CUADRO: AN INTERACTIVE PACKAGE TO SOLVE THE QUADRATIC PROGRAMMING PROBLEM

by

Oscar Manrique Andrade

A thesis submitted to the School of Graduate Studies and Research of the University of Ottawa in partial fulfillment of the requirements of Master of Science in Systems Science

Ottawa, Ontario, 1982
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
I hereby declare that I am the sole author of this thesis.

I authorize University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Oscar Manrique Andrade

I further authorize University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Oscar Manrique Andrade
University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.
ABSTRACT

The quadratic programming problem (QPP) deals with the optimization of a quadratic objective function, subject to linear constraints. In general, these constraints can be either in equality or inequality form and the nonnegativity conditions on the variables can apply for all or part of them. However, this problem has to be standardized as a preliminary step before trying to get a solution.

Some methods of solution are briefly presented and their computational behaviour analyzed. The method of Beale is widely discussed in both theoretical and practical ways and is adopted as the procedure to find the optimum. An interactive software package written in FORTRAN is presented.
I would like to express sincere gratitude to my supervisor, Professor Louis G. Birta, for suggesting the topic of this thesis, for his constant advice, valuable suggestions and guidance. He always found time for productive discussions and encouragement to my work.
INTRODUCTION

Mathematical programming is the name given to the analysis of problems where the optimum of a function is to be found, and the variables are subject to inequality or equality constraints. The term "linear programming", corresponds to the case where the function to be optimized, i.e. the objective function, is linear and where the constraint set is the intersection of a finite number of half spaces. Nonlinear programming is, then, the case where the objective function is nonlinear.

An important case of nonlinear programming is the quadratic programming problem (QPP), where the objective function is a quadratic function and the constraints set is as in the linear case, a finite set of half spaces.

Research on nonlinear programming has gone almost parallel to the development of linear programming, but the nonlinear case has found some mathematical difficulties notably more important than have hindered its development. Thus, we still cannot speak today of a complete theory of nonlinear programming. In this development, the period of the last two decades has, thus far, appeared to be a time of appraisal and consolidation. Generally speaking, the major achievements in this time have been related to evaluating and improving the efficiency of existing solution methods.
The rates at which many of the most commonly used algorithms converge to optimal solutions have been subjected to both theoretical and empirical examination [8, 25, 29].

Recently, particular attention has been paid to "Large-scale Mathematical Programming", that is, to the difficulties encountered in attempting to solve the very large nonlinear problems that sometimes arise in Operations Research. In this field, important works have been done in the developing of methods based on solving a sequence of QPP as a route towards the solution of a general nonlinear programming problem.

This thesis begins with a brief presentation of some of the most commonly used algorithms for solving the QPP and analyses their computational behaviour. The main intent of this thesis, however, is to present an interactive software package for the QPP, and discuss the most important aspects and problems encountered during its development.
## CONTENTS

### ABSTRACT ........................................ iv

### ACKNOWLEDGEMENTS .................................. v

### INTRODUCTION ...................................... vi

### Chapter

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. THE QUADRATIC PROGRAMMING PROBLEM ............ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>HISTORICAL OUTLINE .......................... 1</td>
<td></td>
</tr>
<tr>
<td>STATEMENT OF THE PROBLEM ................. 2</td>
<td></td>
</tr>
<tr>
<td>SOME APPLICATIONS .......................... 4</td>
<td></td>
</tr>
<tr>
<td>II. MATHEMATICAL BACKGROUND ................... 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>GLOBAL EXTREMA ................................ 8</td>
<td></td>
</tr>
<tr>
<td>THE SIMPLEX METHOD ......................... 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>THE canonical form ....................... 10</td>
<td></td>
</tr>
<tr>
<td>Basic solutions ......................... 13</td>
<td></td>
</tr>
<tr>
<td>Computational criteria ............... 14</td>
<td></td>
</tr>
<tr>
<td>Geometrical interpretation ........ 16</td>
<td></td>
</tr>
<tr>
<td>THE KUHN-TUCKER CONDITIONS ........... 19</td>
<td></td>
</tr>
<tr>
<td>III. THE METHOD OF BEALE ..................... 22</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>THEORETICAL ASPECTS OF THE METHOD .......... 22</td>
<td></td>
</tr>
<tr>
<td>UPDATING THE TABLEAUX .................. 26</td>
<td></td>
</tr>
<tr>
<td>A NUMERICAL EXAMPLE .................. 28</td>
<td></td>
</tr>
<tr>
<td>THE ALGORITHM .......................... 33</td>
<td></td>
</tr>
<tr>
<td>IV. SOME METHODS FOR THE QPP AND THEIR COMPUTATIONAL PERFORMANCE .......... 36</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>THE ALGORITHM OF WOLFF .................... 37</td>
<td></td>
</tr>
<tr>
<td>THE METHOD OF FANTZIG ..................... 39</td>
<td></td>
</tr>
<tr>
<td>EXPERIMENTATION .......................... 40</td>
<td></td>
</tr>
<tr>
<td>RESULTS AND ANALYSIS .................. 42</td>
<td></td>
</tr>
<tr>
<td>COMPUTER TIME AND STORAGE ........ 45</td>
<td></td>
</tr>
<tr>
<td>V. THE INTERACTIVE PACKAGE .................. 49</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>STRUCTURE OF THE PACKAGE ................ 49</td>
<td></td>
</tr>
<tr>
<td>STORAGE ORGANIZATION .................. 51</td>
<td></td>
</tr>
<tr>
<td>NOTATION AND DEFINITIONS ........ 53</td>
<td></td>
</tr>
<tr>
<td>THE SUBPROGRAMS .......................... 55</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>The Main program ..................... 56</td>
<td></td>
</tr>
<tr>
<td>Subroutine DIALOG ................... 56</td>
<td></td>
</tr>
</tbody>
</table>

- viii -
LIST OF TABLES

Table | page
---|---
1. INITIAL TABLEAU FOR BFAST | 26
2. MEAN NUMBER OF ITERATIONS FOR DIFFERENT LEVELS | 42
3. TIME AND STORAGE REQUIREMENTS | 47
4. AN EXAMPLE OF TABLE OUTPUT | 70

LIST OF FIGURES

Figure | page
---|---
1. GEOMETRICAL INTERPRETATION OF EXAMPLE 1 | 17
2. GEOMETRICAL INTERPRETATION OF EXAMPLE 2 | 18
3. THE METHOD OF BFAST ON EXAMPLE | 30
4. PACKAGE INTERACTION | 50
5. THE PROGRAM AND THE SUBROUTINES | 63
Chapter I

THE QUADRATIC PROGRAMMING PROBLEM

1.1 HISTORICAL OUTLINE

The subject of mathematical programming has formally existed as a well defined subject area, since the late 1940's when George Dantzig developed the simplex method for linear programming. Later, in 1951 H.W.Kuhn and J.W.Tucker, in the Second Berkeley Symposium on Mathematical Statistics and Probability, published the paper "Nonlinear Programming" [18] which established necessary conditions for optimal solutions to nonlinear programming problems. Since that time, the history of Mathematical Programming can be divided, roughly speaking, into two parallel and occasionally convergent streams of development: one inspired by the Kuhn-Tucker optimality criterion and the other based on the principles of Dantzig's Simplex Method.

Initially, the development of Dantzig's Simplex Method was rather narrowly confined to the identification and study of various important subclasses of linear programming problems (Transportation, Network flows, Game theory, etc). Then in the late 1950's, a confluence of the Simplex
computational approach and the Kuhn-Tucker theory produced a number of different algorithms for solving quadratic programming problems. Basically, these methods rely on the fact that the first partial derivative of a quadratic function is linear: this is precisely what makes quadratic programs the most easily handled of all nonlinear problems.

Perhaps the most important period in the development of the solution methods of quadratic problems and, in general, of nonlinear ones was between 1959-1967. Some important works appeared presenting useful and efficient algorithms that can be applied to many nonlinear programs [1,11,30]. Some were applicable to problems having linear constraints only and still others were designed specifically for certain categories of the nonlinear programming problem... the quadratic programming problem is in this class.

1.2 STATEMENT OF THE PROBLEM

Generally speaking, the problem of optimizing a function of one or more variables is called a mathematical programming problem. In considering the nonlinear programming problem one always refers to think of a minimization problem. This implies no loss of generality
since if the objective is to maximize $F(x)$, this can be converted to an equivalent minimization problem by letting $f(x) = -F(x)$ and then minimizing $f(x)$.

Consider a convex quadratic function of $n$ variables which is to be minimized:

$$G(x) = \frac{1}{2}x'Ax + b'x + c$$

where:

- $x$ is an $n$-vector of the unknowns
- $A$ is a symmetric positive semidefinite matrix with $n$ rows and $n$ columns
- $b$ is an $n$-vector
- $c$ is a scalar

The $m$ constraints ($m \leq n$) associated with the problem, are

$$Px - q \leq 0$$

$$x > 0$$

where:

- $P$ is a matrix with $m$ rows and $n$ columns
- $q$ is an $m$-vector with strictly positive components

Thus the QPP is characterized by linear constraints, nonnegativity requirement on the variables and an objective function $G(x)$ which is a positive semidefinite quadratic form.

The reason why the matrix $A$ must be positive semidefinite arises from one of the most serious problems in nonlinear programming: the problem of local and global minima. If
is a positive semidefinite matrix, any local minimum is also a global minimum. When \( A \) is not positive semidefinite, then there is a possibility that several local minima exist.

The QPP is often written as

\[
\min \{ G(x) = (1/2)x'Ax + b'x + c \mid Px \leq q, \ x \geq 0 \} \quad (P1)
\]

Note that the constraints and nonnegativity conditions are the same as in the case of linear programming. Constraints which are not in this form must be first transformed to this standard form by using any of techniques of linear programming; i.e., the phase I-phase II procedure or the big M technique (by adding artificial variables) [7].

In the statement of the QPP there are no restrictions on the choice of the elements of \( P, q \) or \( b \); but the restriction that \( A \) be positive semidefinite is crucial. So far, there is no method for solving the QPP where \( A \) is an arbitrary symmetric matrix. As noted in the subsequent discussions, some methods even require that \( A \) be positive definite.

1.3 SOME APPLICATIONS

Although linear functions are the most widely used type in the formulation of mathematical optimization problems, quadratic functions are quite firmly established in second
place. This is due to the fact that a large number of the functional relationships occurring in the real world are either truly quadratic or may be so approximated.

For example, the area of a disk, cube or other regular figures is proportional to the square of their characteristic linear dimensions. The kinetic energy carried by a rocket or atomic particle is proportional to the square of its velocity, while the potential energy of a rigid standing wall or dam is a quadratic function of its height. The revenue of a monopolistic firm that sells $x_1$ units of some product at a unit price of $x_2$ is $x_1x_2$, which is also quadratic. In statistics, the variance of a given sample of observations is a quadratic function of the values that constitute the sample.

One very common example of a QPP arises when data are to be fitted to a mathematical model by the least squares method. Suppose an economist theorizes that the fraction $B_j$ of the American public's total consumption expenditure that is allocated to goods and services in the $j$th class ($j = 1, 2, \ldots, n$) is constant from year to year, regardless of the overall level of consumption (assume all types of goods and services have been partitioned into $n$ mutually exclusive classes). He has available as data a detailed breakdown of all consumption expenditures over an $m$-year period and wishes to use these statistics to form least squares estimates of
the unknown parameters $B_j$. Let $C_i$ be the total consumption expenditure and let $X_{ij}$ be the amount allocated or goods and services in the $j^{th}$ class ($j = 1, 2, \ldots, n$) during the $i^{th}$ year ($i = 1, 2, \ldots, m$)

$$C_i = \sum_{j=1}^{m} X_{ij}$$

$i = 1, 2, \ldots, n$

Then the problem can be formulated as

$$\text{minimize } H$$

where

$$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} (X_{ij} - B_j C_i)^2$$

$i = 1, 2, \ldots, m$

Subject to $\sum_{j=1}^{n} B_j = 1$ and $0 < B_j < 1$

Another quadratic program known as the portfolio problem arises in the planning of investments. Suppose we want to invest a total of $B$ in $n$ different stocks and bonds $X_j$ ($j = 1, 2, \ldots, n$). The annual return on a $1$-investment in the $j^{th}$ security is a random variable whose expected value and variance (based on historical data) are $F_j$ and $\sigma_j^2$; the covariance of the $i^{th}$ and $j^{th}$ returns is $S_{ij} = \gamma_{ij}$.

If one wants only to maximize the expected overall return, regardless of risk, it would only be necessary to invest in the security offering the greatest $F_j$. However, one might prefer to invest in such a way as to minimize the
overall variance of the portfolio subject to the condition of an expected total return of at least SP per year. If we define $S$ to be a symmetric $N$ by $r$ matrix with elements $c_{ij}$, the problem can be formulated as follows:

find the values of $x_1, x_2, \ldots, x_n$ that minimize $Z$ where

$$Z = x' S x$$

subject to

$$\sum_{j=1}^{n} R_j x_j \geq n$$

$$\sum_{j=1}^{n} x_j = B \quad \text{and} \quad x_j \geq 0$$

$j = 1, 2, \ldots, n$
A function $f(x)$ takes its absolute minimum at a point $x^*$ if $f(x) \geq f(x^*)$ for all $x$ over which the function $f(x)$ is defined. The absolute minimum is also called global minimum.

A function $f(x)$ has a relative minimum at some point $x^*$ if there exists an interval including $x^*$, no matter how small, such that for all $x$ in this interval, $f(x) \geq f(x^*)$. The relative minimum is also referred to as local minimum. Similarly, there exist global maxima and local maxima, whose definitions follow logically from the previous ones.

For a concave function, linear interpolation between any two points never overestimates the value of the function. Obviously, if $f(x)$ is concave, then $-f(x)$ is convex.

Let $f(x)$ be a convex function over a closed interval $a \leq x \leq b$, then any local minimum of $f(x)$ in this interval is also the global minimum of $f(x)$ over the interval. The global maximum of a convex function $f(x)$ over a closed
interval $a \leq x \leq b$ will be taken on $a^+$ either $x = a$ or $x = b$ or both. The two previous statements are often stated as theorems and their proofs can be found in the optimization literature [9,10]. We may well be interested in minima that occur when one or more of the variables are at their upper or lower bound. It is readily seen that there are $2^n$ possible combinations of the $a_i$ and $b_i$ which the $x_i$ may take. If it is to minimize $z = f(x)$ and there is no characteristic known of $f(x)$ (even if it is known that it is continuous and differentiable), it can only be hoped to find a local minimum. However, considerably more can be said if the function being maximized or minimized is concave or convex.

If $f(x)$ is a convex function, it has a unique minimum over a feasible convex region and the same minimum value is found for a convex subset of the feasible region. Furthermore, the minimum of a concave function over a convex region occurs at an extreme point of the region, or otherwise points which are not extreme points can be written as linear combinations of extremal points.

The problem (P1) is said to be a convex quadratic programming problem if $P$ is positive semidefinite, so that $G(x)$ is convex. If $A$ is negative semidefinite, so that $G(x)$ is concave, then problem (P1) is called a concave quadratic programming problem.
2.2 THE SIMPLEX METHOD

2.2.1 The canonical form

When dealing with linear programming problems, the simplex method provides the most efficient procedure for solution. To apply the simplex method, the problem is assumed in its canonical form

\[ \begin{align*}
\text{Max } z &= b'x \\
Px &= q \\
x &\geq 0
\end{align*} \quad (2) \]

with n unknowns and m constraints, then:

- \( b \) is an n-vector,
- \( P \) is an \( mxn \) matrix,
- \( q \) is an m-vector and
- \( x \) is the n-vector of the unknowns.

This formulation is clearly very restrictive; however linear programming applies equally to situations where the constraints are inequalities or a mixture of equalities and inequalities, where some of the variables can have negative values and where the objective function is to be maximized.

In general, the case may be

\[ \text{maximize } Z \]

where

\[ Z = d'y \]
Subject to
\[
\begin{align*}
Fy & \leq r \\
Gy & \geq s \\
Hy & = t \\
y_i & \geq 0 \quad i = 1, 2, \ldots, k \quad ; \quad k \leq n
\end{align*}
\]
which need to be transformed into canonical form.

If a constraint is a less-than condition, then an equation can be made out of this constraint by adding a nonnegative variable \(u\) to the left-hand side and writing:
\[
Fy + u = r
\]
u is called a slack variable and is an additional unknown that has to be determined. Similarly, if a constraint is a greater-than condition, it can be written as
\[
Gy - v = s
\]
where \(v\) again is a nonnegative variable (usually called a surplus variable) that constitutes an additional unknown of the problem. If some of the variables are not constrained to be nonnegative, then these variables can be expressed as
\[
y_j = y'_j - y''_j \\
y'_j \geq 0 \quad y''_j \geq 0
\]
j = 1, 2, \ldots, n-k
because any number can always be expressed as the difference of two nonnegative numbers. Finally, instead of maximizing \(Z\), \(-Z\) will be minimized.

The problem becomes
\[
\text{minimize } Z
\]
where

\[
Z = \begin{bmatrix}
\tilde{d} & 0 & 0 \\
0 & u & v
\end{bmatrix}
\]

subject to

\[
\begin{bmatrix}
F & I & 0 \\
G & 0 & -I \\
H & 0 & 0
\end{bmatrix}
\begin{bmatrix}
y \\
u \\
v
\end{bmatrix}
= 
\begin{bmatrix}
r \\
s \\
t
\end{bmatrix}
\]

by putting

\[
x = \begin{bmatrix}
y \\
u \\
v
\end{bmatrix},
\quad P = \begin{bmatrix}
F & I & 0 \\
G & 0 & -I \\
H & 0 & 0
\end{bmatrix},
\quad q = \begin{bmatrix}
r \\
s \\
t
\end{bmatrix},
\quad b = \begin{bmatrix}
-d \\
\emptyset
\end{bmatrix}
\]

problem (P2) is gotten, which is in canonical form.

It should be noted that the number of unknowns have been increased by \(u + v + (n-k)\). Thus any method that solves the canonical form will also handle more general situations involving mixtures of equalities and inequalities, partial nonnegativity constraints, and minimization or maximization. The canonical form of the problem will be referred to hereafter.
2.2.2 Basic solutions

It will be convenient to define some concepts that can be useful in discussing future topics.

1. Feasible solution: Any solution to $Px = q$, $x \geq 0$.

2. Basic solution: A solution of $Px = q$, obtained by setting $n-m$ variables equal to zero and solving the remaining $m$ variables, provided that the determinant of the coefficients of these $m$ variables does not vanish (so that the values of these $m$ variables are uniquely determined).

3. Basis: The collection of $m$ variables which are not equal to zero in the construction of a basic solution.

4. Basic feasible solution: A basic solution of $Px = q$ which also satisfies $x \geq 0$.

5. Optimal solution: A feasible solution which satisfies $b'x = \min$.

6. Optimal basic solution: A basic feasible solution which satisfies $b'x = \min$.

Before starting the simplex procedure, a basic feasible solution must be found. For this, the vector $q$ in the constraints $Px = q$ must be strictly positive. Whenever necessary, a new transformation has to be done to the system such that one decision variable be isolated in each constraint with a $+1$ coefficient, that $-1$ variable not
appear in any other constraint and that it have a zero coefficient in the objective function.

The variables added at this point are called artificial variables because they don’t belong to the system and must eventually be suppressed. They are introduced only because they constitute immediately a basic feasible solution. After creating the artificial variables, they have to vanish. For this, the infeasibility form w defined by

\[ x_{n+1} + x_{n+2} + \ldots + x_{n+m} = w \]

has to be driven to zero [7,13]. Only then have we a basic feasible solution to start the simplex procedure.

2.2.3 **Computational criteria**

1. **Optimality criterion**: Suppose that in a minimization problem every nonbasic variable has a nonnegative coefficient in the objective function of a canonical form. Then the basic feasible solution given by that canonical form minimizes the objective function over the feasible region.

2. **Unboundedness criterion**: Suppose that in a minimization problem some nonbasic variable has a negative coefficient in the objective function of a canonical form. If that variable has positive or zero coefficients in all constraints, then the objective
function is unbounded from below over the feasible region.

3. Improvement criterion: Suppose that in a minimization problem some nonbasic variable has a negative coefficient in the objective function of a canonical form. If that variable has a negative coefficient in some constraint, then a new basic feasible solution may be obtained by pivoting.

4. Ratio and pivoting criterion: When improving a given canonical form by introducing the variable $x_s$ into the basis, pivot in a constraint that gives the minimum ratio of the right-hand-side coefficient to the corresponding $x_s$ coefficient. Compute these ratios only for constraints that have a positive coefficient of $x_s$.

When introducing the variable $x_s$ into the basis, another basic variable must leave its place to $x_s$. This substitution (pivoting)\footnote{The commonly known procedure for pivoting is not treated in this paper. However, it can be found in any book dealing with linear programming [7,13].} is merely the familiar variable elimination technique from high-school algebra, known more formally as Gauss-Jordan elimination. Consequently, after pivoting, the form of the problem has been altered, but the modified equations still represent the original problem and have the same feasible solutions and the same objective value when
evaluated at any given feasible point.

2.2.4 Geometrical interpretation

Consider the following problem stated in its canonical form:

$$\text{minimize } Z$$

where

$$Z = 0x_1 + 0x_2 + 3x_3 + x_4 - 2\alpha$$

subject to

$$x_1 - 3x_3 + 3x_4 = \xi \quad (1)$$
$$x_2 - 8x_3 + 4x_4 = 4 \quad (2)$$
$$x_j \geq 0 \quad (j = 1, 2, 3, 4)$$

Figure 1 illustrates the situation. For any value of $z$, say $z = -17$, the objective function is represented by a straight line. As $z$ decreases from $-20$, the line corresponding to the objective function moves parallel to itself across the feasible region. At $z = -20$ it meets the feasible region only at the point $x_3 = x_3 = 0$ and, for $z < -20$ it no longer touches the feasible region. Consequently, $z = -20$ is optimal.

If we change our objective function to $z = -3x_3 + x_4 - 20$ (which is shown in figure 2) decreasing $x_3$, while holding $x_4 = 0$, corresponds to moving outwards from the origin along the $x_3$-axis. As we move along the axis, we never meet either constraint (1) or (2). Also, as we move along the $x_3$-axis, the value of the objective function is increasing.
Figure 1: GEOMETRICAL INTERPRETATION OF EXAMPLE 1

to $+\infty$. Hence the objective function is unbounded from
Figure 2: GEOMETRICAL INTERPRETATION OF EXAMPLE 2

above.
On the other hand, if our objective function is changed to \( z = 3x_3 - x_4 - 20 \), which is also shown in figure 2, decreasing \( x_4 \) corresponds to moving from the origin along the \( x_4 \)-axis. In this case, however, we encounter constraint 2 at \( x_4 = 1 \) and constraint 1 at \( x_4 = 2 \). Consequently, to maintain feasibility in accordance with the ratio criterion, we move to the intersection of the \( x_4 \)-axis and constraint 2, which is the optimal solution.

2.3 THE KUHN-TUCKER CONDITIONS

In attempting to develop algorithms for solving nonlinear programming problems, it is useful to have some information concerning the characteristics of an optimal solution. Kuhn and Tucker [18] showed that if a vector \( x^* \) minimizes \( f(x) \), subject to constraints \( h_i(x) \geq 0 \), then,

\[
\frac{\partial f}{\partial x_j} + \sum_{i=1}^{m} t_i \frac{\partial h_i}{\partial x_j} = 0
\]

\( h_i(x^*) \geq 0 \) (K^I)

\( t_i h_i(x^*) = 0 \)

\( t_i \geq 0 \)

for all \( i = 1, 2, \ldots, m \)

\( j = 1, 2, \ldots, n \)
where

t_i is the Lagrange multiplier associated with the i-th constraint.

These are the well-known Kuhn-Tucker conditions.

However, in the case of the QPF, the Kuhn-Tucker conditions may have a special form. The function \( f(x) \) to be minimized is in this case

\[
G(x) = (1/2)x'Px + b'x + c
\]

and the constraints \( h(x) \geq 0 \) are of two kinds
The side constraints

\[
h^1 = q - Px \geq 0
\]

and the nonnegativity constraints

\[
h^2 = x \geq 0.
\]

Then,

\[
\frac{\partial G(x)}{\partial x} = Ax + b
\]

\[
\frac{\partial h^1(x)}{\partial x} = \frac{\partial}{\partial x} (q - Px) = -\sum_{i=1}^{n} v_i P_i^j = -P'u
\]

where \( u_i \) is the Lagrange multiplier associated with the i-th side constraint. Clearly, the m-vector \( u \) is the Lagrange multiplier associated with the m-side constraints \( h^1(x) \).

Similarly, for the nonnegativity constraints we have

\[
\frac{\partial h^2(x)}{\partial x} = \sum_{j=1}^{n} v_j e_j = v
\]

where \( e_j \) is the j-th column of the identity matrix \( I \) and \( v_j \) is the Lagrange multiplier associated with nonnegativity constraint \( x_j \).

At the minimum point, the conditions are:
\[ Ax + b = -P'u + v \]
\[ u \geq 0 \]
\[ v \geq 0 \]
\[ x \geq 0 \]

Moreover, \( t_i h_i(x^*) = 0 \) becomes

\[ u_i h_i(x^*) = 0 \text{ or } u_i (g_j - \sum_{i=1}^{n} P_{ij} x_i) = 0 \]

\( j = 1, 2, \ldots, n \)

If \( y_i \) represents the nonnegative slack variable associated with the \( i^{th} \) constraint

\[ y_i = g_i - \sum_{j=1}^{n} (P_{ij} x_j) \]

then, from above

\[ u'y = 0 \]

Similarly, for the nonnegativity constraints \( v'x = 0 \).

Then, \( t_i h_i(x^*) = 0 \) becomes \( v'x + u'y = 0 \) which is true. Since \( v, u, x, y \geq 0 \), the only way for it to hold is that \( v'x = 0 \) and \( u'y = 0 \).

Then, the Kuhn-Tucker conditions for the OPP can be written as

\[ Ax + b + P'u = v \]
\[ v'x + u'y = 0 \]
\[ q = Rx = y \]
\[ x, y, u, v \geq 0 \]
3.1 THEORETICAL ASPECTS OF THE METHOD

The method of Beale [1] was published as an application of the simplex method to quadratic programming and it reduces to the simplex method when the objective function is linear.

Consider the problem of minimizing the quadratic objective function $G(\bar{x})$ ($\bar{x}$ an $n$-vector), subject to the constraints

$$\bar{P}\bar{x} \leq q$$

$$\bar{x} \geq 0$$

where $\bar{P}$ is an $mxn$ matrix and $q$ is an $m$-vector.

We can assume without loss of generality that a basic feasible solution can be obtained from this set of constraints (see section 2.3.1). Call $y$ the vector of the variables that give a basic feasible solution at the starting point, then the constraints become:

$$\bar{P}\bar{x} + y = q$$

$$\bar{x} \geq 0 , \ y > 0$$

or

$$\begin{bmatrix} \bar{P} \\
-\bar{P} \end{bmatrix} \begin{bmatrix} \bar{x} \\
y \end{bmatrix} = \begin{bmatrix} q \\
0 \end{bmatrix}$$
Let

\[
[x] = \begin{bmatrix} \bar{x} \\ y \end{bmatrix} \quad \text{and} \quad [P] = \begin{bmatrix} \bar{P} & I \end{bmatrix}
\]

Then \(x\) is an \((m+n)\)-vector and \(P\) an \(m \times (n+m)\). As a result the constraints can be written

\[
Px = q
\]

\[
x \geq 0
\]

It is assumed that the basic variables are the last \(x_{n+k}\) \((k=1,2,\ldots,m)\) and the first \(x\) \((j=1,2,\ldots,n)\) are nonbasic. Define the vector \(z = (z_0, z_1, \ldots, z_m)\)

\[
z = \begin{bmatrix} 1 \\ x \end{bmatrix}
\]

Then the objective function can be expressed in terms of \(z\) as

\[
G(z) = z' \Gamma' z
\]

where

\[
\Gamma = \begin{bmatrix} c & b'/2 \\ - & - & - & - \\ b/2 & (1/2)' \end{bmatrix}
\]

Note now that for \(1 \leq k \leq m\) the \((k+1)\)th entry of \(\partial G/\partial z_k\) is

\[
b_k + \sum_{j=1}^{n} a_{kj} z_j
\]

The constraints can be used to express the basic variables in terms of the nonbasic ones

\[
x_i = p_{ic} = \sum_{j=1}^{n} p_{ij} z_j
\]
where \( p_{io} = q \), at the starting point where a basic feasible solution is immediately available. At any trial solution of the problem, the basic variable \( x_i = p_{io} \) and the nonbasic variables \( z \) are all zero. Also, at any trial point, the equations of the constraints can be used to express the objective function \( G \) in terms of the nonbasic variables only.

If \( \partial G / \partial z_s \geq 0 \), then an increase in \( z_s \) will not reduce \( G \); but if \( \partial G / \partial z < 0 \), then a small increase in \( z_s \) will reduce \( G \). In the general case, when \( G \) is an arbitrary continuously differentiable function, the process of increasing \( z_s \) (when \( \partial G / \partial z_s < 0 \)), must terminate when either one of the following conditions holds:

1. A basic variable, say \( x_t \), has become zero and is about to become negative.

2. \( \partial G / \partial z_s \) vanishes and is about to become positive.

In case 1. (which is the only possibility in the case of linear programming), the remedy is to change the basis by making \( x_t \) nonbasic in place of \( z_s \). Case 2. can give rise to substantial difficulty if \( G \) is an arbitrary function; but if \( G \) is a quadratic function, then \( \partial G / \partial z_s \) is a linear function of the nonbasic variables. In this situation, case 2. results in the creation of a new constraint.

If \( \partial G / \partial z_s \) becomes positive as \( z_s \) is increased, (with all other nonbasic variables equal to zero) then a new variable is defined as follows:
\[ u = h_s + \sum_{j=1}^{n} a_s^j z_j \]

Now \( u \) is made the new nonbasic variable, and throughout the constraints and the expression for \( G \), the above equation is used to substitute for \( z_s \) in terms of \( u \) and the other nonbasic variables. Note that if \( z_s \) is one of the original \( x \)-variables (not one of the introduced \( u \)-variables), then there will be additional basic variable after this step.

One important feature of the new \( u \) variable is that it is not restricted to nonnegative values. It is therefore called a 'free' variable as opposed to the original \( x \)-variables which are called restricted variables. Since \( u \) is not sign restricted, if \( \delta G/\delta u > 0 \) then \( G \) can be reduced by making \( u \) negative.

It is convenient to have a set of nonbasic variables that all vanish at the trial solution, since the values of the basic variables are then simply given as the constant terms in the tableau. The new variable is chosen so that it will not be profitable to change its value when we change the values of any other nonbasic variable. It has been shown [19,3] that this method terminates in a finite number of steps.
3.2 **UPDATING THE TABLEAUX**

When passing from one iteration to the next, the expressions for the constraints are handled exactly as in linear programming, (the constraints part is often called the simplex tableau). However, for the objective function, there is a different approach.

**TABLE 1**

**INITIAL TABLEAU FOR BEALE**

<table>
<thead>
<tr>
<th>Constraints tableau</th>
<th>variables</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>c</td>
<td>-p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objective function tableau</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
</tr>
<tr>
<td>b/2</td>
</tr>
<tr>
<td>z</td>
</tr>
<tr>
<td>b/2</td>
</tr>
<tr>
<td>1/2</td>
</tr>
</tbody>
</table>

Table 1 shows the initial tableau for the method of Beale. The constraints tableau shows the basic variables in terms of the nonbasic ones. The objective function tableau shows the symmetric matrix \( \Gamma \) of size \((n+1) \times (n+1)\) whose first row can be used to determine optimality in the trial point: a positive value in each one of the columns \(v\), where \(z_v\) is a restricted variable and a zero value in each
of the columns $t$, where $z_t$ is a free variable ("), denotes an optimal tableau.

The procedure suggested by Beale can be expressed algebraically by saying that, starting from the objective function that is in terms of the nonbasic variables $z$, the $z$-variable that is to become basic is replaced by the equivalent expression in terms of the basic variable $x$ (that is to become nonbasic) and the other nonbasic variables, giving rise to a new expression for $G$ in terms of the new nonbasic variables. This is not very convenient in a computer, since it involves operating on both the rows and columns of the matrix. It is worthwhile examining the algebraic expressions for the combined effects of these transformations.

If we denote the pivotal column by $s$ and if the expression for the new basic variable $z$ is, in terms of the new nonbasic variables

$$z_t = e_s + e_s z_s + \sum_{k \neq s} e_k z_k$$

where $z_s$ denotes the new nonbasic variable replacing $z_t$, then the new coefficients $g'_{kj}$ are given in terms of the old coefficients $g_{kj}$ and the $e_k$ as follows:

$$g'_{ss} = g_{ss} e_s^2$$

$$g'_{ks} = g'_{ek} = g_{ks} e_s + g_{ss} e_s e_k$$

$$g'_{kj} = g_{kj} + g_{ks} e_j + g_{sj} e_k + g_{ss} e_k e_j$$

where $k, j \neq q$

If $g^*$ is defined as
\[ g^* = (1/2) g \cdot e \]
\[ g^* = g + (1/2) g \cdot e \quad \text{for} \quad k \neq q \]
then, the above expressions can also be written
\[ g' = 2g^* e \]
\[ g' = q' = g^* e + q e \]
\[ g' = g + q^* e + q^* e \]

The optimum point is reached when the derivatives of the objective function with respect to each of the nonbasic variables are nonnegative and the derivative of the objective function with respect to the free variables introduced in the problem are all zero. This is so because at this point an admissible change in any of the restricted variables will increase the value of the objective function and it is impossible either to increase or to decrease any of the free variables. Hence the value of \( c \) cannot be minimized further.

3.3 A NUMERICAL EXAMPLE

To illustrate this method, an example given by Beale [2] that has the characteristic of highlighting the principal features of the algorithm, is presented.

The problem is
minimize \( G = -6x_1 + 2x_1^2 - 2x_1x_2 + 2x_2^2 \)

subject to \( x_1 + x_2 \leq 2 \quad x_1, x_2 \geq 0 \)

Introducing a slack variable \( x \), we have the constraint:
\( x_1 + x_2 + x_3 = 2 \quad x_1, x_2, x_3 \geq 0 \)

A feasible solution for starting the process is
\( x_1 = 0 \quad x_2 = 0 \)

This corresponds to point \( A \) in figure 3. At this point, the following expressions for the variable \( x_3 \) and the objective function hold:
\[
x_3 = 2 - x_1 - x_2
G = (-3x_1) + (-3 + 2x_1 - x_2)x_1 + (- x_1 + 2x_2)x_2
\]

and so at the trial point:
\( x_1 = 0 \quad x_2 = 0 \quad x_3 = 2 \quad G = c \)

Since \( \frac{\partial G}{\partial x_1} = -3 \), \( G \) can be decreased by increasing \( x_1 \) (\( x_2 \) held at its value of 0)
\( \frac{\partial G}{\partial x_1} = -3 + 2x_1 - x_2 \) vanishes for \( x_1 = 3/2 \)

This point still belongs to the feasible region since \( x_3 \) equals 0 only when \( x_1 = 2 \) and then \( \frac{\partial G}{\partial x_1} = -3 + 2x_1 = +1 \)

so, a free variable is introduced:
\[
u_1 = \frac{\partial G}{\partial x_1} = -3 + 2x_1 - x_2
\]

where the subscript 1 on \( u \) means that it is the first free variable introduced into the problem. This \( u_1 \) variable replaces \( x_1 \) as an independent variable that vanishes.
Figure 3: THE METHOD OF BEALE ON EXAMPLE

\[ u_1 \text{ is zero in all the points where the ellipse } G = \]

- 30 -
constant has a horizontal tangent. From above: $x_4 = 3/2 + (1/2)u_1 + (1/2)x_2$

Hence

$$s = 1 \quad e_0 = 3/2 \quad e_1 = 1/2 \quad e_2 = 1/2$$

$$q_{ss} = 1 \quad q_0^* = -3/2 \quad q_1^* = 1/2 \quad q_2^* = -1/2$$

Then the second trial point can be set and the tableau reads

$$x_4 = 3/2 + (1/2)u_1 + (1/2)x_2$$
$$x_3 = 1/2 - (1/2)u_1 - (3/2)x_2$$
$$G = (-9/2 + (3/2)x_2)1$$
$$+ (1/2)u_1$$
$$+ (-3/2 (3/2)x_2) x_2$$

At the second point B

$$u_1 = 0, \quad x_2 = 0, \quad x_1 = 3/2, \quad x_3 = 1/2, \quad G = -9/2$$

Since $\partial G/\partial x_2 = -3/2, x_2$ must be put into the basis ($u_1$ stays 0)

$$\partial G/\partial x_2 = -3/2 + (3/2)x_2 \quad \text{vanishes for } x_2 = 1$$

It is not possible to reach this value since $x_3$ vanishes before $x_2$, that is, for $x_2 = 1/3$.

Then we make $x_3$ nonbasic and we express the new basic variable $x_2$ and the previous one $x_4$, as functions of $x_3$ and $u_4$, the new nonbasic set.

This gives as a third point:

$$x_2 = 1/3 - (1/3)u_1 - (2/3)x_3$$
$$x_4 = 5/3 + (1/3)u_1 - (1/3)x_3$$

Hence,

$$s = 2 \quad e_0 = 1/3 \quad e_1 = -1/3 \quad e_2 = -2/3$$
\[ (1/2)q_{ss} = 3/4 \quad g^* = -5/4 \quad q^*_1 = -1/4 \quad q^*_2 = -1/2 \]

and the objective function at this point is:
\[ G = (-16/3 + (1/3)u_1 + (2/3)x_3) 1 + (1/3 + (2/3)r_1 + (1/3)x_3) u_1 + (2/3 + (1/3)u_1 + (2/3)x_3) x_3 \]

The trial point corresponds to point C:
\[ u_1 = 0, x_3 = 0, x_2 = 1/3, x_1 = 5/3, G = -16/3 \]

Since an x-variable has been made nonbasic, \( u_1 \) must become basic. The only way to decrease \( G \) at this point is by decreasing \( u_1 \), because \( \partial G/\partial u_1 > 0 \). This is possible until
\[ 5/3 + (1/3)u_1 - (1/3)x_3 = 0 \]

that is, for \( u_1 = -5 \). But \( \partial G/\partial u_1 = 1/3 + (2/3)(-5) = -3 \) vanishes first, so the next nonbasic variable is a free variable \( u_2 \) defined by
\[ u_2 = 1/3 + (2/3)u_1 + (1/3)x_3 \]
i.e.,
\[ u_1 = -1/2 + (3/2)u_2 - (1/2)x_3 \]

hence,
\[ s = 1 \quad e_o = -1/2 \quad e_1 = 3/2 \quad e_2 = -1/2 \]
\[ (1/2)q_{ss} = 1/3 \quad q^*_o = 1/6 \quad q^*_1 = 1/2 \quad q^*_2 = 1/6 \]

and a new tableau is available:
\[ u_1 = -\frac{1}{2} + \frac{3}{2}u_2 - \frac{1}{2}x_3 \]
\[ x_1 = \frac{3}{2} + \frac{1}{2}u_2 - \frac{1}{2}x_3 \]
\[ x_2 = \frac{1}{2} - \frac{1}{2}u_2 - \frac{1}{2}x_3 \]
\[ G = \left( -\frac{11}{2} + \frac{1}{2}x_3 \right) + (3/2)u_2 \]
\[ + (1/2) \]
\[ + (1/2)x_3 \]
\[ x_3 \]

Since the derivative of \( G \) with respect to the free variable \( u_2 \) is zero and the derivative with respect to the restricted variable \( x_3 \) is positive, the minimum point has been reached. The problem solution therefore is:

\[ x_1 = \frac{3}{2}, \quad x_2 = \frac{1}{2}, \quad x_3 = 0, \quad G = -\frac{11}{2} \]

This corresponds to point D in figure 3.

3.4 THE ALGORITHM

1. Optimality: Check the first row of the objective function tableau. If the tableau is optimum, stop.

2. Pivot column: Select in the first row of the tableau, among the \( u \)-column a non-zero element. If all the elements in the first row corresponding the \( u \)-columns are zero or there are no \( u \)-columns then choose among the \( x \) columns the most negative. Call
this element $g$. Then $r$ is the pivot column, which defines the variable $z$ (either $u$ or $x$) that becomes basic.

3. **Pivot row**: Divide the absolute value of the element $g$ by the element $a$ in the diagonal of the objective function tableau. If this element is positive, then divide the elements $p$ of the first column of the constraints tableau by the absolute value of the correspondent element $p$ in the pivot column, but only if this element has the same sign as the element $g$. Obtain the same ratio in all the rows (except the first one) of the objective function tableau having in the first column one element with the same sign as $g$, (take absolute value of $q$, $h \neq 0$). The row giving the minimum ratio is selected as pivot row. The element located in the intersection of the pivot column and the pivot row (either $g$ or $p$) is called the pivot element.

4. If the pivot element is $p$, that is, the pivot row belongs to the constraints tableau, replace the variable $z$ (in the top of the pivot column) by the variable $z$ (heading the pivot row).

5. If the pivot element is $q$, that is, the pivot row belongs to the objective function tableau, replace $z$
by a new variable $u$, that goes to the constraint's tableau.

6. Update the tableaux (constraints section and objective function section) and repeat from step 1.
Chapter IV
SOME METHODS FOR THE QPP AND THEIR COMPUTATIONAL PERFORMANCE

Several articles have appeared in the literature comparing the computational efficiency of three of the best known simplicial methods of quadratic programming [1, 11, 30], namely the methods of Wolfe, Beale and Dantzig. A large number of problems have been solved with coefficients which were random variates obtained from a pseudo-random number generator. In certain cases, problem parameters were varied to determine whether they have a significant effect on the rate of convergence of the algorithms.

The following results are taken from a study carried out by Braitsch [8], who followed an experimental approach to compare the algorithms of Wolfe, Beale and Dantzig. A fourth method is also treated which is a slight modification of Wolfe's algorithm. This modification have been introduced in an attempt to reduce the effects of some of the inefficiencies inherent in Wolfe's algorithm. The method of Beale was presented in the previous chapter. A brief discussion of the methods of Wolfe and Dantzig is presented here.
4.1 **The Algorithm of Wolfe**

Wolfe [30], working from an idea suggested by Makowitz [23], developed a method for the QPP that is based on solving the Kuhn-Tucker conditions. Computationally, it corresponds to a simple modification of the artificial variable technique for finding a first feasible solution to a linear programming problem [7].

For the problem (P1), \((3n + 2m)\) supplementary variables are added to the corresponding Kuhn-Tucker conditions (KT2):

\[ w_i, \quad i = 1, 2, \ldots, n \]
\[ z_j^1, z_j^2, \quad i = 1, 2, \ldots, (m+n) \]

Now note that the \(x\)-vector which satisfies the following conditions when \(w = z^1 = z^2 = 0\) and \(r = 1\), is the solution to the original problem:

\[
\begin{align*}
px + y + & w = q \\
Ax + v + & P'u + z^1 - z^2 + rb = 0 \\
v'x + u'y = & 0 \\
x, v, y, u, w, z^1, z^2 \geq & 0
\end{align*}
\]

The simplex method is performed with an additional rule: \(v'x + u'y = 0\) must hold throughout the procedure. This assures that at every point and for all \(i\), no \(v_i\) and \(x_i\) can have nonzero value (cannot be in the basis) at the same time.

Wolfe's method exists in two forms: The short form that finds a minimum only if \(b = 0\) or \(A\) is positive definite and
The long form that can be applied without restrictions. The short form requires two phases in the procedure and the long form adds one more phase to the results obtained with the short form. In the first phase, \( r \) is set to such that \( rb = 0 \) (since in the short form, this guarantees a solution) and \( w \) is minimized (in fact, driven to zero), holding \( r, u \) and \( v \) out of the basis.

The initial tableau is set:

\[
w = q - PX - y \\
z_1 = 0 - r'X + v' - P'u + z_2 - rb
\]

and the simplex method is applied until \( \sum_{i=1}^{n} w_i = 0 \).

The second phase minimizes \( z_1 \) and \( z_2 \). At this point, there is no need to differentiate between \( z_1 \) and \( z_2 \) since both must vanish. \( r \) must not enter the basis in this phase. Again, the simplex method is used until the sum of all entries in \( z_1 \) and \( z_2 \) is zero. This concludes the short form.

In the third phase, (in the long form) \( r \) is set to 1, and the objective is to obtain two successive tableaux in which the \( r \) values are slightly below and above 1. With each of these \( r \)'s there is associated a solution vector and a linear interpolation is used to obtain the solution vector corresponding to \( r = 1 \), which is the problem solution.
4.2 THE METHOD OF DANTZIG

G.B. Dantzig [11] took the algorithm of Wolfe and developed a slightly more compact version of the method that works somewhat faster if $A$ is either positive definite or semidefinite. The significant shortcoming is that it may not terminate at all if $A$ is indefinite.

It uses the Kuhn-Tucker conditions (K2) and the initial tableau is set as:

$$v = b + Ax + P'u$$
$$y = q - Px$$

An important distinction between two kinds of solutions must be stated: A tableau is said to be in standard form if $v'x + u'y = 0$ holds. Otherwise it is in nonstandard form. The initial tableau is in standard form: $v$ and $v$ are basic with nonzero values and $v$ and $x$ are nonbasic, hence their values are zero. Whenever a tableau is in nonstandard form, the next iteration takes it back to the standard form.

The simplex method is performed with some special rules for interchanging the variables: If a tableau is in standard form, the variable to enter the basis is the $x_k$ whose corresponding $v_k$ (basic) has the largest positive value. If a tableau is in nonstandard form, the variable to enter the basis is the nonbasic $v_j$ whose corresponding $x_j$ is also nonbasic. The variable to be deleted from the basis is
selected with the same minimum ratio criterion used in the simplex method for linear programming [7, 10].

The objective is to minimize \( \sum_{j=1}^{n} v_j \). A standard tableau is optimum when the values of \( v \) in the basis are all negative; that is, no \( v \) applies for a new iteration. The \( x \) vector in this optimum tableau is the solution to the problem.

4.3 EXPERIMENTATION

In Braitsch's study some preliminary experiments were carried out to help select the factors and levels for the primary experimentation. Listed below are the nine factors selected that may be important in determining how fast a quadratic programming algorithm will converge.

1. Algorithm: Refers to the algorithms of Wolfe, Dantzig, Beale and W.B. The following factors apply to each of the four algorithms.

2. Rule: Rule used in each algorithm for selecting the variable to enter the basis. In all cases but three, the variable entered was one that resulted in the largest rate of increase in the criterion function. The exceptions were W.B., where no selection rule was necessary and the nonstandard form cases of Beale and Dantzig.
3. Size: Defined as the number of x-variables or the number of constraints in the P matrix, since squared P-matrices of size m x m were used. Sizes of 15, 20, and 25 were used.

4. Negative b: The number of negative coefficients in the cost vector b. Levels of 3, 9, and 15 were used.

5. Rank: Number of linearly independent columns in the matrix A. Ranks of 3 and 15 were used.

6. Range: Range of the random numbers used to generate the P-matrix on the computer. The ranges (r) used were: -20 ≤ r ≤ 20 and -180 ≤ r < 180.

7. b-range: Range of the random numbers used to generate the b-vector. Ranges used were -20 ≤ r < -1 and -180 ≤ r < -1.

8. Q-range: Ranges for the random numbers in the q vector. Ranges used were 1 ≤ r ≤ 20 and 1 ≤ r ≤ 180.

9. P-range: Only the single level of 1 ≤ r ≤ 180 was used for the random numbers that made up the P-matrix.

A complete factorial model was used where, for each of the four algorithms, all combinations of factors (3) through (9) were run. The experiment was replicated ten times. The various factor combinations were run by nesting a series of loops in the computer program, each loop corresponding to a factor. It was assumed that since the problem components
were randomly generated, further randomization would not be necessary. Each randomly generated problem was solved by all four algorithms.

4.4 RESULTS AND ANALYSIS

<table>
<thead>
<tr>
<th>FACTOR</th>
<th>DANTZIG</th>
<th>BPALE</th>
<th>W.B.</th>
<th>WOLF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>15</td>
<td>6.9</td>
<td>7.8</td>
<td>8.3</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>7.5</td>
<td>8.7</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>7.8</td>
<td>9.3</td>
<td>9.6</td>
</tr>
<tr>
<td>Negative p</td>
<td>3</td>
<td>4.5</td>
<td>4.6</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>7.8</td>
<td>8.8</td>
<td>9.4</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>9.9</td>
<td>12.4</td>
<td>12.1</td>
</tr>
<tr>
<td>Rank</td>
<td>3</td>
<td>7.3</td>
<td>8.6</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>7.5</td>
<td>8.7</td>
<td>9.6</td>
</tr>
<tr>
<td>A-range</td>
<td>[-20,20]</td>
<td>6.1</td>
<td>4.8</td>
<td>7.6</td>
</tr>
<tr>
<td></td>
<td>[-180,180]</td>
<td>8.7</td>
<td>12.4</td>
<td>10.3</td>
</tr>
<tr>
<td>b-range</td>
<td>[-20, -1]</td>
<td>7.9</td>
<td>10.1</td>
<td>9.5</td>
</tr>
<tr>
<td></td>
<td>[-180, -1]</td>
<td>6.9</td>
<td>7.1</td>
<td>8.4</td>
</tr>
<tr>
<td>q-range</td>
<td>[1, 20]</td>
<td>7.2</td>
<td>7.7</td>
<td>8.6</td>
</tr>
<tr>
<td></td>
<td>[1, 180]</td>
<td>7.6</td>
<td>9.5</td>
<td>9.3</td>
</tr>
<tr>
<td>Algorithm means</td>
<td>7.4</td>
<td>8.6</td>
<td>8.9</td>
<td>21.5</td>
</tr>
</tbody>
</table>

Table 2 shows the average number of iterations taken at each level of all of the factors that were varied, for each of the algorithms.
The data in this table lead to the conclusion that the rank of the matrix has little effect, but the size factor is very significant. A number of the results for Dantzig's and Beale's algorithms are explainable in terms of the relative importance of the linear and nonlinear part of the objective function. It is clear that increasing the b-range or decreasing the A-range will emphasize the linear part of $G(x)$. Also, decreasing the q-range or increasing the P-range increases the importance of the linear part. This is true because these latter two actions tend to lower $x$-values which reduces the contribution of the quadratic part more than that of the linear part.

Average iterations decreased in the three cases mentioned above, that is increasing the b-range, decreasing the A-range and decreasing the q-range. This is reasonable, because in the extreme case when the quadratic portion is irrelevant and the problem becomes a linear programming problem, both methods reduce to the Simplex method which moves between adjacent extreme points in search of solution. In the nonlinear case, when the quadratic part is significant, the solution often lies in one of the faces in the feasible region rather than at an extreme point. This tends to add iterations because of the necessity of optimizing on faces. These extra iterations often include increasing the number of basic primal variables.
There is a major difference in the way Beale's and Dantzig's methods perform in the Linear Programming case. While Beale's method reduces to the Simplex method in the constraints portion, Dantzig's alternates between regular Simplex iterations and Dual Simplex iterations in two parts of the tableau, thus taking twice as many iterations as Beale's. As the quadratic part becomes more important, average iterations increase at a greater rate in Beale's algorithm than in Dantzig's. The major reason for this is that when we move from one face in the feasible region to another (by making a sign restricted variable nonbasic), it is necessary to replace all of Beale's free variables because these variables were used to optimize on the previous face.

Since each unit of size augmented adds a constraint and an x-variable, the new constraint adds another face to the feasible region which could raise iterations if the algorithm had to examine it on the way to the optimal solution.

Negative b is a very important factor in Dantzig's and Beale's methods. Each added negative b-coefficient aids a variable that can profitably enter in the first iteration for both algorithms. Although not all of the x-variables having negative b-coefficients will be in the optimal solution, the number will tend to increase with negative b.
and iterations will be required to pivot them into the basis. This discussion does not apply to Wolfe's algorithm because it uses a different type of objective function than the other three algorithms. Algorithm WB generally requires fewer iterations because it uses only one artificial variable. The performance of this algorithm is difficult to explain, for although it is a modification of Wolfe's algorithm, it reacts to different problem types in the same way as Dantzig's.

4.5 COMPUTER TIME AND STORAGE

It is difficult to get an accurate measure of the computer time required by the various algorithms because of the large difference that programming technique can cause. A more reliable measure might be based on the number of tableau elements that must be transformed at each iteration. In this way, we can take advantage of theoretical shortcuts for improving time efficiency that might normally be ignored by a programmer. These shortcuts include transforming only half of Beale's Objective tableau because it is symmetric, eliminating transformations in Dantzig's standard-form tableaux because of symmetry relations and reducing the
space that must be set aside due to free variables in Beale's algorithm by noting that the number of these free variables cannot exceed the rank of the $A$-matrix.

In the experimentation, approximately 70 percent of Dantzig's tableaux were in standard form. All tableaux transformations were assumed to take one unit of time, except for the transformations in Beale's objective tableau, which took about 1.6 units in computer tests. The time to transform variables added in Beale's simplex tableau due to free variables will be ignored because it is difficult to estimate, and judging from data available, probably would not affect the time estimate by more than a few percent at worst. Table 5 gives time and storage estimates for several combinations of problem parameters.

Timewise, Beale's and Dantzig's algorithms are superior, with Beale's algorithm tending to be preferable when the number of $x$-variables does not greatly exceed the number of constraints. This is due to the fact that Beale's tableaux are in terms of the nonbasic variables. In terms of storage, Beale's algorithm takes better advantage of situations where the $B$-matrix is nearly square or the rank of $B$ is not a very large fraction of the number of rows or columns in $A$. The latter phenomenon is due to a reduction in space necessary
TABLE 3
TIME AND STORAGE REQUIREMENTS

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>I-rank</th>
<th>Algorithm</th>
<th>Time</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>Z</td>
<td>BEALE</td>
<td>1.8</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DANTZIG</td>
<td>2.6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W. B.</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WOLFE</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Z</td>
<td>BEALE</td>
<td>5.2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DANTZIG</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W. B.</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WOLFE</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Z</td>
<td>BEALE</td>
<td>16.8</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DANTZIG</td>
<td>16.3</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W. B.</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WOLFE</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Z</td>
<td>BEALE</td>
<td>16.8</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DANTZIG</td>
<td>16.3</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W. B.</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WOLFE</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

With, N: number of variables, M: number of constraints
Time: Time per iteration (Z^2 units) and one unit = 23 x 10^{-6} secs. on the IBM 360-75
Storage: Requirements of storage in memory (Z^2 locations)

To hold the rows of variables driven out of the non-basic set because of the introduction of free variables.

The results of this study suggest that in most of the cases, Beale's algorithm will generally take more iterations than Dantzig's, but the difference is not very big. But Beale's algorithm has some other advantages: It will find a local minimum even if A is not definite which may be of value in some applications; second, a redefinition of the
A-matrix, which when combined with a computer code using the
product form of the inverse, can solve larger problems than
Dantzig's algorithm, provided the rank of \( A \) is low.
5.1 STRUCTURE OF THE PACKAGE

The method of Beale was selected among the methods available after considering several important factors and, specifically, because

1. Beale's method does not require matrix \( A \) to be positive definite; it works also with \( A \) negative definite as well as in the semi-definite case.

2. The tableaux used are small, giving economy in storage and computing time.

3. It uses the simplex method, thus is efficient.

4. The computational properties are acceptable or superior to the other methods compared.

5. It is the geometrically most illustrative among all methods for the QPP.

6. The procedure is relatively simple, thereby facilitating the programming task.

Figure 4 provides a general view of the package format. It consists of two major parts: the first part deals with the communication with the user, interchange of information and
the transformation of the data into a suitable format for both the user and the program itself. The second part is the procedure that actually solves the problem.

The execution of the package begins by giving the user a summary of its objective, the QPP is defined and some hints are given for putting the general objective function into matrix form. The user is informed about the procedure for communicating with the package.

No special format for the input data from the terminal is needed. The data entered by the user is checked and only the expected type of data is accepted. This is made with the aid of TAPE 99, a FORTRAN device that enables the reading of data with variable format.
The data received is displayed again on the terminal and the user is asked to check its accuracy. Corrections are permitted when errors are found. The tableau for the method of Beale is constructed from the data received and various parameters are initialized. The initial tableau is displayed if the user has chosen this output option.

The Beale algorithm is then initiated to solve the QPP, and again, according to the user's selected output option, intermediate tableaux and/or the final solution are displayed.

5.2 STORAGE ORGANIZATION

The procedure requires the matrix

\[
\begin{bmatrix}
  c & b'/2 \\
  b/2 & \frac{1}{2}
\end{bmatrix}
\]

which is \((n+1)\times(n+1)\) and symmetric (because \(I\) is symmetric). In order to save storage, only the upper triangular part of this matrix is stored. The \(m\times(n+1)\) matrix

\[
\begin{bmatrix}
  q & -P
\end{bmatrix}
\]

is also created.
All 2-dimensional arrays required in the procedure are stored in a vector format which is created from the concatenation of the rows of the matrix. For example, the 2x3 matrix \( P \), where

\[
\begin{bmatrix}
P_{11} & P_{12} & P_{13} \\
P_{21} & P_{22} & P_{23}
\end{bmatrix}
\]

is stored as the 6-vector

\[
\begin{bmatrix}
P_{11} \\
P_{12} \\
P_{13} \\
P_{21} \\
P_{22} \\
P_{23}
\end{bmatrix}
\]

It then follows that the entry in row \( i \), column \( j \) of \( P \) appears in \( v(k) \) where

\[
k = (i-1)\alpha + j
\]

and \( \alpha \) is the column dimension of \( P \) (\( \alpha = 3 \) in this example).

A slight modification of this process is used to store only the upper triangular part of symmetric matrices.

A means for distinguishing the variables that belong to the basis is required. For this purpose, the original nonbasic variables are always given the indices 1 through n.
and the original basic variables the indices $n+1$ through $n+m$. The free variables, referred to as "u" variables in chapter 3, are treated internally as x variables and they are assigned indices between $n+m+1$ and $2n+m$. This was done to simplify their treatment and there is no risk of confusion for the user since these variables do not appear in the solution vector.

5.3 NOTATION AND DEFINITIONS

Throughout the programs, whenever possible, the same notation as that in chapter 3, was used.

$N$ The number of variables (user input); remains fixed during the execution of the procedure.

$M$ The number of constraints (user input); may vary during the procedure when free variables are added (each one yields a new constraint).

$VA$ A $\frac{N(N+1)}{2}$ - vector containing the upper triangular part of the A matrix in the quadratic objective function (user input).

$B$ Coefficient vector of the linear part of the objective function (user input).
C The constant value in the objective function (user input).

VP \( \mathbf{1} \times \mathbf{M} \times \mathbf{N} \) - vector containing the entries of the constraint matrix \( \mathbf{P} \) (user input).

Q \( \mathbf{M} \times \mathbf{N} \) - vector containing the right-hand side values of the constraints (user input).

EPS A small positive number corresponding to the width of the zero-band used in checks for zero at critical points during the solution (user input).

BASIC0 \( \mathbf{N} \times \mathbf{N} \) - vector containing the indices of the nonbasic variables.

BASIC1 \( \mathbf{N} \times \mathbf{N} \) - vector containing the indices of the basic variables.

LOS Vector containing the variables remaining in the objective function which are different from the free variables.

I3S Vector of the basic variables.

LIBRE Counts the free variables added to the problem.

ALIBRF Stores the number of free variables at a given point.

IPIV Stores the index of the pivot row.

KPIV Stores the index of the pivot column.

PL Set to \( n+1 \).

PICO Logical variable whose value is .TRUE. if the pivot is in the CONSTRAINTS part; otherwise, its value is .FALSE.
ISDON E  Has a value 1 if the problem has been solved; otherwise it has a value of 0.

MAX  Contains the maximum value in the first row of the Objective function tableau.

Y0  Contains the value of the minimum ratio \(a(i,0)/a(i,j)\) for which \(a(i,0)\) is nonnegative and \(a(i,j)\) is positive.

JC  Set to \(N*(N+1)/2\).

J1  Set to \(N*M\).

SOL  The vector which contains the problem solution.

DISPLAY  The value assigned to this variable (by the user) serves to specify one of several available display options.

ITFP  Counts the number of iterations that have been carried out.

5.4 THE SUBPROGRAMS

The overall program is divided into twenty one FORTRAN subroutine subprograms. Each performs a specific task. These subroutines are linked and controlled by the main program which serves as a driver routine.
5.4.1 The Main program

This program calls subroutines DIALOG, INITIA, SCMTX, SHCWIT, and FUNIT. It interchanges information among them and transfers control from one to the other as the solution process is carried out.

5.4.2 Subroutine DIALOG

Presents the package to the user, defines the problem and the variables, gives instructions for the use of the program and gets from the user the values of the parameters to be used in the problem, checks the validity of these values and if necessary accepts any correction suggested. The subroutines called from within DIALOG are READIT, READM and MSGAG. This subprogram is called only by the main program.

5.4.3 Subroutine READIT

Reads one integer value from the terminal (specifically for N, M and DSPLAY). Analyzes the data received and sends messages to the user when necessary. This subprogram calls subroutine MSGAG and is called only by subroutine DIALOG.
5.4.4 Subroutine READEM

This program reads a predetermined set of real values from the terminal, transforms the data into double precision numbers and analyzes each input accepts it or sends an error message when necessary. It is used to read the values of A, B, C, P, Q and PPS. This subprogram calls subroutine M7SSAG and is called only by subroutine DIALOG.

5.4.5 Subroutine INITIA

It initializes the values of the parameters to be used in the program. Sets values for BASICO, BASICO, LCF, LASF, JC, J1, LIBF, LIBFF, LIBFF, PL and ITFF. This subprogram is called only by the Main program.

5.4.6 Subroutine SETMTX

Transforms the data received from the user into a form suitable for use in Beale's method; i.e., creates the matrix $\Gamma$ shown in section 3.1. It also creates the matrix $[q \mid \Gamma]$ for the constraints tableau. This subprogram is called only by the Main program.
5.4.7 **Subroutine SHOWIT**

Displays information relating to the tableau according to the value of DSPLAY. It is called at the end of each iteration to inform the user about the progress of the computations. If DSPLAY = 2, subroutines PROFW and PRICCM are called. If DSPLAY = 1, the current solution is displayed. If DSPLAY = 0 control is returned to the calling routine. This subroutine is called by both, the Main program and the subroutine RUNIT.

5.4.8 **Subroutine RUNIT**

It is the subroutine that carries out Beale's algorithm. It calls subroutines MALIB, COLPIV, SETV, PATIO, CONACT, OPVACT, POONE, SHOWIT, CHANGE, DELPM and OPTIMO. It performs certain operations and transfers control to the subroutines, according to the results obtained. Controls the interaction of all the subroutines and identifies when the problem does not have a solution or when a solution has been found. This subprogram is called only by the Main program.

5.4.9 **Subroutine MALIB**

This subroutine finds the largest value in the first row of the objective function tableau among the columns associated with the free variables. It is called each time a free variable is added to the problem. If MALIB finds any
positive value, this position defines the column of the pivot for the next iteration. It is called only by subroutine PUNIT.

5.4.10 Subroutine COLPIV

It finds the column of the pivot for the next iteration. If MALIB has not found a positive value, COLPIV searches in the first row of the objective function tableau for the most negative number and defines that position to be the column of the pivot for the next iteration. If COLPIV doesn't find any negative value, we are at the optimum point. COLPIV is called only by subroutine PUNIT.

5.4.11 Subroutine SETV

SETV sets the value of a variable V to be either 1 or -1, according to the sign of the element in the matrix A located at the intersection of its main diagonal and the pivot column. It is called only by subroutine PUNIT.

5.4.12 Subroutine RATIO

This subroutine finds the pivot row. To do this, it determines the least value of the ratio \( a(i,0)/a(i,j) \) where \( j \) is the column of the pivot, as \( i \) takes on its admissible values. If there are 2 different \( i \)'s which yield the same minimum value, then this operation is repeated for all \( j \)'s.
different from the pivot column. This second step is necessary in order to avoid degeneracy in the next iteration. It is called only by subroutine "UNIT".

5.4.13 **Subroutine CONTACT**

It updates the tableau of the constraints. Performs the Simplex procedure to interchange the variables determined by the pivot. This subroutine is called only by subroutine "UNIT".

5.4.14 **Subroutine OFUACT**

It updates the tableau of the objective function. Performs the necessary computations to update the tableau at each iteration. It is called by subroutine "UNIT".

5.4.15 **Subroutine ADDONE**

Adds one more free variable to the problem. This means also a new constraint to the system that goes to the simplex tableau. This subroutine is called only by subroutine "UNIT".

5.4.16 **Subroutine CHANGE**

It performs the interchange of variables after each iteration and modifies the values of the parameters affected by this change. It is called by subroutine "UNIT".
5.4.17 **Subroutine DELETE**

This subroutine deletes a constraint from the system, after the value of the free variable that originated it has reached a zero value in the objective function and a new constraint enters the system. It is called by the subroutine `UNIT`.

5.4.18 **Subroutine PROFUN**

It prints the tableau of the objective function. It reconstructs the matrix $T$ and shows the result. Up to five columns are displayed at one time. If the tableau has more than five columns, they are successively displayed in groups of five until all have been shown. This subprogram is called by subroutines `SHOWIT` and `OPTIMO`.

5.4.19 **Subroutine PRICON**

This subroutine prints the tableau of the constraints. It reconstructs the $P$ matrix and as in PROFUN, only five columns are shown at a time. The remaining columns are displayed in subsequent tableaux. PRICON is called by subroutines `SHOWIT` and `OPTIMO`.

- 61 -
5.4.20 **Subroutine OPTIMO**

It displays the problem solution. Prints a suitable message depending on whether ISDONE equals 0 or 1. If DISPLAY = 2, it calls PFOFUN and PFICON, otherwise, creates the solution vector SO1 and displays it with the appropriate messages. It is called by subroutine FUNIT.

5.4.21 **Subroutine MESSAG**

Prints messages for the user. It is called by subroutines DIALOG, PFOFUN, PFICON, READEM, READIT, SHOWIT and OPTIMO.

5.5 **PROGRAM ASSEMBLY**

Figure 5 shows how the routines are interconnected. The darker blocks represent the subprograms that are sequentially called within the main program. The dotted arrows represent subroutine calls.
Figure 5: THE PPOGFAM AND THE SUBROUTINES
Chapter VI
USING THE PACKAGE

6.1 CONNECTING TO THE PACKAGE

The package is to be used under VM-CMS. To run it, the user must have the following files in his disk:

CUADRO EXEC
SETB99 TXTLIB
CUADRO FORTRAN

The file CUADRO EXEC contains:

GLOBAL TXTLIB FORTRAN
FORTRAN 81
FI 5 TERMINAL(FFPFT F LRECL 80
FI 6 TERMINAL(FFPFT F LRECL 80
CP SPOOL CONS START *
LOAD 81 (START
CP SPOOL CONS STOP CLOSE
TYPE --------- -------- -------- -------- -------- -------- --------
TYPE TYPE 'CUADRO PROBLEM' TO ENTER FILENAME FILETYPE
TYPE --------- -------- -------- -------- -------- -------- --------
EXEC RECEIVE

- 64 -
The file SFEB99 TXTLIB (also known as "tape 99" or "buffer 99"), is a 132-byte array that exists in the region of core assigned to the programme. The space for this buffer is automatically created if the compiler detects the use of "tape unit 99".

The file CUADRO FORTRAN contains the source program of the package (see Appendix B).

To run the package, the user must type the command

cuadro cuadro
6.2 PREPARING THE DATA

The package is organized to handle any problem that is in the following standard form

\[
\text{minimize } \{ (1/2)x'x + b'x + c \mid P x < q, \; x \geq 0 \}
\]

To illustrate how a problem can be put into standard form, the example discussed in section 3.2 is taken as model. Suppose the problem is given as minimize \( G \)

where

\[ G = -6x_1 + 2x_1^2 - 2x_1x_2 + 3x_2 \]

Subject to
\[ x_1 + x_2 \leq 2 \]
\[ x_1, x_2 \geq 0 \]

It can be easily verified that this is equivalent to minimize \( G \)

where

\[
G = (1/2)\begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -6 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [0]
\]

Subject to
\[
\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq [2], \; x_1, x_2 \geq 0
\]

The values assigned by the user (see Appendix *) are as follows
\[ N = 2 \quad M = 1 \]

\[ VA = \begin{bmatrix} 4 \\ -2 \\ 4 \end{bmatrix} \quad b = \begin{bmatrix} -6 \\ 0 \end{bmatrix} \quad c = 1.01 \]

\[ VP = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad q = [2] \]

Note that \( q \) must be strictly positive. If in a problem the standardized constraint part is not immediately available (by inspection), a basic starting solution must be searched. The technique discussed in section 2.3.1 can be used here, since the constraints part of the QPP has the same form as that of the linear programming problem.

6.3 MESSAGES

During execution of the package, various types of messages are transmitted to the user's terminal. These can be

1. Informative messages: where no answer is expected from the user; i.e. messages outlining the package, its outputs, etc.

2. Data requesting messages: where an answer (numerical data) is expected from the user; i.e.
VALUE OF n (i.e. the number of variables you have'),
'type the value of your m-vector q. (the right-hand
side value in your constraints)', etc.

3. Control messages: where an answer 'yes' (hit return)
or 'no' (type no) is expected from the user, example
'you have n variables?', 'are there the correct
values of your m-vector q? (the right-hand side terms
in the constraints)?', type q q ... q', etc.

4. Error messages: when the data is of a different kind
than expected, the user is asked to input again, example
'---------> # ***** ERROR IN YOUR ENTRY: 12#456
input again of type q to quit'
'the number of constraints cannot be negative! try
again ..

etc.

6.4 INTERPRETING THE RESULTS

If the value in the display option (DISPLAY) is 0, the
package provides only the optimum value of the objective
function together with the associated x-vector. If the
choice is 1 the current value of the objective function a-
each iteration is provided, from the starting point to the final one.

When the choice of display is 2, the two main tableaux are displayed at each iteration. An example of such a display is shown in Table 4.

Indeed, the form of the tableaux used in Peale's method contains complete information about the problem at any step. The element in the upper left corner the objective function tableaux (i.e., g(1,1)) contains the value of the objective function in that trial point. In the starting point, this corresponds to the value of c.

All the variables in the objective function tableaux have a value of zero. All the variables in the constraint tableaux have a nonzero value, given by the constant part in the constraint equation. However, if a variable has a subscript greater than N, then it is a slack variable (or a free variable) which has a zero coefficient in the objective function, hence it has no effect on the value of G.

The package contains a feature which permits the generation of a hard-copy of all information displayed on the screen during the computing session. This is achieved in the following way:

1. After the completion of the OPP solution, some information is displayed as the process of storing the solution file is progressing. The user is asked
TABLE 4  
AN EXAMPLE OF TABLE OUTPUT

<table>
<thead>
<tr>
<th>ITERATION NUMBER 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OBJECTIVE FUNCTION</strong></td>
</tr>
<tr>
<td>X5</td>
</tr>
<tr>
<td>-0.2074D 02</td>
</tr>
<tr>
<td>X5</td>
</tr>
<tr>
<td>X2</td>
</tr>
<tr>
<td>X3</td>
</tr>
<tr>
<td>X6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CONSTRAINTS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>X5</td>
</tr>
<tr>
<td>X1 = 0.4000D 00</td>
</tr>
<tr>
<td>X4 = 0.6000D 00</td>
</tr>
<tr>
<td>X7 = 0.6667D 00</td>
</tr>
</tbody>
</table>

THE CURRENT VALUE OF THE OBJECTIVE FUNCTION IS -20.74000D00

THE VALUES OF THE VARIABLES ARE
X1 = 0.40000000
X4 = 0.60000000
X7 = 0.66666700

ALL THE OTHERS ARE EQUAL TO ZERO

to type CUADFO PROGRAM as the FILENAME and FILETYPE
requested. If <CR> is made instead, no information is received.

2. The message 'DO YOU WANT A HARD-COPY OF THE INFORMATION DISPLAYED?' (<CR> IF NO, ELSE 'YES') is displayed and the desired answer is expected from the user.

3. If the answer is 'YES', the PRINTOUT command is sent, and the user is asked to type his batch account number and name.

4. If the answer is <CR> (no), the file CUMPRO PROBLEM is erased from the user's disk.
Chapter VII
CONCLUSIONS

The interactive computing environment is acknowledged as a far superior problem solving environment than the batch mode. The intent of the work outlined in this thesis has been to provide such an environment for solving the quadratic programming problem. The package which has been developed makes possible a convenient and straightforward solution to problems of this class and in particular simplifies the process of specifying the problem parameters. Very little technical expertise is required of the user. Note also that the package is able to handle the situation where M=0; i.e., when the problem is unconstrained.

A highly modular organization has been followed in the development of the programme which should facilitate extensions or changes if these become required. The CUADRO package is a complete unit on its own. It could nevertheless be easily restructured to form part of some other package e.g., a general function minimization package. The calculations in the programme are carried out in double precision mode in order to minimize the influence of round-off errors. Inasmuch as FORTRAN IV was used as the programming language, extensive use of structured
programming concepts could not be employed; nevertheless an attempt was made to avoid excessive use of branching (GO TO) statements in order to improve the readability of the code.

At the present time, the dimensioning of various arrays in the program restricts the total number of constraints and variables to be less than 100. In order to handle larger problems, these arrays would have to be appropriately re-dimensioned.
Appendix A

AN EXAMPLE PROBLEM FUN WITH "HF PACKAGE"

GLOBAL TXTLIB FORVLIB
FORTRAN CUADRO

FI 5 TERMINAL (RECFM F LRECL 80
FI 6 TERMINAL (RECFM F LRECL 80
CF SPOOL CONS START *
LOAD CUADRO (START
EXECUTION BEGINS...

THIS PACKAGE USES THE METHOD OF BALIT TO SOLVE THE QUADRATIC PROGRAMMING
PROBLEM, HAVING N VARIABLES AND M CONSTRAINTS (M<=N), GIVEN IN ITS STANDARD
FORM:

\[ \text{MIN } G = \frac{1}{2} x'Ax + c'x + C \]

\[ \text{SUBJECT TO:} \quad \begin{align*}
px & \leq 0, & \text{with } o(j) > 0 \\
x(k) & \geq 0, & \text{for } k = 1, 2, \ldots, N \\
& & \text{for } j = 1, 2, \ldots, M
\end{align*} \]

WHERE:
- C: THE CONSTANT VALUE OF YOUR OBJECTIVE FUNCTION
- B: THE VECTOR OF THE LINEAR PART IN THE C, P
- A: AN NXN MATRIX CONTAINING THE QUADRATIC COEFFICIENTS OF THE C, P
- P: AN NXN MATRIX WITH THE COEFFICIENTS OF THE CONSTRAINTS
- Q: AN M-VECTOR WITH THE RIGHT-HAND SIDE VALUES OF THE CONSTRAINTS

NOTE THAT MATRIX P IS A SYMMETRIC MATRIX WITH \( p(i,j) = 2 \text{TWICE THE COEFFICIENT OF} x(i)x(j) \) WHEN \( i = j \) (THE DIAGONAL OF THE MATRIX)

COEF. OF \( x(i)x(j) \) WHEN \( i \) DIFFERENT FROM \( j \) (THE OFF-DIAGONAL ELEMENTS)

WHEN TYPING THE DATA REQUESTED, PLEASE RECALL:
* AT LEAST ONE BLANK (SPACE) MUST SEPARATE SUCCESSIVE NUMBERS
* THE DECIMAL POINT IS NOT NECESSARY, UNLESS IT IS PART OF THE NUMBER
* IF YOU WANT TO QUIT TYPE "Q" INSTEAD OF THE INFORMATION REQUESTED
* WHEN ASKING TRUE OR FALSE, TYPE "NO" AND <CR> IF FALSE, <CR>
  <CR> (HIT RETURN) IF TRUE

- 74 -
TYPE THE VALUE OF N (I.E. THE NUMBER OF VARIABLES YOU HAVE)

-3

YOUR VALUE OF N = -3 IS NOT POSITIVE!
RESTART ...

TYPE THE VALUE OF N (I.E. THE NUMBER OF VARIABLES YOU HAVE)

2

TYPE THE VALUE OF M (I.E. THE NUMBER OF CONSTRAINTS YOU HAVE)

3

THE STANDARDIZED PROBLEM REQUIRES M <= N ...
IN YOUR PROBLEM, A MAXIMUM OF 2 CONSTRAINTS (M) WILL BE ACCEPTED

TYPE THE VALUE OF M (I.E. THE NUMBER OF CONSTRAINTS YOU HAVE)

1

THE QUADRATIC PART OF THE OBJECTIVE FUNCTION, BEING A SYMMETRIC MATRIX, ONLY THE UPPER-TRIANGULAR PART MUST BE TYPED

TYPE THE 2 VALUES OF THE ROW NUMBER 1

4 -2

TYPE THE VERY LAST ELEMENT OF YOUR MATRIX

5

TYPE THE 2-VECTOR B, (THE LINEAR PART OF YOUR OBJECTIVE FUNCTION)

-6 0

TYPE THE VALUE OF C, I.E. THE CONSTANT PART IN YOUR OBJECTIVE FUNCTION

0

--------- > 0

***** ERROR IN YOUR ENTRY:

INPUT AGAIN OR TYPE Q TO QUIT

TYPE THE 2 VALUES OF THE ROW NO. 1 IN YOUR CONSTRAINTS MATRIX

1 1.5

TYPE THE VALUE OF Q IN YOUR CONSTRAINT (I.E., THE RIGHT-HAND SIDE VALUE)

2

THE VALUES IN THE ZERO-BAND ARE TAKEN AS ZERO IN CRITICAL POINTS OF THE PROBLEM.

TYPE THE VALUE OF THE ZERO-BAND FOR YOUR PROBLEM

1.3-7

NOW, LET US CHECK THE CORRECTNESS OF THE DATA RECEIVED

TYPE "NO" AND <CP> IF FALSE, OR
<CP> (HIT RETURN) IF TRUE

- 75 -
YOU HAVE 2 VARIABLES?

YOU HAVE 1 CONSTRAINTS?

IS THE LAST ELEMENT OF THE MATRIX IN YOUR OBJECTIVE FUNCTION?
4.0000 -2.0000

IS 5.0000 THE LAST ELEMENT OF THE MATRIX IN YOUR OBJECTIVE FUNCTION?

TYPE THE VERY LAST ELEMENT OF YOUR MATRIX
4

IS 4.0000 THE LAST ELEMENT OF THE MATRIX IN YOUR OBJECTIVE FUNCTION?

ARE THESE THE CORRECT VALUES OF YOUR 2 VECTOR B
(THE LINEAR PART IN YOUR OBJECTIVE FUNCTION)?
-6.000 0.0

THE CONSTANT VALUE C IN YOUR OBJECTIVE FUNCTION IS 0.0?

ARE THESE THE CORRECT VALUES OF ROW NUMBER 1
IN YOUR MATRIX P (CONSTRAINTS)?
1.0000 1.5000

TYPE THE 2 VALUES OF THE ROW NO. 1 IN YOUR CONSTRAINTS MATRIX P
1 1

ARE THESE THE CORRECT VALUES OF ROW NUMBER 1
IN YOUR MATRIX P (CONSTRAINTS)?
1.0000 1.0000

ARE THESE THE CORRECT VALUES OF YOUR 1 VECTOR Q
(THE RIGHT-HAND SIDE TERMS IN YOUR CONSTRAINTS)?
2.0000

THE WIDTH OF YOUR ZERO-BAND IS 0.0000000000? 

SELECT ONE OF THE FOLLOWING OPTIONS:
0 >ONLY THE FINAL ANSWER WILL BE SHOWN
1 >THE VALUES OF THE OBJECTIVE FUNCTION AND THE BASIC VARIABLES WILL BE SHOWN AT EACH ITERATION
2 >THE COMPLETE TABLEAUX FOR THE METHOD OF Ruhl WILL BE DISPLAYED AT EACH ITERATION

TYPE THE NUMBER OF THE OPTION YOU CHOOSE
2
ORIGINAL TABLEAU

OBJECTIVE FUNCTION

\[
\begin{array}{ccc}
1 & 2 \\
\text{X 1} & \text{X 2} & \\
0.0 & \text{X 1} & \text{X 2} \\
1 & -0.30000D 01 & 0.0 \\
2 & 0.0 & -0.10000D 01 & 0.20000D 01
\end{array}
\]

CONSTRAINTS

\[
\begin{array}{ccc}
\text{X 1} & \text{X 2} & \\
3 = & 0.20000D 01 & -0.10000D 01 & -0.10000D 01
\end{array}
\]

THE CURRENT VALUE OF THE OBJECTIVE FUNCTION IS 0.0

THE VALUES OF THE VARIABLES ARE

\[
\begin{array}{ccc}
\text{X 3} = & 2.00000D 00 \\
\end{array}
\]

ALL THE OTHERS ARE EQUAL TO ZERO
ITERATION NUMBER  1

OBJECTIVE FUNCTION

\[
\begin{array}{ccc}
1 & 2 \\
\hline
x_4 & x_2 \\
\hline -0.45000D 01 & 0.0 & -0.15000D 01 \\
0.0 & 0.50000D 00 & 0.0 \\
-0.15000D 01 & 0.0 & 0.15000D 01 \\
\end{array}
\]

CONSTRAINTS

\[
\begin{array}{ccc}
& & \\
\hline
x_4 & x_2 \\
\hline x_3 & 0.50000D 00 & -0.50000D 00 & -0.15000D 01 \\
x_1 & 0.15000D 01 & 0.50000D 00 & 0.50000D 00 \\
\end{array}
\]

THE CURRENT VALUE OF THE OBJECTIVE FUNCTION IS \(-4.50000000\)

THE VALUES OF THE VARIABLES ARE
\[
\begin{align*}
x_3 &= 0.50000000 \\
x_1 &= 1.50000000 \\
\end{align*}
\]

ALL THE OTHERS ARE EQUAL TO ZERO
ITERATION NUMBER 2

OBJECTIVE FUNCTION

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X 4</td>
<td>-0.53333D 01</td>
<td>0.33333D 00</td>
</tr>
<tr>
<td>X 3</td>
<td>0.66667D 00</td>
<td>0.66667D 00</td>
</tr>
</tbody>
</table>

CONSTRAINTS

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X 4</td>
<td>0.33333D 00</td>
<td>-0.33333D 00</td>
</tr>
<tr>
<td>X 3</td>
<td>0.66667D 00</td>
<td>-0.66667D 00</td>
</tr>
</tbody>
</table>

THE CURRENT VALUE OF THE OBJECTIVE FUNCTION IS \( -5.33333333 \)

THE VALUES OF THE VARIABLES ARE

\[
\begin{align*}
X_2 &= 0.33333333 \\
X_1 &= 1.66666667
\end{align*}
\]

ALL THE OTHERS ARE EQUAL TO ZERO
<table>
<thead>
<tr>
<th>OBJECTIVE FUNCTION</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x 4</td>
<td>-0.55000D 01</td>
<td>0.0</td>
</tr>
<tr>
<td>x 3</td>
<td>0.50000D 00</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONSTRAINTS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>x 2</td>
<td>0.50000D 00</td>
<td>-0.50000D 00</td>
</tr>
<tr>
<td>x 1</td>
<td>0.15000D 01</td>
<td>0.50000D 00</td>
</tr>
</tbody>
</table>

The current value of the objective function is **-5.50000000**

The values of the variables are:

- x 2 = 0.50000000
- x 1 = 1.50000000

All the others are equal to zero.
SOLUION

OBJECTIVE FUNCTION

\[
\begin{array}{ccc}
1 & 2 \\
X_4 & X_3 \\
-0.50000000 & 0.00000000 \\
X_4 & 0.00000000 & 0.15000000 \\
X_3 & 0.50000000 & 0.00000000 \\
& & 0.50000000 \\
\end{array}
\]

CONSTRAINTS

\[
\begin{array}{ccc}
X_4 & X_3 \\
2 = 0.50000000 & -0.50000000 & -0.50000000 \\
1 = 0.15000000 & 0.50000000 & -0.50000000 \\
\end{array}
\]

THE OPTIMAL IS \(-5.50000000\)

THE VALUES OF THE VARIABLES ARE

\[
\begin{array}{ccc}
X_2 & X_1 \\
0.50000000 & 1.50000000 \\
\end{array}
\]

ALL THE OTHERS ARE EQUAL TO ZERO.

OSCAR MAYFLOUT, 1982

---

- 81 -
CF SPOOL CONS STOP CLOSE
CON FILE 0463 TO VWPSI

TYPE 'CUADRO PROBLEM' TO ENTER FILENAME FILETYPE

EXEC RECEIVE
?DR file 463 unnamed
Enter FILENAME FILETYPE for PAD. <cr>=no RECEIVE
Cuadro problem
RECORD LENGTH IS '132' BYTES.
File CUADRO PROBLEM A RECEIVED from VWPSI

DO YOU WANT A HARD COPY OF THE INFORMATION DISPLAYED?

(<cr> IF NOT, ELSE YES)
yes
PRINTOUT CUADRO PROBLEM
batch account number VWPSLXXX
programmer name OSCAR
RUN FILE 0464 TO OSVS1 COPY=0001 PCDS=000318 NO HOLD
P; T=3.38/4.45 13:06:16
Appendix B
THE FORTRAN PROGRAMS

CALL DIALOG
CALL INITIA
CALL SFTMTX
CALL SHOWIT
CALL FUNIT
STOP
END

SUBROUTINE FUNIT
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER BASIC0,BASIC1,F,L,ALBTPT,F,T,V
REAL*8 MAX
LOGICAL PICO
COMMON /NUMRC/ N,N
COMMON /SUBJCT/ P(230)
COMMON /LIBFEF/ LIB(50),LITPE,F,LIBFF(50)
COMMON /BASICA/ BASICO(50),BASIC1(50)
COMMON /VALUES/ EPS,MIC,TP,P,DISPLAY,KIM
COMMON /MAXIMO/ MAX
COMMON /MINIMO/ YC
COMMON /PARAMET/ V,T,T
COMMON /PIVOTE/ IPIV,KPIV
COMMON /KNUMBER/ KCO,KVA
COMMON /LENGTH/ FL,J,JC,J1
COMMON /LOGICO/ PICO
COMMON /FINAL/ ISDONE
ISDONE = 1
1110 IF (LIBRE. EQ.C) GO TO 1120
CALL MALIB
IF (DABS (MAX) .GT.EPS) GO TO 1200
1120 IF (KVA.LE.0) GO TO 1130
CALL COLPIV
IF (YF.LT.-EPS) GO TO 1200
1130 ISDONE = 1
GO TO 1990
1200 CALL SFTTV
CALL PATTO
IF (IPIV,NE.0.GE.MAX,NE.0) GO TO 1300
ISDONE = 0
GO TO 1990
1300 IF (IPIV,EQ.0.CF,YF.GT.DABS (MAX)) GO TO 1400
I=BASIC0(KPIV)
BASIC0(KPIV)=BASIC1(IPIV)
BASIC1(IPIV)=I
IT=R
P=1
CALL CONACT
IF (ISDONE.EQ.0) GO TO 1990
P=I
PICO=.TRUE.
CALL OPEN
GO TO 1600
1400 ALIBFE(KPIV) = ALIBFE(KPIV) + 1
T=0
I=0
1410 IF (LIBFE.EQ.0) GO TO 1500
DO 16 I=1,LIBFE
   IF (LIBR(I).EQ.KPIV) GO TO 1430
   CONTINUE
   GO TO 1500
1430 CALL ADDONE
CALL CONACT
IF (ISDONE.EQ.0) GO TO 1990
PICO=.FALSE.
CALL OPEN
CALL SHOWIT
GO TO 1110
1500 CALL CHANGE
GO TO 1410
1600 IF (BASIC1(IPIV).LE.N+M+1) GC TO 1650
CALL DELFROM
1650 CALL SHOWIT
GO TO 1110
1990 CALL OPTIMO
RETURN
END

SUBROUTINE INITIA
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER BASIC0,BASIC1,ALIBFE,FL
COMMON /NUMFRO/ N, * 
COMMON /BASIC/ BASIC0 (50), BASIC1 (50)
COMMON /LIBFES/ LIBR (50), LIBR, ALIBFE (50)
COMMON /KNUMFP/ KCO, KVA
COMMON /LENGTH/ FL, JC, J1
COMMON /VALUES/ EPS, MIO, ITDP, DISPLAY, KM
COMMON /OTPOS/ LCS (50), LAS (50)
JC= ((N+1) *(N+2))/2
J1= (M+N+2) * (N+1)
EPS = DABS (EPS/2, D0)
NI=N+1
DO 10 I=1,N
   BASIC0(I) = I
10 CONTINUE
DO 13 K=1,N
   LCS(K) = K
   LAS(K) = K
ALIBF(K) = 0
LIBF(K) = K
13 CONTINUE
IF (M GT 0) GO TO 1102
DO 14 I=1,M
LAS(I) = I
BASIC(I) = N+I
14 CONTINUE
1102 KVA = N
KCO = M
FL = N+1
KIM = N+M
ITDF = *
LIBF = 0
MIO = M
RETURN
END

SUBROUTINE SETMTX
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION G(200), H(100)
COMMON /NUMFPO/ N,M
COMMON /SUBJCT/ P(200), Q(50)
COMMON /FUNCN/ A(100), B(50), C
KM = (N+1)*(N+2)/2
LM = (N+1)*N/2
H(1) = C
JH = 2
DO 15 I=1,N
H(JH) = B(I)/2
JH = JH + 1
15 CONTINUE
DO 16 J=1,LM
H(JH) = A(J)/2
JH = JH + 1
16 CONTINUE
DO 17 K=1,KM
A(K) = H(K)
17 CONTINUE
IT = (M+1)*(N+1)
NI = N+1
DO 11 K=1,NI
G(K) = 0.D0
11 CONTINUE
IN = N
I1 = 1
DO 19 J=1,M
NI = NI + 1
G(NI) = Q(J)
DO 20 IJ=I1,IN
SUBROUTINE SETV
IMPLICIT REAL*8 (A-H,O-Z)
REAL*8 MAX,
INTEGER V
COMMON /PIVOTF/ IPIV,KPIV
COMMON /VALUES/ EPS
COMMON /NUMERO/ N,M
COMMON /OPUNCN/ A(100)
COMMON /MAXIMC/ M
COMMON /PAFAMF/ V
KI=KPIV*(N+1)-((KPIV-1)*KPIV)/2+1
MAX=0.D0
IF(A(KI).*GT.*EPS) MAX=A(KPIV+1)/A(KI)
V=-1
IF(MAX.GT.0.D0) V=1
RETURN
END

SUBROUTINE ADGIE
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER T,F,L,Z
COMMON /PIVOTF/ IPIV,KPIV
COMMON /LENGTH/ FL
COMMON /NUMERO/ N,M
COMMON /PAPPME/ V,T
COMMON /SUBJECT/ P(200)
COMMON /OPUNCN/ ?(100)
Z=0
KAP=KPIV+1
DO 17 K=1,KAP
    JO=K-1
    KU=T+F*L+JO+1
    AJO=KPIV+Z+1
    P(KU)=A(AJO)
17 Z=Z+F-JO
NIL=N-KPIV
IF(NIL.LE.1)GO TO 1450
DO 18 K=1,NIL
$K_0 = T \cdot \text{FL} + (K_{PIV} + K) + 1$

$K_{OJ} = K_{PIV} \cdot (N + 1) - (K_{PIV} \cdot (K_{PIV} - 1)) / 2 + K + 1$

$P(K_0) = \lambda(K_{OJ})$

$1450 \text{IPIV} = "}$

END

SUBROUTINE CHANGE
IMPLICIT REAL*8 (A-H, O-Z)
INTEGER T, P, BASIC1, BASIC0
COMMON /KNUMFC/ KCO, KV
COMMON /PAREME/ V, T
COMMON /LIBRES/ LIBR (50), LIBF
COMMON /PIVOTE/ IPIV, KPIV
COMMON /NUMFC/ V, T
COMMON /BASICA/ BASIC0 (50), BASIC1 (50)
COMMON /OTPOS/ LOC (50), LAS (50)
COMMON /VALUES/ EP, MIT
LIBF(LIBF) = KPIV
BASIC1(M+1) = BASIC0 (KPIV)
BASIC0 (KPIV) = N + MTO + KPIV
LAS(M+1) = M + 1
KCO = M+1
IF (KVA.LT.1) GO TO 1540
DO 19 K = 1, KVA
    IF (LOS (K) . EQ. KPIV) T = K
19 CONTINUE
KVA = KVA - 1
1540 IF (KVA.LT.T.0.P.T. LE.0) GO TO 1560
DO 20 K = T, KVA
    LOS (K) = LOC (K+1)
20 CONTINUE
M = M + 1
T = M
F = 1
RETURN
END

SUBROUTINE DELETE
IMPLICIT REAL*8 (A-H, O-Z)
INTEGER FL, ALIBFP, BASIC1
COMMON /KNUMFC/ N, M
COMMON /KNUMET/ KCO, KV
COMMON /BASICA/ KB0 (50), BASIC0 (50)
COMMON /PIVOTE/ IPIV, KPIV
COMMON /LIBRES/ LIBR (50), LIBF, LIBT (50)
COMMON /OTPOS/ LOC (50)
COMMON /LENGTH/ FL
COMMON /SUBJECT/ P (200)
BASIC1 (IPIV) = BASIC1 (M)
KCO = KCO - 1
KVA = KVA + 1
LOS (KVA) = KPIV
LIBPE = LIBPE - 1
IF (LIBPF .LT. 1) GO TO 1640
DO 21 K = 1, LIBPE
   IF (LIBPF (K) .EQ. KPIV) GO TO 1630
   CONTINUE
   GO TO 1630
21 CONTINUE
GO TO 1640
1630 DO 22 T = K, LIBPE
   LIB (T) = LIB (T + 1)
22 CONTINUE
GO TO 1640
1640 N = N + 1
KIP = KIP + PL + 1
KIA = M + PL + 1
DO 23 K = 1, N
   KII = K - 1
   KIU = KIP + KII
   KIC = KIA + KII
   P (KIU) = P (KIO)
23 CONTINUE
M = M - 1
ALIBPE (KPIV) = ALIBRE (KPIV) - 1
RETURN
END

SUBROUTINE PROFUN
IMPLICIT REAL*8 (/-H,0-Z/)
DIMENSION X(100)
INTEGER BO
COMMON /OFUNC/ A (100)
COMMON /NUMERO/ N
COMMON /BASICA/ BC (50)
COMMON /VALUES/ EPS, MIC, ITPF
IF (ITPE .EQ. 0) CALL MPSSAG (15)
L = 2*N + 1
N = N + 1
L = L + 1
IF (NL .GT. 5) NL = 5
NO = NL - 1
WRITE (6, 214) (K, K = 1, NO)
WRITE (6, 215) (BO (K), K = 1, NO)
DO 216 I = 1, NN
   DO 217 K = 1, NI
      F (K, GT. I) GC TO 219
      K1 = (L - K + 1) * (K - 1) / 2 + I
      GO TO 217
      K1 = (I - I + 1) * (I - 1) / 2 + K
      X (K) = A (K1)
   IF (I .NE. I) GO TO 280
240 WRITE(6,220) (X(K),K=1,NI)
   GO TO 216
280 WRITE(6,222) B0(I-1), (X(K),K=1,NI)
216 CONTINUE
285 IF (NO.EQ.N) RETURN
   NI=NI+1
   NO=N
   WRITE(6,214) (K,K=NI,N0)
   WRITE(6,215) (B0(K),K=NI,N0)
20  226 I=1,NN
   DO 227 K=NI,NO
      K1=K+1
      IF (K1.LT.I) GO TO 228
      K1= (L-I+1)* (I-1)/2+K1
      GO TO 227
228 KI= (L-K1+1)* (K1-1)/2+I
227 X(K)=A(KI)
   IF (I.EQ.1) WRITE(6,230) (X(J),J=NI,N1)
   IF (I.EQ.1) GO TO 226
   WRITE(6,232) B0(I-1), (X(J),J=NI,N1)
226 CONTINUE
250 RETURN

14  FORMAT(/,5X,'OBJECTIVE FUNCTION',/11X,4(12X,-2))
15  FORMAT(/,23X,4(12X,-2))
20  FORMAT(/,3X,5(2X,P12.5))
22  FORMAT(/,1X,'X',12,1X,D12.5,4(2X,D12.5))
23  FORMAT(/,1X,'X',-2,13X,4(2X,D12.5))
   END

SUBROUTINE PPICON
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER B0,B1,PL
COMMON /NUMERO/ N,M
COMMON /BASIC/ B0(50), B1(50)
COMMON /SUBJCT/ P(20C)
COMMON /LENGTH/ PL
N1=N+1
M1=M+1
IF (N1.GT.5) M1=5
NO=N-1
CALL MSGASG(16)
WRITE(6,307) (B0(K),K=1,NO)
DO 306 I=2,MI
   KI=(I-1)*M1+1
   KC=KI+NO
   WRITE(6,309) B1(I-1), (P(K),K=KI,KD)
306 CONTINUE
IF (NO.EQ.N) RETURN
32O NI=NO+1
   NO=N
IF (N0-NI.GT.5) N0=NI + 5 CUA0 3190

WRITE (6,307) (B0(K), K=NI,N0)
DO 310 I=2,M1
   K1= (I-1) * R1 + NI + 1
   K2=K1 + NC-NI
   WRITE (6,311) B1(I-1), (P(K), K=K1,K2)
310 CONTINUE
IF (NO.EQ.N) FFIURN CUPO3660
GO TO 320 COT 03670

307 FORMAT (/,23X,4('X',I2,11X))
309 FORMAT (/,'1X','1X','I2,12.5,4(2X,12.5))
311 FORMAT (/,'1X','1X','I2,13X,4(2X,12.5))
END

SUBROUTINE MALIB
IMPLICIT F8*8 (A-H,O-Z)
REAL*B MAX
COMMON /OFPUNC/ A (100)
COMMON /LIBRES/ LIBF (50), LIBPF
COMMON /NUMREC/ N M
COMMON /PIVOTE/ KDUM, KPIV
COMMON /MAXIV/ MIX
KPIV=LIBF (1)
KOH=LIBF (1) +1
MAX=A(KOH)
IF (LIBF(2),LT,2) RETURN
MAX=A(KOH)
IF (LIBF(2),GT,2) RETURN
DO 402 K=2,LIBF(2)
   IF (LIBF(K),EQ,0) GO TO 402
   KOH=LIBF (K) +1
   IF (A(KOH),EQ,0) GO TO 402
   IF (DABS(MAX),GT,DABS (A(KOH))) GO TO 402
   KPIV=LIBF (K)
   MAX=A(KOH)
402 CONTINUE
RETURN
END

SUBROUTINE OFUNC
IMPLICIT F8*8 (A-H,O-Z)
INTEGER FL, Z
LOGICAL PIC
COMMON /SUBJCT/ P(200)
COMMON /OFPUNC/ A(100)
COMMON /LENGTH/ RI
COMMON /PIVOTE/ TP, KP
COMMON /NUMREC/ N M
COMMON /LOGICO/ PICO
IN=N+1
LO 502 TP=1, IN
   J=IF -1
   IP=1, IN
   J=IF -1
IF (J.EQ.KP) GO TO 502
Z=0
IF (J.GT.KP) Z1 = KP-1
IF (J.LT.KP) Z1 = J
IZ1 = Z1 + 1
DO 504 IS = 1, IZ1
   K = IS - 1
   KH = J + Z + 1
   KH1 = IP*FL + J + 1
   KHO = KP + Z + 1
   A(KH) = A(KH) + P(KH1) * A(KHO)
504 Z = Z + N - K
T = Z + KP
502 CONTINUE
506 IKP = KP + 1
IF (IKP.GT.N) GO TO 53^DO 510 K = IKP, N
   DO 512 J = K, N
   K = J, N
   KH = Z + N - KP + J + 1
   KH1 = IP*FL + J + 1
   KHO = T + K - KP + 1
   A(KH) = A(KH) + P(KH1) * A(KHO)
512 Z = Z + N - K
   DO 522 J = IKP, N
   KH = T + J - KP + 1
   KH1 = IP*FL + J + 1
   A(KH) = A(KH) + P(KH1) * A(T + 1)
522 Z1 = 0
   DO 532 I = 1, KP
   J = I - 1
   Z = 0
   KH = KP + Z1 + 1
   KH1 = IP*FL + J + 1
   SUMA = I (KH) + P(KH1) * A(T + 1)
   DO 534 IS = 1, IIF
   K = IS - 1
   KH = IP*FL + K + 1
   A(KH1) = A(KH1) + P(KH) * SUMA
534 Z = Z + N - K
532 Z1 = Z1 + N - J
IF (IKP.GT.N) GO TO 550
   DO 542 J = IKP, N
   Z = 0
   IF = J + 1
   DO 542 IS = 1, IF
   K = IS - 1
   TP(KNE, KP) GO TO 546
   Z = Z + 1
   GC TO 542
546 KH = J + Z
   KHO = T + J - KP + 1
SUBROUTINE COLPIV
IMPLICIT FPAL*8 (A-H,O-Z)
COMMON /OFUNCN/ A (100)
COMMON /OTFOS/ LGS(50)
COMMON /KNUMFF/ KCO,KVA
COMMON /NUMTFC/ L,M
COMMON /MINIMO/ YO
COMMON /PIVOTT/ IPIV,KP
KP=LGS(1)
KHO=LGS(1)+1
YO=A(KHO)
IF (KVA.LT.2) RETURN
DO 602 K=2,KVA
  KHO=LGS(K)+1
  IF (A(KHO).GE.YO) GO TO 602
  KHO=IPIV(K)
  IF (A(KHO).LT.YO) GO TO 602
  YO=YO-A(KHO)*A(KHO)
  A(KHO)=Y
602 RETURN
SUBROUTINE FATIC
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER PL,V,Z
COMMON /LENGTH/ PL
COMMON /PIVOT/ IF,KP
COMMON /SUBJECT/ P(200)
COMMON /OTFOS/ KLOS(50),LAS(50)
COMMON /NUMEE/ KCO
COMMON /VALUES/ FFS,NTO,ITEP,DSPLAY,KT
COMMON /MINIMK/ YC
COMMON /PARAMK/ V
COMMON /NUMFF/ N,M
IP=1
IF(KCO.LT.1) RETURN
DO 702 I=1,KCO
KH=LAS(I)*FL+1
KH1=KH+KP
IF(V*P(KH1).GT.EPS) GO TO 704
702 CONTINUE
RETURN
704 YO=V*P(KH)/P(KH1)
IP=LAS(I)
Z=I+1
IF(Z.LT.KCO) RETURN
DO 706 I=Z,KCO
KH=LAS(I)*FL+1
KH1=KH+KP
IF(V*P(KH1).LE.EPS) GO TO 706
YP=V*P(KH)/P(KH1)
IF(YP.GE.YC) GO TO 708
IP=LAS(I)
YO=YP
GO TO 706
708 IF(YP.NE.YC) GO TO 706
JO=LAS(I)
DO 710 K=1,N
KHO=IP*FL+K+1
KH2=IP*FL+KP+1
KH=JO*FL+K+1
YP=V*P(KHO)/P(KH2)
YO0=V*P(KH)/P(KH1)
IF(YP.LT.YFO) GO TO 706
IF(YE0.LT.YFP) GO TO 712
710 CONTINUE
712 IP=IO
SUBROUTINE CONTACT
IMPLICIT REAL*8 (I-H, O-Z)
INTEGER PL,K
COMMON /SUBJCT/ P(200)
COMMON /PAPAME/ V,T,T0
COMMON /NUMEC/ K1,K1
COMMON /PIVOTE/ IP,KP
COMMON /LENGTH/ RL
COMMON /FINAL/ I5DONE
KH=IP*PL+KP+1
IF(P(KH).EQ.O) GO TO 820
PIVOTA=1.D0/P(KH)
II0=II+1
III=II+1
KK0=1
KK1=K1+1
IF(III.GT.III) GO TO 802
DO 804 II=II0,III
   IF(I.EQ.IP) GO TO 804
   K=KK-1
   IF(K.EQ.KP) GO TO 806
   K1=II*FL+K+1
   KH2=IP*PL+K+1
   P(KH1)=F(KH1)-P(KH2)*PIVOTA
DO 806 KK=KK0,KK1
   K=KK-1
   IF(K.EQ.KP) GO TO 806
   K1=II*FL+K+1
   KH2=IP*PL+K+1
   P(KH1)=F(KH1)-P(KH2)*PIVOTA
806 CONTINUE
804 CONTINUE
DO 810 KK=KK0,KK1
   K=KK-1
   K2=IP*PL+K+1
810 IF(K.NE.KP) P(KH2)=-P(KH2)*PIVOTA
802 P(KH)=PIVOTA
END

SUBROUTINE SHOWIT
IMPLICIT REAL*8 (I-H, O-Z)
DIMENSION SOL(50)
INTEGER CCL(50)
INTEGER DISPLAY,PL,BASIC1,BISTIC
COMMON /VALUES/ FP*,RIO,ITER,DISPLAY
COMMON /NUMEC/ N*
- 44 -
COMMON /OFUNCN/ J (100)
COMMON /SUBJCT/ P (200)
COMMON /BASICA/ BASICO (50), BASTC1 (50)
COMMON /LENGTH/ RL
IF (DSPLAY .EQ. 0) GO TO 1890
IF (DSPLAY .EQ. 1 .AND. ITEF .EQ. 0) CALL MESSAG (17)
IF (DSPLAY .EQ. 1 .AND. ITEF .EQ. 0) GO TO 1820
IF (DSPLAY .GE. 2 .AND. ITEF .EQ. 0) GO TO 1812
1812 CALL PQOFUN
1820 ITER = ITER + 1
WRITE (6, 106) A (1)
K = M + 1
DO 28 = 1, K
KIO = (I - 1) * FL + 1
SOL (I) = F (KIO)
28 CONTINUE
WRITE (6, 107) (BASIC1 (K), SOL (K + 1), K = 1, M)
CALL PTOCON
1890 RETURN
106 FORMAT ('CURRENT VALUE OF THE OBJECTIVE FUNCTION IS ', *F15.8)
213 FORMAT ('ITERATION NUMBER ',I2)
107 FORMAT ('THE VALUES OF THE VARIABILITY ARE ', '/30X','X',10(I2, *= 'F15.8, '/30X,'X'))
END

SUBROUTINE OPTING
IMPLICIT REAL*8 (A-H, O-Z)
DIMENSION SOL (50)
COMMON /VALUES/ EPS, MIO, IPE, DSPLAY
COMMON /FINAL/ I SDONE
COMMON /NUMBER/ N, M
COMMON /BASICA/ BASICO (50), BASTC1 (50)
COMMON /SUBJCT/ P (200)
COMMON /OFUNCN/ J (100)
COMMON /LENGTH/ RL
IF (ISDONE .EQ. 0) CALL MESSAG (19)
IF (ISDONE .EQ. 0) GO TO 1750
IF (ISDONE .EQ. 1) CALL MESSAG (20)
IF (DSPLAY .EQ. 0 .OR. DSPLAY .EQ. 1) GO TO 1710
CALL PQOFUN
CALL PTOCON
1710 WRITE (6, 106) A (1)
K = M + 1
DO 28 = 1, K
KIO = (I - 1) * FL + 1
SUBROUTINE DIALOG
IMPLICIT F*AL*8 (A-H,O-Z)
DIMENSION G(200), D(100)
INTEGER DISPLAY
COMMON /NUMEG/ N,M
COMMON /SUBJCT/ P(200), Q(50)
COMMON /OFUNCN/ A(100), B(50), C
COMMON /VALUES/ P,H,M,T,DISPLAY
DATA BLANK/' ' /
CALL MESSAG(1)
CALL MESSAG(2)
CALL MESSAG(3)
KEY=0
100 CALL MESSAG(4)
CALL READIT(N)
IF(N.GT.0)GO TO 41
WRITE(*,6) N
GO TO 100
41 IF(KEY.EQ.1)GO TO 1901
102 CALL MESSAG(5)
CALL READIT(M)
IF(M.GT.0)GO TO 42
CALL MESSAG(6)
GO TO 102
42 IF(M.EQ.N)GO TO 43
WRITE(*,6) N
GO TO 102
43 IF(KEY.EQ.1)GO TO 106
CALL MESSAG(7)
KO=0
121 DO 13 J=1,N
IF(JIK.EQ.K)GO TO 104
IF(JIK.NF.0)GO TO 122
JL=J-1
NI=NI-JL
NI=NI
IF(NI.EQ.1)GO TO 104
106 F.CONTINUE
WRITE(*,6) (BASIC1(K), SOL(K+1), K=1,N)
CALL MESSAG(18)
CALL MESSAG(21)
107 FORMAT(/,5X, ' THE OPTIMAL IS ',F15.8)
108 FORMAT(/,5X, ' ALL THE OTHERS ARE EQUAL TO ZERO ',/,'18(-')/,58X, 'CAR MANIFIQUE, 1982',/,78('-'))
END

SOL(I) = P(K+1)
WRITE (6,70) NI, J
GO TO 105

122 WRITE (6,70) NIL, JIK
GO TO 105

104 CALL MESSAGE (8)
105 CALL READEM (D, NIL)

KIN = 1
IF (JIK .NE. C) KC = 10 - NIL - 1
IF (JIK .NE. C) NI = NIL
DO 14 K = KIN, NT
   KO = KO + 1
   A (KO) = D (K)
14 CONTINUE

118 WRITE (6, 68) N
CALL IFADEM (B, N)
IF (KEY .EQ. 1) GO TO 115
116 CALL MESSAGE (9)
CALL READEM (C, 1)
IF (KEY .EQ. 1) GO TO 112
IF (M .EQ. 0) GO TO 140
130 JA = 1
JI = 0
112 DO 11 I = 1, M
   IF (JTO .NE. C .AND. KEY .EQ. 1) GO TO 110
   WRITE (6, 66) N, I
   GO TO 111
110 WRITE (6, 66) N, JIO
111 CALL READEM (G, *)
   IF (KEY .EQ. 1) JIC = I
   IF (KEY .EQ. 1) JA = (JTO - 1) * N + 1
   JI = JIC * N
   J = 1
   DO 12 K = JA, JI
      B (K) = G (J)
      J = J + 1
12 CONTINUE
   IF (KEY .EQ. 1) GO TO 108
   JA = JI + 1
11 CONTINUE
   IF (M .GT. 1) GO TO 103
   CALL MESSAGE (10)
   GO TO 131
103 WRITE (6, 64) H
131 CALL READEM (Q, *)
   DO 10 J = 1, M
      IF (O (J, GTO) .GT. 10) GO TO 10
      WRITE (6, 65) I, O (J)
   GO TO 103
10 CONTINUE
IF (KEY, EQ. 1) GO TO 180
140 CALL MESSAGE (22)
   CALL SPADEN (1, 1)
IF (KEY, EQ. 1) GO TO 181

--- NEXT SECTION CHECKS THE CORRECTNESS OF THE DATA ---

CALL MESSAGE (11)
   KEY = 1
1901 WRITE (6, 901) N
   REWIND 5
READ (5, 33, END=106) CHAR
   IF (CHAR .NE. BLANK) GO TO 100
   IF (N.EQ.O) CALL MESSAGE (12)
   IF (N.EQ.O) GO TO -1
   WRITE (6, 74) N

7001 FFINX 5
READ (5, 33, END=107) CHAR
   IF (CHAR .NE. BLANK) GO TO 102
107 IO=1
   IM=0
   DO 16 IT=1, N
      II=I-1
      IO=IO+NIL
   1017 IF (I.EQ.N) GO TO 128
      IA=IA+NIL
      JO=JO-NIL
      WRITE (6, 79) NIL, I, (A(KI), KI=J0, JA)
      GO TO 119

128 LST=(N+1) *N/2
      WRITE (6, 90) A (LST)
119 REWIND 5
   READ (5, 33, FND=15) CHAR
      IF (CHAR .EQ. BLANK) GO TO 15
      JIK=I
      GO TO 121

16 CONTINUE
115 WRITE (6, 78) N, (B(KI), KI=1, N)
   REWIND 5
READ (5, 33, FND=113) CHAR
   IF (CHAR .EQ. BLANK) GO TO 118
113 WRITE (6, 77) C
   REWIND 5
READ (5, 33, FND=1180) CHAR
   IF (CHAR .EQ. BLANK) GO TO 116
1180 IF (M.EQ.O) GO TO 181
   DO 15 IK=1, M
      KA=IK*X
      KO=(IK-1)*X+1
   108 WRITE (6, 75) N, IK, (P(J), J=KO, KA)
   FFINX 5
READ (5, 33, FND=15) CHAR
   IF (CHAR .EQ. BLANK) GO TO 15

- 98 -
JIC=JK
GO TO 112

15 CONTINUE
16 WRITE (F, 72) "*(Q(I), I=1, M)*
REWIND 5
READ (5, 33, END=181) CHAP
IF (CHAP.NF. BLANK) GO TO 103

181 WRITE (6, 81) * 
REWIND 5
READ (5, 33, END=181) CHAP
IF (CHAP.NF. BLANK) GO TO 140

88 CALL MESSAG(13)
CALL FEPRIT(DISPLAY)
RETURN

60 FORMAT (/,' YOUR VALUE OF N = ', I2,' IS NOT POSITIVE ! ', /)
61 FORMAT (/,' THE STANDARDIZED PROBLEM REQUIRE M <= ',..., /)
62 FORMAT (/,' IN YOUR PROBLEM, A MAXIMUM OF ', I2,' CONSTRAINTS (M) WILL BE ACCOMMODATED'.
63 FORMAT (/,' TYPE THE ', I2,' -VECTOR P, (THE LINEAR PART OF YOUR OBJECTIVE FUNCTION) ')
64 FORMAT (/,' TYPE THE ', I2,' VALUES OF THE ROW NO. ', I2,' IN YOUR COEFFICIENT MATRIX P')
65 FORMAT (/,' TYPE THE ', I2,' VALUES OF THE ROW NUMBER ', I2)
66 FORMAT (/,' TYPE THE VALUE OF YOUR ', I2,' -VECTOR Q, (THE RIGHT-HAND SIDE OF YOUR CONSTRAINTS) ')
67 FORMAT (/,' THE COMPONENT NO. ', I2,' OF YOUR Q VECTOR, NAMELY ', D12.5,' IS NOT POSITIVE: ',/)
68 FORMAT (/,' YOUR PROBLEM IS NOT IN STANDARD FORM, INPUT AGAIN .... ')
69 FORMAT (/,' ARE THESE THE CORRECT ', I2,' ELEMENTS OF THE ROW NUMBER ', I2)
70 FORMAT (/,' IN YOUR OBJECTIVE FUNCTION ? ',/)
71 FORMAT (/,' YOU HAVE ', I2,' CONSTRAINTS ? ')
72 FORMAT (A5) 
73 FORMAT (A5) 
74 FORMAT (/,' YOU HAVE ', I2,' VECTORS ? ')
75 FORMAT (/,' ARE THESE THE CORRECT VALUES OF YOUR ', I2,' -VECTOR B, ', D12.5,' (THE LINEAR PART IN YOUR OBJECTIVE FUNCTION) ?',/)
76 FORMAT (/,' ARE THE CORRECT VALUES OF YOUR ', I2,' -VECTOR Q, ', D12.5,' (THE RIGHT-HAND SIDE IN YOUR CONSTRAINTS) ?',/)
77 FORMAT (/,' ARE THESE THE CORRECT VALUES OF YOUR ', I2,' -VECTOR Q, ', D12.5,' (THE RIGHT-HAND SIDE IN YOUR CONSTRAINTS) ?',/)
78 FORMAT (/,' ARE THESE THE CORRECT VALUES OF YOUR ', I2,' -VECTOR Q, ', D12.5,' (THE RIGHT-HAND SIDE IN YOUR CONSTRAINTS) ?',/)
79 FORMAT (/,' ARE THESE THE CORRECT VALUES OF YOUR ', I2,' -VECTOR Q, ', D12.5,' (THE RIGHT-HAND SIDE IN YOUR CONSTRAINTS) ?',/)
80 FORMAT (/,' ARE THESE THE CORRECT VALUES OF YOUR ', I2,' -VECTOR Q, ', D12.5,' (THE RIGHT-HAND SIDE IN YOUR CONSTRAINTS) ?',/)
81 FORMAT (/,' THE WIDTH OF YOUR ZERO-BAND IS ', D12.5, ' ?')
END

SUBROUTINE MESSAG(INDEX)
GO TO (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22)
FOIMOMA (*:59X,'UNIVERSITY OF OTTAWA',/,'62X,'SYSTEMS SCIENCE',/,'69X, 'CUPA089240 *
,1982',/,'1X,78(',-','),/,'35X,' CUPA089240 */,'1X,78(',-','),/,' THIS PROGRAM
* CRAGF UPS THE METHOD OF BANL TO SOLVE THE QUADRIC PROGRAMMING! CUPA09270
*/,' PROBLEM, HAVING N VARIABLES AND M CONSTRAINTS (M<=N), GIVEN
* ITS STANDARD ';,' FORM : ',//,
* MIN G = (1/2)X'AX + B'y + C ',//,
* SUBJECT TO : PX <= Q ', WITH Q(J) > 0 ',//,
- 100 -
X(K) >= 0  
K = 1, 2, ..., N

WHERE:  
C: THE CONSTANT VALUE OF YOUR OBJECTIVE FUNCTION
B: THE VECTOR OF THE LINEAR PART IN THE O. F.
A: AN NXN MATRIX CONTAINING THE QUADRATIC COEFFICIENTS
P: AN NXM MATRIX WITH THE COEFFICIENTS OF THE CONSTRAINTS
Q: AN M-VECTOR WITH THE RIGHT-HAND SIDE VALUES OF THE CON CONSTRAINTS

F: THE VECTOR OF THE LINEAR PART IN THE O. F.

* OF THE O. F.

NOTE THAT MATRIX A IS A SYMMETRIC MATRIX WITH A(I,J) = A(J,I)

TWICE THE COEFFICIENT OF X(K)**2 WHEN K=I=J (THE DIAGONAL OF THE)

COEF. OF X(I)X(J) WHEN I DIFFERENT FROM J (THE OFF-DIAGONAL ELEMENTS)

THE DECIMAL POINT IS NOT NECESSARY, UNLESS IT IS PART OF THE NUMBER

IF YOU WANT TO QUIT TYPE "Q" INSTEAD OF THE INFORMATION REQUIRED

WHEN ANSWERING TRUE OR FALSE, TYPE "T" OR "F" OR "NO" OR "YES"

THE VALUE OF M (THE NUMBER OF CONSTRAINTS)

THE VALUE OF N (THE NUMBER OF VARIABLES)

THE QUADRATIC PART OF THE OBJECTIVE FUNCTION, BEING A SYMMETRIC MATRIX

ONLY THE UPPER-TRIANGULAR PART MUST BE TYPED

THE VERY LAST ELEMENT OF YOUR MATRIX

TYPE THE VALUE OF C, I.E. THE CONSTANT PART IN YOUR OBJECTIVE

HAND SIDE VALUE

NOW, LET US CHECK THE COEFFICIENTS OF THE DATA RECEIVED

TYPE "NO" AND IF FALSE, OR 'YES'

THE COMPLETE TABLEAUX FOR THE METHOD OF SIMPLE WILL BE DISPLAYED AT EACH ITERATION

YOU DID NOT TRY ANY DATA... INPUT AGAIN OR "Q" TO QUIT
SUBROUTINE FDADM (F, L)
INTEGER A(80), NUMER(12), DOT, BLANK, QUIT
INTEGER FMT(40)
DOUBLE PRECISION F(40)
LOGICAL FLAG
DATA BLANK/' */A/80*/ ' /QUIT/ 'Q'/MN/ 'D' /DOT/ 'D'/
DATA NUMER/'1','2','3','4','5','6', '7', '8', '9', '0', 'D', 'I'/
REWIND 5
READ (5,100,END=77) (A(I), I=1,80)
FLAG=.FALSE.
DO 45 I=1,80
  IF (A(I).EQ.QUIT) CALL EXIT
45 CONTINUE
77 DO 60 J = 1,80
  IF (A(J).NE.BLANK) GO TO 63
60 CONTINUE
CALL MESSAGE (14)
GO TO 40
63 DO 70 I=1,80
  IF (A(I).EQ.MIN. OR A(I).PO.DOT.OR. A(I).PO.BLANK) GO TO 70
  DC 72 K=1,12
  IF (A(I).EQ.NUMER(K)) GO TO 70
72 CONTINUE
  WRITE (6,101) (I)
  FLAG=.FALSE.
  GO TO 75
70 CONTINUE
75 IF (FLAG) GO TO 66
WRITE (6,102) (I(K), K=1,80)
GO TO 40
66 JAK=1
I2=0
Z1=0
K=1
50 IF (A(K).EQ.BLANK. AND I1.NE.0) I2=K-1
IF (A(K).NE.BLANK. AND I1.EQ.0) Z1=K
IF (I2.PO.0) GO TO 10
IF (T2 > T1 + 1, T2 - 1) go to 22
WRITE (6, 104) JAK, (A(K), K = 1, 80)
GO TO 40

22 WPITF (6, 103) (A(KI), KI = I1, I2)
CALL SETB9° (FMT, 10)
IH = I2 - I1 + 1
WRITE (99, 99) IH
CALL FESB9°
READ (99, FMT) F (JAK)
JAK = JAK + 1
I1 = 0
I2 = 0
10 K = K + 1
IF (K > 80) GO TO 78
IF (JAK > L) GO TO 8
GO TO 50
78 RETURN
99 FORMAT (' (D', I1, ', C')
100 FORMAT (80A1)
101 FORMAT (' ---------------------- ', A1)
102 FORMAT (' **** ERROR IN YOUR ENTRY : ', /, 80A1, ', ' INPUT AGAIN OR TYPE 10590
*SEE 0 TO QUIT')
103 FORMAT (91)
104 FORMAT ('/ ', YOUR ENTRY NUMBER ', /12', IS TOO LONG, ' MAXIMUM OF 9 POCU10590
*ITIONS ', / ', WILL BE ACCEPTED. INPUT AGAIN : /, /, 80A1) END

SUBROUTINE READIT(F)
INTEGER A (80), NUMFR (10), FLANK, QUIT
INTEGER F (1), FMT (10)
LOGICAL FLAG
DATA A/80*'A BLANK/'A'/D AOU71*/A' M
DATA NUMFR/'1','2','3','4','5','6','7','8','9', '0'/
40 F E N D 5
READ (5, 106, FND=77) (A(I), I = 1, 80)
DO 45 1 = 1, 80
IF (A(I).EQ. QUIT) CALL EXIT
45 CONTINUE
FLAG = . T P U F.
47 DO 60 J = 1, 80
IF (A(J).NE.BLANK) GO TO 63
60 CONTINUE
CALL MESSAG (14)
GO TO 46
63 DO 70 1 = 1, 80
IF (F(I).EQ.MIN.0) A(I).EQ.FLANK) GO TO 7F
DO 72 K = 1, 10
IF (A(I).EQ.NUMFR(K)) GO TO 7F
72 CONTINUE
WRITE (6, 103) A(I)
FLAG = . F I S F.
GC TO 75

70 CONTINUE

75 IF (FLAG) GO TO 66
   WRITE (6,102) (A(K),K=1,80)
   GO TO 40

66 MIKE=0
   I2=0
   I1=0
   K=1

50 IF (A(K), EQ, BLANK, AND, I1, NE, 0) I2=K-1
   IF (A(K), NE, BLANK, AND, I1, EQ, 0) I1=K
   IF (I2, EQ, 0) GO TO 10
   WRITE (99,101) (A(KI),KI=I1,I2)
   CALL FMTB99 (FMT,10)
   IH=I2-I1+1
   WRITE (99,99) IH
   CALL FFSB99
   READ (99,FMT) F
   MIKE=1
   K=K+1
   IF (K, GT, 80) GO TO 78
   IF (MIKE, NE, 0) GO TO 78
   GO TO 50

78 RETURN

99 FORMAT ('(I1,I1,')')
101 FORMAT (9A1)
100 FORMAT (80A1)
102 FORMAT (*** ENTER IN YOUR ENTRY: ',/,'20A1,'/.' , INPUT AGAIN OR
* TYPE Q TO QUIT')
103 FORMAT ('----------------------' ,A1)

END
BIBLIOGRAPHY


Center, University of California, Berkeley, CA 94720.


