATRIX               *C
C*      LEFTC : COLUMN SIZE OF THE FIRST MATRIX            *C
C*      IRIHTC: COLUMN SIZE OF THE SECOND MATRIX           *C
C*      ID1   : ROW DIMENSION OF THE FIRST MATRIX          *C
C*      ID2   : COLUMN DIMENSION OF THE FIRST MATRIX       *C
C*      ID3   : ROW DIMENSION OF THE SECOND MATRIX         *C
C*      ID4   : COLUMN DIMENSION OF THE SECOND MATRIX      *C
C*      RESULT: MULTIPLICATION RESULT                      *C
C* INPUT      :                                            *C
C*      RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,ID1,ID2,ID3,ID4     *C
C* OUTPUT     :                                            *C
C*      RESULT                                             *C
C*                                                         *C
C***********************************************************C
C*
C*
      SUBROUTINE MULT(RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,RESULT,ID1,ID2,
     1 ID3,ID4)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION RLEFT(ID1,ID2),RIGHT(ID3,ID4),RESULT(ID1,ID4)
      DO 100 I = 1,LEFTR
      DO 100 J = 1,IRIHTC
         RESULT(I,J) = 0.0D0
100   CONTINUE
      DO 200 I = 1,LEFTR
      DO 300 J = 1,IRIHTC
      DO 400 K = 1,LEFTC
      RESULT(I,J) = RESULT(I,J) + RLEFT(I,K)*RIGHT(K,J)
400   CONTINUE
300   CONTINUE
200   CONTINUE
      RETURN
      END
C*
C*
C***********************************************************C
C*                                                         *C
C*  SUBROUTINE GENRTE : GENERATES A REAL NUMBER RANDOMLY   *C
C*  ARGUMENTS :                                            *C
C*    SEED    :  THE SEED FOR THE GENERATOR                *C
C*    RANDOM  :  THE GENERATED NUMBER                      *C
C*  INPUT :                                                *C
C*    SEED                                                 *C
C*  OUTPUT:                                                *C
C*    SEED,RANDOM                                          *C
C*                                                         *C
C***********************************************************C
C*
C*
```

```
      SUBROUTINE GENRTE(SEED,RANDOM)
      IMPLICIT REAL*8(A-H,O-Z)
      X = 3373.0D0
      Y = 6925.0D0
      WORD = 32768.0D0
      TMAX = 24.0D0
      ONE = 1.0D0
      SEED = DMOD((X*SEED + Y),WORD)
      RANDOM = INT(TMAX*(SEED/WORD) + ONE)
      RETURN
      END
//GO.FT12F001 DD DSN=U10832A.INP12.DATA,DISP=(OLD),
// UNIT=STORAGE,SPACE=(TRK,(5,2)),DCB=(LRECL=80,
// BLKSIZE=7440,RECFM=FB)
```

```
C*********************************************************C
C*                                                        *C
C*  THIS PROGRAM CALLS MINOS. IT PROVIDES THE MATRIX OF   *C
C*  OF THE QUADRATIC TERMS. IT ALSO CALCULATES THE GRAD   *C
C*  OF THE OBJECTIVE FUNCTION.                            *C
C*                                                        *C
C*********************************************************C
C*                                                        *C
C*  AUTHOR : FOUAD M. KHALILI                             *C
C*  DATE   : NOV. 20, 1987                                *C
C*                                                        *C
C*********************************************************C
C*
C*
C*
        IMPLICIT REAL*8(A-H,O-Z)
        DOUBLE PRECISION Z(10000)
        DATA  NWCORE/10000/
        CALL MINOS1(Z,NWCORE)
        STOP
        END
C**
C**
C***********************************************************C
C* SUBROUTINE FUNOBJ : TO CALCULATE THE OBJECTIVE FUNCTION *C
C* OF THE PROBLEM.                                         *C
C* ARGUMENTS :                                             *C
C*      MODE,NPROB,NSTATE,Z ARE DEFINED BY MINOS           *C
C*      N : NUMBER OF NONLINEAR VARIABLES.                 *C
C*      X : THE NONLINEAR VARIABLES.                       *C
C*      G : THE GRADIENT VECTOR.                           *C
C*      F : THE OBJECTIVE FUNCTION                         *C
C*      NWCORE : THE WORKING SPACE.                        *C
C* INPUT :                                                 *C
C*      NWCORE                                             *C
C* OUTPUT                                                  *C
C*      G AND F                                            *C
C*                                                         *C
C***********************************************************C
C*
C*
        SUBROUTINE FUNOBJ(MODE,N,X,F,G,NSTATE,NPROB,Z,NWCORE)
        IMPLICIT REAL*8(A-H,O-Z)
        DOUBLE PRECISION X(N),G(N),Z(NWCORE)
        COMMON /QPCOMM/ Q(100,100)
C**
C** COMPUTATION OF   F = 1/2 X'QX,  G = QX
C**
        IF (NSTATE.EQ.1) CALL SETQ(50)
        F = 0.0D0
        DO 200 I = 1,N
           GRAD = 0.0D0
           DO 100 J = 1,N
              GRAD = GRAD + Q(I,J)*X(J)
100        CONTINUE
           F = F + X(I)*GRAD
           G(I) = GRAD
200     CONTINUE
C**
        F = 0.5D0*F
```

```
      ENTRY FUNCON
      ENTRY MATMOD
      RETURN
C** END OF FUNOBJ FOR QP
      END
C*************************************************************C
C*                                                           *C
C*  SUBROUTINE SETQ : FINDS Q, THE HESSIAN MATRIX.           *C
C*  INPUT :                                                  *C
C*    ID : DIMENSION OF Q                                    *C
C*  OUTPUT :                                                 *C
C*    MATRIX Q                                               *C
C*                                                           *C
C*************************************************************C
C*
C*
      SUBROUTINE SETQ(ID)
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON /QPCOMM/ Q(100,100)
     DIMENSION  B(100),C(100),RES1(100),RES2(100),
   1 A(100,100),ATRANS(100,100),X(100),U(100)
     N = 100
     TYPE = 0.0D0
     SEED = 50.0D0
     NOFROW = 4
     NOFCOL = 4
     NOACTV = 2
     NOZERO = 0
C** GENERATE X AND U VECTORS
     DO 100 I = 1,NOFCOL
        CALL GENRTE(SEED,RANDOM)
        X(I) = RANDOM
100  CONTINUE
     DO 110 I = 1,NOFROW
        CALL GENRTE(SEED,RANDOM)
        U(I) = RANDOM
110  CONTINUE
     NOI = NOFROW-NOACTV
     IF(NOI.LT.1) GO TO 830
     DO 120 I = 1,NOI
        U(I) = 0.0D0
120  CONTINUE
C** GENERATE MATRIX A (OR CTRANS IN FLETCHER'S PAPER)
830  DO 200 I = 1,NOFROW
     DO 200 J = 1,NOFCOL
        CALL GENRTE(SEED,RANDOM)
        IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
        A(I,J) = RANDOM
200  CONTINUE
C** GENERATE MATRIX Q (OR A IN FLETCHER'S PAPER)
     DO 300 I = 1,NOFCOL
     DO 300 J = 1,NOFCOL
     IF (I.GT.J) GO TO 300
        CALL GENRTE(SEED,RANDOM)
        IF (SEED.LT.16000.0D0) RANDOM = -RANDOM
        ATRANS(I,J) = RANDOM
300  CONTINUE
     DO 1000 I = 1,NOFCOL
     DO 1000 J = 1,NOFCOL
        IF (I.LE.J) GO TO 1000
```

```
          ATRANS(I,J) = ATRANS(J,I)
1000    CONTINUE
C** TYPE = 0.=> Q IS INDEFINITE
C** TYPE = 1.=> Q IS POSITIVE DEFINITE
       IF (TYPE.EQ.0.0D0) GO TO 10
       CALL MULT(ATRANS,ATRANS,NOFCOL,NOFCOL,NOFCOL,Q,N,N,N,N)
       DO 1200 I = 1,NOFCOL
       DO 1200 J = 1,NOFCOL
          IF (I.EQ.J)Q(I,J) = Q(I,J) + 1.0
1200    CONTINUE
       GO TO 40
10      DO 800 I = 1,NOFCOL
       DO 800 J = 1,NOFCOL
          Q(I,J) = ATRANS(I,J)
800     CONTINUE
40      IF(NOZERO.LT.1) GO TO 820
       NOPLUS = NOFCOL-NOZERO+1
       DO 810 I = NOPLUS,NOFCOL
       DO 810 J = 1,NOZERO
          Q(I,J) = 0.0D0
820     CONTINUE
       DO 860 I = 1,NOZERO
       DO 860 J = NOPLUS,NOFCOL
          Q(I,J) = 0.0D0
860     CONTINUE
C** COMPUTE VECTOR C (OR B IN FLETCHER'S PAPER)
820     DO 700 I = 1,NOFCOL
       DO 700 J = 1,NOFROW
700     ATRANS(I,J) = A(J,I)
       CALL MULT(ATRANS,U,NOFCOL,NOFROW,1,RES1,N,N,N,1)
       CALL MULT(Q,X,NOFCOL,NOFCOL,1,RES2,N,N,N,1)
       DO 400 I = 1,NOFCOL
          C(I) = RES1(I) - 2.0D0*RES2(I)
400     CONTINUE
C** COMPUTE VECTOR B ( OR D IN FLETCHER'S PAPER)
       CALL MULT(A,X,NOFROW,NOFCOL,1,B,N,N,N,1)
       DO 900 I = 1,NOFCOL
          IF (X(I).GT.0.0D0) GO TO 900
          CALL GENRTE(SEED,RANDOM)
          C(I) = C(I) + RANDOM
900     CONTINUE
       DO 910 I = 1,NOFROW
          IF (U(I).GT.0.0D0) GO TO 910
          CALL GENRTE(SEED,RANDOM)
          B(I) = B(I) - RANDOM
910     CONTINUE
       DO 500 I = 1,NOFCOL
       DO 500 J = 1,NOFCOL
          Q(I,J) = 2*Q(I,J)
500     CONTINUE
       RETURN
       END
C*
C*
C*************************************************************C
C*                                                           *C
C*  SUBROUTINE GENRTE : GENERATES A REAL NUMBER RANDOMLY     *C
C*  ARGUMENTS :                                              *C
C*    SEED     :  THE SEED FOR THE GENERATOR                 *C
C*    RANDOM   :  THE GENERATED NUMBER                       *C
```

```
C*   INPUT :                                                     *C
C*     SEED                                                      *C
C*   OUTPUT:                                                     *C
C*     SEED,RANDOM                                               *C
C*                                                               *C
C****************************************************************C
C*
C*
      SUBROUTINE GENRTE(SEED,RANDOM)
      IMPLICIT REAL*8(A-H,O-Z)
      X = 3373.0D0
      Y = 6925.0D0
      WORD = 32768.0D0
      TMAX =24.0D0
      ONE = 1.0D0
      SEED = DMOD((X*SEED + Y),WORD)
      RANDOM = TMAX*(SEED/WORD) + ONE
      I = RANDOM
      RANDOM = I
      RETURN
      END
C*
C*
C****************************************************************C
C*                                                               *C
C*   SUBROUTINE MULT : MULTIPLIES TWO MATRICES RLEFT AND RIGHT.*C
C*   ARGUMENTS :                                                 *C
C*       RLEFT : THE FIRST MATRIX                                *C
C*       RIGHT : THE SECOND MATRIX                               *C
C*       LEFTR : ROW SIZE OF THE FIRST MATRIX                    *C
C*       LEFTC : COLUMN SIZE OF THE FIRST MATRIX                 *C
C*       IRIHTC: COLUMN SIZE OF THE SECOND MATRIX                *C
C*       ID1    : ROW DIMENSION OF THE FIRST MATRIX              *C
C*       ID2    : COLUMN DIMENSION OF THE FIRST MATRIX           *C
C*       ID3    : ROW DIMENSION OF THE SECOND MATRIX             *C
C*       ID4    : COLUMN DIMENSION OF THE SECOND MATRIX          *C
C*       RESULT: MULTIPLICATION RESULT                           *C
C* INPUT       :                                                 *C
C*       RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,ID1,ID2,ID3,ID4          *C
C* OUTPUT      :                                                 *C
C*       RESULT                                                  *C
C*                                                               *C
C****************************************************************C
C*
C*
      SUBROUTINE MULT(RLEFT,RIGHT,LEFTR,LEFTC,IRIHTC,RESULT,ID1,ID2,
     1 ID3,ID4)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION RLEFT(ID1,ID2),RIGHT(ID3,ID4),RESULT(ID1,ID4)
      DO 100 I = 1,LEFTR
      DO 100 J = 1,IRIHTC
         RESULT(I,J) = 0.0D0
100   CONTINUE
      DO 200 I = 1,LEFTR
      DO 300 J = 1,IRIHTC
      DO 400 K = 1,LEFTC
      RESULT(I,J) = RESULT(I,J) + RLEFT(I,K)*RIGHT(K,J)
400   CONTINUE
300   CONTINUE
200   CONTINUE
```

```
RETURN
END
```

```
C***************************************************************C
C** THIS IS A SAMPLE OF THE TWO INPUT FILES SPECS AND MPS THAT **C
C** ARE REQUIRED AS AN INPUT FOR MINOS.                        **C
C***************************************************************C
  BEGIN QP
     NONLINEAR VARIABLES        4
     SUPERBASICS LIMIT          5
     SUMMARY FILE               9
     SUMMARY FREQUENCY          1
     ITERATIONS LIMIT          52
  END QP
NAME          QP
ROWS
 G  ROW1
 G  ROW2
 G  ROW3
 G  ROW4
 N  C
COLUMNS
    X1        ROW1      0.160000D+02
    X1        ROW2      -.300000D+01
    X1        ROW3      -.200000D+01
    X1        ROW4      0.160000D+02
    X1        C         -.392000D+03
    X2        ROW1      -.200000D+01
    X2        ROW2      -.500000D+01
    X2        ROW3      0.150000D+02
    X2        ROW4      0.130000D+02
    X2        C         0.640000D+02
    X3        ROW1      0.180000D+02
    X3        ROW2      -.120000D+02
    X3        ROW3      0.140000D+02
    X3        ROW4      0.220000D+02
    X3        C         0.284000D+03
    X4        ROW1      -.200000D+01
    X4        ROW2      -.600000D+01
    X4        ROW3      -.800000D+01
    X4        ROW4      -.900000D+01
    X4        C         -.365000D+03
RHS
    B         ROW1      0.341000D+03
    B         ROW2      -.245000D+03
    B         ROW3      0.170000D+03
    B         ROW4      0.415000D+03
ENDATA
```

APPENDIX D

ADDITIONAL REFERENCES ON THE

QUADRATIC PROGRAMMING PROBLEM

SELECTED REFERENCES

1. Abadie, J. and J. G. Carpentier. "Generalization of the Wolfe Reduced-gradient method to the case of nonlinear constraints," In Optimization, R. Fletcher (Ed), Academic Press, London, pp. 37-49, 1969.

2. Aubin, J. P. Explicit Methods of Optimization. Bordas Dunod Gauthier-Villars, 1984.

3. Bazaraa, M.S. and Shetty, C. M. Non-linear Programming Theory and Algorithms. John Wiley and Sons, N.Y., 1979.

4. Benveniste, R. "Evaluating computational efficiency: A stochastic approach," Math. Prog., 21, (1981), pp. 152-171.

5. Benveniste, R. "One way to solve the parametric quadratic programming problem," Math. Prog., 21,(1981), pp. 224-228.

6. Best, M. J. "Equivalence of some quadratic programming algorithms," Math. Prog., 30, (1984), pp. 711-87.

7. Best, M. J. and R. J. Caron. "A method to increase the computational efficiency of certain quadratic programming algorithms," Math. Prog., 25, (1983), pp. 354-358.

8. Best, M. J. and K. Ritter. "An effective algorithm for quadratic minimization problems," MRC Technical Summary Report, #1691, University of Wisconsin, 1976.

9. Biggs, M. C. "On the convergence of some constrained minimization algorithms based on recursive quadratic programming," J. Inst. Math Appl., 21, (1978), pp. 67-82.

10. Bergthaler, C. "Minimum risk problems and quadratic programming," Discussion paper No. 7115, Louvain, Center for Operations Research and Econometrics, June 1971.

11. Boot, J. C. G. "On Sensitivity analysis in convex quadratic programming problems," Operations Research, (1963), pp. 771-786.

12. Bunday, B. D. Basic optimization methods, E. Arnold, 1984.

13. Chandrasekaran, R. and A. Tamir. "Optimization problems with algebraic solutions: quadratic fractional programs and ratio games," Math. Prog., 30, (1984), pp. 326-339.

14. Chang, Y. Y. and R. W. Cottle. "Least index resolution of degeneracy in quadratic programming," Math. Prog., 18, (1980), pp. 127-137.

15. Conn, A. R. and J. W. Sinclair. "Quadratic programming via a non-differentiable penalty function," Department of Combinations and Optimization Research, University of Waterloo, Rep. Corr 75-15 (1975).

16. Cottle, R. W. "Symmetric dual quadratic programs," Quarterly of Applied Mathematics, 21, (1963), pp. 237-243.

17. Cottle, R. W., and G. B. Dantziz. "Complementary pivot theory," In Mathematics of the Decision Sciences (part 1), Dantziz and Veinott (Eds), Providence, R. I., American Math. Soc., 1968.

18. Cottle, R. W. and A. Djang. "Algorithmic equivalence in quadratic programming I:A least-distance programming program," Report No. 76-26, Dept. of Operations Research, Stanford, University, Oct. 1976. (or Journal of Opt. Theory and Applications, 28, (1979), pp. 275-301.)

19. Cottle, R. W. and J. S. Pang. "On the convergence of a block successive overrelaxation method for a class of linear complementarity problems," Math. Prog., Study 17, (1982), pp. 126-138.

20. Daniel, J. W. "Stability of the Sol'n of Definite Quadratic Programs," Math. Prog.,5,(1973), pp. 41-53.

21. Dantzig, G. B. and R. W. Cottle. "Positive (semi) definite programming," In Nonlinear Programming,Abadie (Ed), Amesterdam, North-Holland Publishing Company, 1967, pp. 57-73.

22. Demokan, N. and A. H. Land. "A paramatric quadratic program to solve a class of bicriteria decision problems," J. Appl. Res. Soc., 32, (1981), pp. 477-488.

23. Dennis, J. B. "A dual problem for a class of quadratic programs," MIT Research Note, No. 1, Nov. 1957.

24. Djang, A. "Algorithmic Equivalence in quadratic programming," Ph.D. Thesis, Stanford University, California, 1980.

25. Dorn, W. S. "Duality in quadratic programming," Quarterly of Applied Math. 18, (2) (1960), pp. 155-162.

26. Dorn, W. S. "Self-dual quadratic programs," Journal of SIAM, 9, (1961), pp. 51-54.

27. Eaves, B. C. "On quadratic programming," Management Science, 17,(1971), pp. 698-711.

28. Eaves, B. C. and R. M. Freund. "Optimal scaling of balls and polyhedra," Math. Prog., 23, (1983), pp. 138-147.

29. Evans, D. A. "A simple worked example of the maximum of a quadratic funtion subject to linear inequalities, Technical Discussion Paper, No. 1, Department of Farm Management and Agriculture Economics, Massey University College, New Zealand, 1963.

30. Evtushenko, Y. G. Numerical Optimization Techniques. Springer-Verlag, N.Y., 1985.

31. Fletcher, R. and M. P. Jackson. "Minimization of a quadratic function of many variables subject only to upper and lower bounds," J. Inst. Math. Applic., 14,(1974), pp. 159-174.

32. Garcia-Palomares, U. M. and A. Restuccia. "A global quadratic algorithm for solving a system of mixed equalities and inequalities," Math. Prog., 21, (1981), pp. 290-300.

33. Gill, P. E. and W. Murray. "Numerically stable methods for quadratic programming," Math. Prog., 14, (1978), pp. 349-372.

34. Gill, P. E., W. Murray and M. H. Wright. Practical Optimization. Academic Press, London, 1981.

35. Golub, G. H. and M. A. Saunders. "Linear least squares and quadratic programming," In Integer and Nonlinear Programming, Abadie (Ed), Amesterdam, North Holland Publishing Co., 1970.

36. Han, S. P. and O. L. Mangasarian. "A dual differentiable exact penalty function," Math. Prog., 25, (1983), pp. 293-306.

37. Hua, L. Optimization. Wiley, 1985.

38. Jefferson, T. R. and C. H. Scott. "Quadratic geometric programming with application to machining economics," Math. Prog., 31,(1985), pp. 137-152.

39. Jeter, M. W. Mathematical Programming. Dekker, 1986.

40. Kuester, J. L. and J. H. Mize. Optimization Techniques with FORTRAN. McGraw-Hill, New York, 1973.

41. Kuhn, H. W. and A. W. Tucker. "Nonlinear programming," In Second Berkely Symposium on Mathematical Statistics and Probability, J. Neyman (Ed), University of Calif. Press, 1951.

42. Land, A. H. and S. Powell. Fortran Codes for Mathematical Programming. John Wiley and Son, New York, 1973.

43. Lazimy, R. "Mixed-integer quadradric programming," 22, (1982), pp. 332-349.

44. Lazimy, R. "Improved algorithm for mixed-integer quadradic programs and a computational study," Math. Prog., 32, (1985), pp. 100-113.

45. Lemke, C. E. "A method of solution for quadratic programming," Management Science, 8, (1962), pp. 442-453.

46. Lemke, C. E. and J. T. Howson. "Equilibrium points of bi-matrix games," J. Soc. Indust. Appl. Math., 12, (1964),pp.413-423.

47. Lenard, M. L. and M. Minkoff. "Randomly generated test problems for positive definite quadratic programming," ACM Transactions on Mathematical Software, 10 (1), (1984), pp. 86-96.

48. Lotstedt, P. "Time-dependent contact problems in rigid body mechanics," Math. Prog., Study 17 (1982), pp. 103-110.

49. Majthay, A. "Optimality conditions for quadratic programming," Math. Prog., 1, (1971), pp. 359-365.

50. Mangasarian, O. L. "Normal solutions of linear programs," Math. Prog., Study 22, (1984), pp. 206-216.

51. Mangasarian, O. L. "Locally unique solutions of quadratic programs, linear and nonlinear complementarity problems," Math. Prog., 19, (1980), pp. 200-212.

52. Mangasarian, O. L. "Sparsity-preserving SOR algorithms for separable quadratic and linear programming," Computer and Ops. Res., 11, (1984), pp. 105-112.

53. Murray, W. "An algorithm for finding a local minimum of an indefinite quadratic program," NPL NAC Report, No. 1,(1971).

54. Murray, W. and M. H. Wright. "Computation of the search direction in constrained optimization algorithms," Math. Prog., Study 16, (1982), pp. 62-83.

55. Pang, Jong-shi. "An equivalence between two algorithms for quadratic programming," Math. Prog., 20, (1981), pp. 152-165.

56. Pierre, Donald. Optimization Theory and Applications. Vieweg and Sohn, Germany, 1984.

57. Powell, M. J. D. "On the quadratic programming algorithm of Goldfarb and Idnani," Math. Prog., Study 25, (1985), pp. 45-61.

58. Shapley, L. "A note on the Lemke-Howson algorithm," Math. Prog., Study 1, (1974), pp. 175-189.

59. Shoup, T. E. Optimization Methods. Prentice Hall, 1987.

60. Sung, Y. Y. and J. B. Rosen. "Global minimum test problem construction," Math. Prog., 24, (1982), pp. 353-355.

61. Tomlin, J. A. "Robust implementation of Lemke's method for the linear complentarity problem," Report SOL 76-24, Dept. of Operations Research, Stanford Univ., 1976.

62. Valiaho, H. "A unified approach to one-parametric general quadratic programming," Math. Prog., 33, (1985), pp. 318-338.

63. Van de panne, C. "Local optima of quadratic programming problems by means of the simplex method for quadratic programming," Discussion paper No. 7335, Heverlee (Belgium), Center for Operations Research and Econometrics.

64. Van de panne, C., and A. Whinston. "A parametric simplical formulation of Houthakker's capacity method," Econometrica, 33, (1965), pp. 354-380.

65. Van de panne, C., and A. Whinston. "Simplicial methods for quadratic programming," Naval Research Logistics Quarterly, 11, (1964), pp. 273-302.

66. Van de panne, C., and A. Whinston. "The symmetric formulation of the simplex method for quadratic programming," Econometrica, 37 (3), (1969), pp. 507-527.

67. Von hohenbalken, B. "A finite algorithm to maximize certain pseudo-concave functions on polytopes," Math. Prog., 9, (1975), pp. 189-206.

68. Von hohenbalken, B. "Simplical decomposition in nonlinear programming algorithms," Math. Prog., 13, (1977), pp. 49-68.

69. Werner, J. Optimization Theory with Applications. Vieweg and Sohn, Germany, 1984.

70. Whinston, A. "The bounded variable problem--an application of the dual method for quadratic programmin," Naval Res. Logistics Quarterly, 12, (1965), pp. 173-180.

71. Wismer, D. A. and R. Chattergy. Introduction to Nonlinear Optimization, A Problem Solving Approach. Elsevier North Holland, Inc., N.Y., 1979.

72. Zangwill, W. I. Nonlinear Programming: A Unified Approach. Prentice-Hall, N.J., 1969.

73. Zwart, P. B. "Nonlinear programming: Counterexample to two global optimization algorithms," Operations Research, 2, (1973), pp. 1260-1266.

2

VITA

Fouad Mustapha Khalili

Candidate for the Degree of

Master of Science

Thesis: A COMPARISON OF THE COMPUTATIONAL PERFORMANCE OF THREE QUADRATIC PROGRAMMING ALGORITHMS

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Sidon, Lebanon, May 5, 1960, the son of Mustapha Hasan Khalili and M. A. Bakri.

Education: Received Bachelor of Science Degree in Civil Engineering from Oklahoma State University in May, 1982; Received Master of Science Degree in Structural Engineering from Oklahoma State University in May, 1984; Completed requirements for the Master of Science Degree at Oklahoma State University in December, 1987.

Professional Experience: Programmer, Department of Agriculture Engineering, Oklahoma State University, January, 1986 to November, 1986. Grader, Oklahoma State University, August, 1987 to present. Member of Golden Key National Honor Society.