Nonlinear programming : augmented Lagrangian techniques for constrained minimization
by Michael James Lowe

A thesis submitted to the Graduate Faculty in partial fulfillment of the requirements for the degree of DOCTOR OF PHILOSOPHY in Electrical Engineering
Montana State University

Abstract:
The nonlinear programming problem (NLP) is presented in its standard form for a constrained minimization problem. The general method of sequential-unconstrained minimization is then given, showing how the constrained NLP problem functions may be transformed into a single unconstrained function having the same solutions as the original problem. A unique unconstrained function, incorporating both equality and inequality constraints, is formulated by augmenting the Lagrangian with weighted problem functions of the constrained NLP. First-order necessary and second-order sufficient conditions are then established for this function to possess local minima that minimize the objective function and satisfy the constraints of the NLP, thus showing the equivalence of constrained minima and unconstrained minima of the augmented Lagrangian. A multiplier algorithm based on satisfying the Kuhn-Tucker first-order conditions of optimality is presented. The algorithm consists of alternating minimization and update phases. This procedure establishes a way to select an active inequality constraint basis, such that during a minimization phase these inequality constraints exhibit a 'locked on' feature and are treated the same as equality constraints. The inequality constraints that are not locked on during a particular minimization phase are held from excessive constraint violation with the penalty terms of the augmented Lagrangian. Although the algorithm is based on first-order conditions, a theorem is presented showing that if second-order conditions are satisfied by the NLP problem functions, the algorithm converges locally for finite penalty weights. As the sequence of minimizers converge to a local minimum, the Lagrange multipliers also converge to their optimal values. These optimal multipliers provide well-known sensitivity information, and a practical interpretation of these multipliers is presented. A F0RTRAN coded computer program ALG0R that implements the multiplier algorithm is described. It generates conjugate gradient search directions by the DFP method. An efficient quadratic-convergent unidirectional search technique is presented for conducting the search. ALG0R was applied to several test functions, and the results were compared to results obtained by other well-known NLP solution algorithms. When based on the number of function calls required to obtain a desired degree of accuracy in the objective function, ALG0R was shown to be a more efficient code than the multiplier program MULTMETH, and the penalty-function codes OPTKOV, POWCON, and SUMT. The formulation of several real-world problems into the NLP format is presented, along with their solutions and interpretations of the sensitivity information possessed by the Lagrange multipliers at the solution points.

NONLINEAR PROGRAMMING: AUGMENTED LAGRANGIAN TECHNIQUES FOR CONSTRAINED MINIMIZATION

by

MICHAEL JAMES LOWE

A thesis submitted to the Graduate Faculty in partial fulfillment of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Paul E. Schlinck
Head, Major Department

Donald A. Pierre
Chairman, Examining Committee

Henry L. Parsons
Graduate Dean

MONTANA STATE UNIVERSITY
Bozeman, Montana

March, 1974

ACKNOWLEDGMENTS

I am indebted to Dr. Donald A. Pierre who served as my advisor, and gave considerable time and effort in the development of this work. Many of his valuable insights and suggestions have been incorporated into this thesis.

I give special thanks to Dr. Roy M. Johnson and Dr. Donald A. Rudberg for their constructive criticism and useful suggestions during all phases of the research. I acknowledge Dr. Paul E. Uhlrich and the Department of Electrical Engineering for providing the financial support and the use of the computer facilities at Montana State University. I express my sincere appreciation to Mrs. Jean Julian who typed the final draft of this thesis.

Finally, I am thankful for the understanding and encouragement of my wife, Patty, who diligently typed preliminary drafts of this work, and my daughter, Krista, who cheerfully endured and helped when she could. To my family I dedicate this thesis.

Michael James Lowe

March, 1974

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The nonlinear programming problem (NLP) is presented in its standard form for a constrained minimization problem. The general method of sequential-unconstrained minimization is then given, showing how the constrained NLP problem functions may be transformed into a single unconstrained function having the same solutions as the original problem. A unique unconstrained function, incorporating both equality and inequality constraints, is formulated by augmenting the Lagrangian with weighted problem functions of the constrained NLP. First-order necessary and second-order sufficient conditions are then established for this function to possess local minima that minimize the objective function and satisfy the constraints of the NLP, thus showing the equivalence of constrained minima and unconstrained minima of the augmented Lagrangian. A multiplier algorithm based on satisfying the Kuhn-Tucker first-order conditions of optimality is presented. The algorithm consists of alternating minimization and update phases. This procedure establishes a way to select an active inequality constraint basis, such that during a minimization phase these inequality constraints exhibit a 'locked on' feature and are treated the same as equality constraints. The inequality constraints that are not locked on during a particular minimization phase are held from excessive constraint violation with the penalty terms of the augmented Lagrangian. Although the algorithm is based on first-order conditions, a theorem is presented showing that if second-order conditions are satisfied by the NLP problem functions, the algorithm converges locally for finite penalty weights. As the sequence of minimizers converge to a local minimum, the Lagrange multipliers also converge to their optimal values. These optimal multipliers provide well-known sensitivity information, and a practical interpretation of these multipliers is presented. A FØRTRAN coded computer program ALGØR that implements the multiplier algorithm is described. It generates conjugate gradient search directions by the DFP method. An efficient quadratic-convergent unidirectional search technique is presented for conducting the search. ALGØR was applied to several test functions, and the results were compared to results obtained by other well-known NLP solution algorithms. When based on the number of

function calls required to obtain a desired degree of accuracy in the objective function, ALGØR was shown to be a more efficient code than the multiplier program MULTMETH, and the penalty-function codes OPTKOV, POWCON, and SUMT. The formulation of several real-world problems into the NLP format is presented, along with their solutions and interpretations of the sensitivity information possessed by the Lagrange multipliers at the solution points.

CHAPTER ONE

INTRODUCTION TO THE NLP AND SEQUENTIAL UNCONSTRAINED MINIMIZATION

1.1 Introduction

The solution techniques of the nonlinear programming problem (NLP) are a powerful tool in mathematical programming and real-world problem solving and decision making. The NLP arises from many apparently unrelated areas and takes on many forms as found in mathematics, the natural and physical sciences, engineering, economics, agriculture, business, and government.

In the abstract form the NLP is that of optimizing (minimizing or maximizing) some entity while satisfying side conditions or constraints. The entity to be optimized may be profit, cost, production, efficiency, consumer utility, social tension, social benefit, energy, inventory, etc., to find an optimal solution or a state of optimal operation. Constraints are often imposed that may limit the possible steps or actions that may be taken to achieve this optimum. The constraints may involve manpower, availability of space, raw materials or funds, machine capabilities, limitations imposed by time, governmental controls, behavior patterns, physical laws, etc.

The NLP presents a solid framework for problem formulation. When an NLP is explicitly stated, a solution

can generally be obtained. Many situations arise, however, where only numerical data or qualitative information is available and the problem functions cannot be determined explicitly, in which case the precise calculation of optimal points is impossible.

Of course the sagacious decision maker does not blindly accept the optimal point for the NLP as the best answer for his real-world problem. The optimal point is considered only as advisory, and the assumptions and the accuracy of the NLP are scrutinized before a decision is made. In many instances, however, NLP solutions have proven themselves in practice, in which cases the optimal point is accepted and implemented almost without question.

Although the transformation of a real-world problem into an explicit NLP is largely an art, there exists a theoretical framework that facilitates problem formulation. The theory clarifies the nature of those problem formulations which are most efficiently solved, and those which have no solution. This thesis is devoted to the development, presentation, and application of this theory.

1.2 The NLP in Standard Form

In a mathematical context we are given a function $f(x)$ which maps the variables $x \in E^n$ into scalar values, where E^n denotes n-dimensional Euclidean space. The function $f(x)$ is called an objective function, cost function, or performance measure because $f(x)$ is that entity to be extremized, that is, to be minimized or maximized with respect to x. Since maximization and minimization are equivalent problems (max f = min (-f)), only the latter is considered here. The point x, however, may not be chosen arbitrarily because of constraints imposed upon it. The constraints are classified as equality or inequality constraints and are defined in terms of x. Any x which satisfies all of the constraints is termed a feasible point and becomes a candidate for the optimal point. For a solvable problem there exists a non-empty set of feasible points. From this set of feasible points, a point x^*, at which f achieves its minimum value, is called an optimal point.

Stated mathematically, the NLP takes the following form: Find a point x^* to

$$\text{minimize } f(x) \tag{1-1}$$

subject to the constraints

$$h_i(x) = 0, \qquad i = 1, 2, \ldots, m_1 \qquad (1\text{-}2)$$

and

$$g_j(x) \geq 0, \qquad j = 1, 2, \ldots, m_2 \qquad (1\text{-}3)$$

where x is an n-dimensional vector. The h_i's, g_j's and f are nonlinear in general and are given (explicit) functions of class C^1. Also it is assumed that the n-dimensional gradient vectors $\nabla h_i \triangleq \partial h_i/\partial x$ and $\nabla g_j \triangleq \partial g_j/\partial x$ are nonzero at those x values which satisfy $h_i(x) = 0$ and $g_j(x) = 0$.

The problem as stated above seeks the constrained absolute minimum of f. In general practice, however, constrained local minima of f are found first; the absolute minimum of f is then found by comparing all constrained local minima.

1.3 Sequential Unconstrained Minimization

Basic to all NLP solution algorithms is a search in n-space for constrained extrema. Many constrained search techniques depend heavily upon the classical theory and methods for computing unconstrained extrema.

Unconstrained search techniques often incorporate a sequence of unidirectional searches in n-space, and are based on an iterative scheme of the form[†]

$$x^{k+1} = x^k + \sigma r^k \qquad (1\text{-}4)$$

where a parametric step σ is taken in a search direction r^k. The step σ is selected to obtain the minimum value of f in the r^k direction. The search direction r^k is usually chosen on the basis of information concerning f and its partial derivatives evaluated both at the point x^k and at previous iteration points. The influence of constraints imposed on x must be taken into consideration as they alter the way in which the constrained search direction is chosen. This may be a difficult task if the constraints are nonlinear. According to Davies and Swann [16][‡], there are two ways in which constraint information is incorporated in the establishment of search directions. They are as follows:

[†]The superscript notation is used generally to denote a change in value of a vector or parameter from one iteration to another. The subscript notation will be used to denote components of a vector or matrix.

[‡]References are associated with numbers in brackets and are listed alphabetically at the end of this thesis.

1. Function modification followed by unconstrained optimization.

   These methods seek to define a new function that has an unconstrained optimum at the same point as the optimum of the given constrained problem. Optimization of this new function will then define the required change in the search direction.

2. Direction modification without altering the function.

   Some of the methods in this category attempt to follow constraint boundaries while others try to rebound from them, so as to continue the search in the feasible region.

There are advantages and disadvantages to both types of methods which are explored in detail by McCormick [43], by Wolf [70], and in [16] where it is stated that no one method for constrained optimization is universally superior to all others and care must be exercised in choosing the appropriate one for a given situation.

The solution techniques developed in this work fall into the first category. The approach in these methods is that of transforming a given constrained minimization problem into a sequence of unconstrained minimization

problems as outlined by Fiacco and McCormick [23]. This transformation is accomplished by augmenting the original cost function with weighted terms of the problem functions, to define a new objective function whose minima are unconstrained in some domain of interest. By gradually removing the effect of the constraints in the new objective function by controlled parameter values, it is possible to generate a sequence of unconstrained problems that have solutions converging to a solution of the original constrained problem.

Thus, the sequential unconstrained method of minimizing the NLP is one associated with finding a sequence of problem solutions x^m such that

$$x^m = \min_{x \in R} A(f, h, g, w^m), \qquad m = 1, 2, \ldots \qquad (1\text{-}5)$$

where A is the new objective function formed by augmenting the cost function f with weighted terms of problem functions, h and g, w^m is the controlled weighting factor, and R is the domain of x. The functions of (1-5)† are constructed so that as $m \to \infty$, a convergent sequence of

---

†Hyphenated numbers in parenthesis refer to equations by (Chapter - equation number).

unconstrained minimizers $\{x^m\}$ approaches a constrained minimum point x^* of the NLP.

As an example of this type of method, consider an augmented function $A(x,w^m)$ which incorporates the equality constraints by Courant's quadratic penalty function [13], and the inequality constraints by Carroll's inverse penalty function [11].

Define

$$A(x, w^m) = f(x) + w^m \sum_{i=1}^{m_1} h_i^2(x) + \frac{1}{w^m} \sum_{j=1}^{m_2} \frac{1}{g_j(x)} \qquad (1\text{-}6)$$

To use this function we require a starting point x^0 which is in the strictly feasible region R^s associated only with the inequality constraints, i.e.,

$$x^0 \in R^s \triangleq \{x \mid g_j(x) > 0, \qquad j = 1, 2, \ldots, m_2\} \qquad (1\text{-}7)$$

Because the function $A(x,w^m)$ is to be minimized, and because $A(x,w^m)$ assumes the value of $+\infty$ at the boundary of R^s, it is guaranteed that no boundary of infeasible points will be generated with respect to the inequality constraints (1-3), i.e., all points generated will remain in R^s.

The method generally proceeds as follows. Select a monotonic sequence $\{w^m\}$, where

$$\{w^m\} = \{w^m \mid w^m > 0,\ w^{m+1} \geq w^m, \text{ and } w^m \to \infty \text{ as } m \to \infty\} \quad (1\text{-}8)$$

and compute a minimum point x^m, where

$$x^m = \min_{x \in R^s} A(x, w^m) \quad (1\text{-}9)$$

for $m = 1, 2, \ldots$. The desirable result is that

$$\lim_{m \to \infty} x^m = x^* \quad (1\text{-}10)$$

where x^* is a solution to the NLP. It is also well known that

$$\lim_{m \to \infty} w^m \sum_{i=1}^{m_1} h_i^2(x^m) = 0 \quad (1\text{-}11)$$

and that

$$\lim_{m \to \infty} \frac{1}{w^m} \sum_{j=1}^{m_2} \frac{1}{g_j(x^m)} = 0 \quad (1\text{-}12)$$

Thus

$$\lim_{m \to \infty} A(x^m, w^m) - f(x^*) = 0 \quad (1\text{-}13)$$

In effect, the influence of the constraints on the augmented function is relaxed and finally removed in the limit, and the augmented function converges to the same minimal value as the original cost function.

The function A(x, w) of (1-6) is called a mixed interior-exterior-point penalty function. If there were no equality constraints of the form (1-2), then (1-6) would have the form

$$f(x) + \frac{1}{w} \sum_{j=1}^{m_2} \frac{1}{g_j(x)} \qquad (1\text{-}14)$$

and would be called an interior-point penalty function, allowing only feasible points with respect to the inequality constraints. On the other hand, if there were no inequality constraints of the form (1-3), then (1-6) would have the form

$$f(x) + w \sum_{i=1}^{m_1} h_i^2(x) \qquad (1\text{-}15)$$

and would be called an exterior-point penalty function. Although it is possible to handle inequality constraints with an exterior penalty function method, as is done in this work, equality constraints can be incorporated only

through exterior-point methods. These methods allow points not feasible to the constraints, but force convergence to a feasible point in the limit as $m \to \infty$.

The development and use of sequential unconstrained methods has a long history. The mainstream of developments are reviewed in [23] and in references cited there. Important results of many of the previous and more recent methods can also be found in [1, 8, 20, 21, 22, 23, 24, 39, 42, 43, 53, 70, 72].

1.4 Motivation for Research

The usual penalty methods for solving nonlinear programming problems are subject to numerical instabilities, because the derivatives of the penalty functions or the penalty functions themselves increase without bound near the solution as computation proceeds, and are held in check by penalty weighting factors which, to insure convergence, must either increase to infinity or decrease to zero [60]. It is noted [34] that penalty function methods can become computationally ineffective if certain constraints tend to dominate the entire constraint set. Several authors [38, 41, 51, 52] confirm this fact by showing that ill-conditioning can occur in the Hessian matrix in

the final phases of a penalty function search, although a method to avoid this situation is given in [41]. When problems do arise in a penalty function search, they invariably occur during the final phase of the search [54]. It is in this phase, however, that relationships between Lagrange multipliers and certain penalty terms become viable, as shown in [7, 23, 54]. In past years the idea has arisen that terminal search instabilities might be circumvented by an approach involving a Lagrangian function containing additional penalty-like terms. Most of the work in this direction has been for problems treating equality constraints. Only recently has this type of method been extended to account for inequality constraints. The convergence proofs for these new methods often pertain to a class of functions which exhibit a certain property, usually convexity, and few algorithms are posed or operational that treat mixed equality and inequality constraints of the general NLP.

Because no penalty function is found to be superior for all types of problems, this author feels that the use of Lagrangian functions and the Kuhn-Tucker theorem [23] affords promising methods of solution to be explored. Besides being viewed directly as special cases of augmented

objective functions, Lagrangians are associated with every method for mathematical programming. There are extremely close connections between the Lagrangian of an NLP and the particular objective function being utilized to transform that particular problem into a sequence of unconstrained minimizations, in that the conditions of the Kuhn-Tucker theorem are directly involved in the associated Lagrangian.

As a practical aspect, Lagrange multipliers provide well-known sensitivity information [1, 72] concerning NLP solutions. While it is true that the results of a penalty function search can be used to estimate Lagrange multipliers, the estimates can be inaccurate because of the problems encountered in the final phases of the search. This author feels, therefore, that it is important that solution techniques for the NLP also solve the dual problem of constrained minimization--that of finding the associated Lagrange multipliers.

The conditions stated in this section have motivated the author to pursue the research presented in this thesis. In Chapter 2 pertinent background material is reviewed, and the theoretical structure is developed to place the multiplier algorithm of this thesis in perspective.

Sensitivity information is presented in Chapter 3, and a practical interpretation of Lagrange multipliers is given. In performing a search in n-space for a solution point to an NLP problem, consideration must be given to generating a search direction and performing a search in that direction for a minimum. Chapters 4 and 5 treat these topics in light of the Lagrangian presented here. Chapter 5 reviews the basic theory of unidirectional searches and presents a quadratic convergent search developed in this research. Chapters 1 through 5 provide the necessary development for the computer implemented algorithms presented in Chapter 6. Numerical results from a number of test problems and selected application problems are given in Chapters 7 and 8. The significant results of this research effort are reviewed in Chapter 9, along with suggestions for further research that are in line with current trends in mathematical programming.

CHAPTER TWO

CONSTRAINED MINIMIZATION BY MULTIPLIER METHODS

2.1 Introduction

In a particular situation one is usually interested in the global solutions to a mathematical programming problem. But in general, only theorems concerning local solutions can be readily proved, unless the problem has certain properties that can be exploited. One such property is that of convexity; in a convex programming problem every local solution can be proved to be a global solution. For most NLP's, however, minimization algorithms converge to various local minima depending upon initial starting conditions, and global minima are obtained by comparing the known local minima. A local minimum point is a point for which, in an open neighborhood about that point, there is no other point that both satisfies the constraints and gives a smaller value of the objective function. Points of global minima are those points of local minima which yield the smallest value of the objective function. Useful theorems can be developed stating necessary conditions and sufficient conditions that a point be a local minimum point to the general NLP, without relying upon special properties of a particular problem. This chapter is devoted to the development of such conditions with regard to an augmented Lagrangian function that is used in solving the NLP.

2.2 Review of Multiplier Techniques

Although Courant [13] formalized the use of penalty functions for nonlinear programming in 1943, it was the works of Fritz John [33] in 1948 and the fundamental theoretical paper [37] of Kuhn and Tucker in 1951 that placed nonlinear programming on a substantial theoretical foundation. The results of Kuhn and Tucker on necessary conditions and sufficient conditions characterized the solution of the nonlinear convex programming problem and gave an equivalence between this problem and the saddle point problem of the Lagrangian. It is known that the existence of a saddle point of the Lagrangian function is heavily dependent upon convexity properties of the underlying problem. In 1956 Arrow and Hurwicz [3] showed that convexity assumptions could be relaxed by using a modified Lagrangian approach. Differential gradient schemes by Arrow and Solow [4] in 1958 presented additional saddle point results in terms of a different modified Lagrangian function. King in 1966 [35], by relinquishing the convexity property and thus sacrificing the Kuhn-Tucker global properties, gave local necessary conditions for inequality constrained extreme values.

The method of multipliers for (equality constrained) nonlinear programming was independently introduced in 1968 by Hestenes [30, 31] and Powell [58]. They proposed a dual method of solution in which squares of the constraint functions are added to the Lagrangian. A series of unconstrained minimizations on the penalized Lagrangian is followed by a multiplier vector update according to a simple rule. Hestenes did not theoretically develop the idea beyond this stage, but Powell [58] showed that if second-order sufficiency conditions for optimality are satisfied, the algorithm should converge locally at a linear rate, without requiring the penalty factors to grow without bound. The main advantage of the algorithm lies in the latter property, since it provides a numerical stability that is not found in the usual penalty methods. Miele, et al. [45, 46, 48, 49, 50] have modified and improved the original methods of Hestenes and Powell and have obtained many computational results for the equality constraint case. Haarhoff and Buys [29] advanced a similar method in 1970. Their method was concerned mainly with the equality constraint case; their extension to accommodate inequality constraints was simply that of including slack-variable

terms to transform inequality constraints into equality constraints. Fletcher [25, 26] advanced yet another technique related to that of Hestenes and Powell. Instead of updating the multiplier vector after a sequence of minimizations, the multiplier was adjusted continuously as the minimization on the penalized Lagrangian was performed. Suggestions were given as to how this method could be extended to include inequality constraints. Termination and convergence properties were also given in [25]. Kort and Bertsekas [36] treat both equality and inequality constraints in their multiplier methods for convex programming.

In 1970 Rockafellar [59] introduced an augmented Lagrangian for inequality constrained convex programming problems for which an unconstrained saddle point corresponded to a solution of the convex programming problem. This approach was further studied by Rockafellar in a series of papers [60, 61, 62, 63].

Arrow, Gould, and Howe [2] in 1971 studied Rockafellar's augmented Lagrangian, included in a general class of augmented Lagrangians, for nonconvex programming problems and established local saddle point properties for this

class of problems. Related approaches to a different class of augmented Lagrangians has been treated by Mangasarian [44] as recently as 1973. The Lagrange multipliers of Rockafellar and Mangasarian are not constrained to be non-negative. Various Lagrangian functions cited above are listed in Appendix A.

In 1971 Pierre [54] suggested more widely applicable multiplier algorithms, accounting for both equality and inequality constraint functions. Pierre [55] introduced a particularly interesting augmented Lagrangian in August, 1973, with local convergence properties and numerical results from an operational algorithm. One of the purposes of this work is to study in detail the augmented Lagrangian of [55].

2.3 Theoretical Preliminaries

The theorems and proofs presented in this chapter for the general NLP require the objective function and the constraint functions to be continuous and to have a suitable degree of differentiability. Also the n-dimensional gradient vectors $\nabla h_i(x)$ and $\nabla g_j(x)$ are assumed to be nonzero at those points x, which satisfy $h_i(x) = 0$ and $g_j(x) = 0$.

With these requirements the following definitions are in order

Definition 1: [Continuity]

A numerical valued function f, defined on a set X in E^n, is said to be continuous at a point $x_0 \in X$ if, given any number $\varepsilon > 0$, there is a nonempty neighborhood N_{ε/x_0}, such that $|f(x) - f(x_0)| < \varepsilon$ for every point $x \in N_{\varepsilon/x_0} \bigcap X$. The function f is said to be continuous on X if it is continuous at each point of X.

Definition 2: [Class of differentiability]

A function f is said to be of class C^1 in an open region X in E^n if f is continuous in X and its first-order partial derivatives with respect to x, $\partial f/\partial x_i$, $i = 1, 2, \ldots, n$, are defined and continuous in X. It is said to be of class C^2 in X if it is of class C^1 and its second-order partial derivatives, $\partial^2 f/\partial x_i x_j$, $i,j = 1, 2, \ldots, n$, are defined and continuous in X.

Differentiability of higher order C^p, $p > 2$, can be defined similarly by use of Definition 2, but will not be required in this work. Well formulated multiplier

algorithms make effective use of first-order necessary conditions of optimality [55]. These conditions will now be developed.

2.4 First-Order Necessary Conditions

From the classical theory of minima and maxima, a first-order necessary condition is given by the following theorem. (For proof, see Appendix B)

Theorem 1: [First-order necessary condition for an unconstrained local minimum]
A necessary condition that a function f of class C^1 have an unconstrained local minimum at a point x^* is that

$$\nabla f(x^*) = \left.\frac{\partial f}{\partial x}\right|_{x = x^*} = 0 \qquad (2\text{-}1)$$

Consider again the general constrained problem restated here from Section 1.2. Find x^* and $f(x^*)$ where

$$f(x^*) = \min_{x} f(x) \qquad (2\text{-}2)$$

subject to the constraints

$$h_i(x) = 0, \qquad i = 1, 2, \ldots, m_1 \qquad (2\text{-}3)$$

and

$$g_j(x) \geq 0, \qquad j = 1, 2, \ldots, m_2 \qquad (2\text{-}4)$$

Let x^* be a point which satisfies (2-3) and (2-4), and let Δx denote an incremental change in x from x^*, i.e., $\Delta x = x - x^*$, where the magnitude $||\Delta x||$ is the Euclidean norm[†] $(\Delta x' \Delta x)^{\frac{1}{2}}$. We may then define a constrained local minimum.

Definition 3: [Constrained local minimum]

For continuous functions defined on a set X in E^n, f is said to have a constrained local minimum at x^* if there exists a real number $\varepsilon > 0$ such that

$$f(x^*) \leq f(x^* + \Delta x) \qquad (2\text{-}5)$$

for all

$$\{\Delta x \mid 0 < ||\Delta x|| < \varepsilon, \; h_i(x^* + \Delta x) = 0, \quad i = 1, 2, \ldots, m_1,$$
$$g_j(x^* + \Delta x) \geq 0, \quad j = 1, 2, \ldots, m_2\}$$

If (2-5) in the above definition can be replaced by

$$f(x^*) < f(x^* + \Delta x) \qquad (2\text{-}6)$$

then f is said to have a strict constrained local minimum at x^*.

[†]The symbol ′ is used to denote the transpose of a vector or matrix. Let C be an m x n matrix. Then C′ is the transpose of C and has dimension n x m where the $(i,j)^{th}$ entry of C′ is c_{ji}.

As a basis for understanding necessary conditions for constrained local minima, the following Lemma is of fundamental importance. (For proof, see Appendix B)

Lemma 1: [Farkas [23]. Theory of linear inequalities] Let $\{v_k\}$, $k = 0, 1, \ldots, q$, be a set of $n \times 1$ vectors. A necessary and sufficient condition that there exist nonnegative scalar values $\{s_k\}$, $k = 1, 2, \ldots, q$, such that

$$v_0 = \sum_{k=1}^{q} s_k v_k \tag{2-7}$$

is that for every vector z such that $z'v_k \geq 0$, $k = 1, 2, \ldots, q$, it follows that $z'v_0 \geq 0$.

The necessary conditions that apply to the NLP are associated with the Lagrangian L:

$$L(x, \xi, \alpha, \beta) = \xi f(x) - \sum_{i=1}^{m_1} \alpha_i h_i(x) - \sum_{j=1}^{m_2} \beta_j g_j(x) \tag{2-8}$$

where α and β are Lagrange multiplier vectors of dimension m_1 and m_2, respectively. The objective function $f(x)$ has an associated multiplier $\xi \geq 0$. The conditions that result for $\xi \geq 0$ are those of Fritz John [1, 33] which are

discussed and implemented by Pierre [56]. For the special case where $\xi = 1$, the resulting conditions are those of Kuhn and Tucker from [23] and are presented in the following development.

With $\xi = 1$, and using matrix notation, equation (2-8) is recast in the form

$$L \triangleq L(x, \alpha, \beta) = f - \alpha' h - \beta' g \tag{2-9}$$

From (2-9) the gradient of L with respect to x follows as

$$\nabla L = \nabla f - \nabla h' \alpha - \nabla g' \beta \tag{2-10}$$

Notation to be used in establishing the necessary conditions that must hold for x^* to be a local minimum is as follows: Let f^* denote $f(x^*)$, ∇f^* denote $\nabla f(x^*)$, etc., and let $\Delta x = x - x^*$ denote a sufficiently small change in x^*. Define the following sets in E^n.

$$S_1^* \triangleq \{\Delta x \mid \Delta x' \nabla g_j^* \geq 0, \quad j \in S_a; \ \Delta x' \nabla h_i^* = 0, \quad i = 1, 2, \ldots, m_1; \ \Delta x' \nabla f^* \geq 0\} \tag{2-11}$$

$$S_2^* \triangleq \{\Delta x \mid \Delta x' \nabla g_j^* \geq 0, \quad j \in S_a; \ \Delta x' \nabla h_i^* = 0, \quad i = 1, 2, \ldots, m_1; \ \Delta x' f^* < 0\} \tag{2-12}$$

and

$$S_3^* \triangleq \{\Delta x \mid \Delta x' \nabla g_j^* < 0, \text{ for at least one } j \in S_a;$$

$$\text{or } \Delta x' \nabla h_i^* \neq 0 \text{ for at least one } i\} \qquad (2\text{-}13)$$

where x^* is assumed to satisfy (2-3) and (2-4) and where

$$S_a \triangleq \{j \mid g_j(x^*) = 0\} \qquad (2\text{-}14)$$

Also to be used are the sets

$$S_b \triangleq \{j \mid g_j(x^*) > 0\} \qquad (2\text{-}15)$$

$$C_a \triangleq \{j \mid \beta_j > 0\} \qquad (2\text{-}16)$$

and

$$C_b \triangleq \{j \mid \beta_j = 0 \text{ and } g_j(x) \leq 0\} \qquad (2\text{-}17)$$

Clearly, $S_1^* \bigcap S_2^* \bigcap S_3^*$ equals the null set Φ, and any Δx in E^n belongs to one of the three sets S_1^*, S_2^*, or S_3^*. The set S_1^* of (2-11) is of limited interest because any Δx in this set does not show obvious reductions in $f(x)$; i.e., $f(x) - f(x^*) \cong \Delta x' \nabla f^* \geq 0$. Also, the set S_3^* of (2-13) is of limited interest because any Δx in this set does not give an x point that satisfies all of the constraints. The important set is therefore S_2^* of (2-12). A necessary condition that x^* be a constrained local minimum point is that all Δx's contained in S_2^* result in x's

which violate one or more of the constraints in (2-3) and (2-4). For the great majority of problems, S_2^* contains no Δx's which violate constraints, in which case a necessary condition that x^* be a constrained local minimum point is that $S_2^* = \Phi$. Unfortunately, there is a restricted class of problems for which the preceding statement does not hold. This restricted class of problems includes the class of problems for which the first-order constraint qualification condition [23] is not satisfied; these special problems must be treated separately. The following theorem, which is proved in Appendix B is given for the $S_2^* = \Phi$ case.

Theorem 2: [Existence of Generalized Lagrange Multipliers]

If i) x^* satisfies constraints (2-3) and (2-4),

ii) the functions f, $\{h_i\}_{i=1}^{m_1}$, $\{g_j\}_{j=1}^{m_2}$ are of class C^1,

iii) $S_2^* = \Phi$

then there exist vectors α^*, β^* such that (x^*, α^*, β^*) satisfies the following conditions:

$$h_i(x) = 0 \qquad i = 1, 2, \ldots, m_1 \qquad (2\text{-}19a)$$

$$g_j(x) \geq 0, \qquad j = 1, 2, \ldots, m_2 \qquad (2\text{-}19b)$$

$$\beta_j g_j(x) = 0, \qquad j = 1, 2, \ldots, m_2 \qquad (2\text{-}19c)$$

$$\beta_j \geq 0, \qquad j = 1, 2, \ldots, m_2 \qquad (2\text{-}19d)$$

$$\nabla L = 0 \qquad (2\text{-}19e)$$

The relations (2-19) are the Kuhn-Tucker relations that constitute a necessary condition that x^* be an optimal solution to the NLP. Note that assumption iii implies that the hypothesis of Lemma 1 are satisfied by the set A at x^* where

$$A = \{\nabla f^*, \{\nabla h_i^*, i = 1, 2, \ldots, m_1\}, \{-\nabla h_i^*, i = 1, 2, \ldots, m_1\}, \{\nabla g_j^*, j \in S_a\}\} \qquad (2\text{-}20)$$

To ensure the existence of finite multipliers when applying Theorem 2, one must be able to determine if $S_2^* = \Phi$. A useful criterion that can be used to test $S_2^* = \Phi$ is given in Theorem 3, which requires the following definition.

Definition 4: [Linear independence and dependence] The vectors $v_1, v_2, \ldots, v_q$ are said to be linearly independent if

$$c_1v_1 + c_2v_2 + \ldots + c_qv_q = 0 \qquad (2\text{-}21)$$

where $c_1, c_2, \ldots, c_q$ are constants, implies that $c_1 = c_2 = \ldots = c_q = 0$. Conversely, vectors $v_1, v_2, \ldots, v_q$ are said to be linearly dependent if and only if v_i can be expressed as a linear combination of v_j $(j = 1, 2, \ldots, q;\ j \neq i)$.

Theorem 3: [Sufficient condition that $S_2^* = \Phi$] A sufficient condition that $S_2^* = \Phi$ and also that the hypothesis of Lemma 1 is satisfied by the set A of (2-20) is that the gradients of the active constraints $\{\nabla h_i\}$, $i = 1, 2, \ldots, m_1$, and $\{\nabla g_j\}$, $j \in S_a$, are linearly independent at x^*. (For proof, see Appendix B)

Theorem 2 is a first-order characterization of local minima in that it involves first-order differentiability of the problem functions. It therefore does not take into account the curvature of the functions if they are

nonlinear. The curvature is measured by second-order derivatives of the problem functions.

Computational advantages can be obtained by augmenting the Lagrangian with additional terms. Consider the augmented Lagrangian L_a [55]:

$$L_a \triangleq L + w_1 \sum_{i=1}^{m_1} h_i^2 + w_2 \sum_{j \in C_a} g_j^2 + w_3 \sum_{j \in C_b} g_j^2 \qquad (2\text{-}22)$$

where L is the Lagrangian of (2-9), and w_1, w_2, and w_3 are penalty weights where $0 < \{w_1, w_2, w_3\} \leq \infty$. With slight loss in generality, we could let $w_2 = w_3$ and replace the corresponding summations in (2-22) by $w_2 \sum_{j \in C} g_j^2$ where $C = C_a \bigcup C_b$. Note also that $C_a \bigcap C_b = \Phi$ and therefore the last summation in (2-22) is equivalent to $w_3 \sum_{j \notin C_a} \frac{1}{2} g_j (g_j - |g_j|)$ which is a term suggested by Pierre [54].

The gradient of L_a with respect to x is

$$\nabla L_a = \nabla f - \sum_{i=1}^{m_1} (\alpha_i - 2w_1 h_i) \nabla h_i - \sum_{j \in C_a} (\beta_j - 2w_2 g_j) \nabla g_j$$

$$- \sum_{j \in C_b} (-2w_3 g_j) \nabla g_j \qquad (2\text{-}23)$$

Consider the direction of the vectors that sum to form $-\nabla L_a$. The first term, $-\nabla f$ points in the direction of greatest incremental decrease in f [47]. A typical equality constraint vector $(\alpha_i - 2w_1 h_i)\nabla h_i$ tends to force constraint satisfaction by pointing in the direction of decreasing $|h_i|$, i.e., if $h_i > 0$ and $h_i > \alpha_i/2w_1$, or if $h_i < 0$ and $h_i < \alpha_i/2w_1$. A stronger statement applies toward forcing constraint satisfaction for inequality constraint terms of the form $(\beta_j - 2w_2 g_j)\nabla g_j$. Since $\beta_j > 0$ for these terms, the gradient vector points in the direction of decreasing $|g_j|$ either if $g_j < 0$ or if $g_j > \beta_j/2w_2 > 0$. Finally, the last vector terms in (2-23) always point away from infeasible areas of constraint satisfaction. Since these terms, associated with the set C_b, apply only for inequality constraints where $\beta_j = 0$ and where $g_j < 0$, these vectors point in the direction of increasing g_j's.

With the augmented Lagrangian defined in (2-22), there is a direct relationship between the constrained local minima of the original problem and the unconstrained local minima of L_a. This can be shown from the following theorem given by Pierre [56].

Theorem 4: [Equivalence between NLP solutions and unconstrained minima of L_a]

Given (x^*, α^*, β^*) that satisfy (2-19), and given w_1, w_2, and w_3 that are sufficiently large such that $\{0 < w_i \leq \infty\}_{i=1}^{3}$ then L_a assumes a local minimum with respect to x at x^* if and only if x^* is a constrained local minimum of f.

Proof: Let (x^*, α^*, β^*) satisfy (2-19) with $\{0 < w_i \leq \infty\}_{i=1}^{3}$. The inactive constraints at x^* are those associated with $S_b = \{j | g_j(x^*) > 0\}$. Let ε be a sufficiently small positive number such that $g_j(x^* + \Delta x) > 0$ for all $j \in S_b$ and for any Δx which satisfies $0 < ||\Delta x|| < \varepsilon$. The terms in S_b therefore will not affect the formulation of the gradient of L_a at x^*.

First, assume that $f(x^*)$ is a constrained local minimum of f, and therefore $S_2^* = \Phi$. Consider $\Delta x \in S_3^*$. Suppose the i^{th} equality constraint is violated by Δx. The term $w_1 \sum_{i=1}^{m_1} h_i^2(x^* + \Delta x)$ in (2-22) will force $L_a(x^* + \Delta x) > L_a(x^*)$ for some $w_1 \leq \infty$. Also, suppose Δx violates the j^{th} inequality constraint for some $j \in S_a$, where $\beta_j \geq 0$. Then one of the terms

$w_2 \sum_{j \in C_a} g_j^2(x^* + \Delta x)$ or $w_3 \sum_{j \in C_b} g_j^2(x^* + \Delta x)$ in (2-22) will force $L_a(x^* + \Delta x) > L_a(x^*)$ for sufficiently large w_2 or w_3 where $w_2, w_3 \leq \infty$.

Thus, we are left to consider only those nonzero $\Delta x \in S_1^*$. For any Δx in this set we have

$$L_a(x^* + \Delta x) = f(x^* + \Delta x) \geq f(x^*) = L_a(x^*) \quad (2\text{-}24)$$

from which

$$L_a(x^* + \Delta x) \geq L_a(x^*) \quad (2\text{-}25)$$

Thus, L_a assumes an unconstrained local minimum at x^* if f has a local constrained minimum at x^*.

Now assume that $L_a(x^*)$ is an unconstrained local minimum of L_a. For the constrained problem, only those Δx are allowed for which $x^* + \Delta x$ satisfies (2-3) and (2-4). For any such Δx, with $||\Delta x|| < \varepsilon$, equation (2-22) reduces to

$$L_a(x^* + \Delta x) = f(x^* + \Delta x) - \sum_{j \in C_a} [\beta_j g_j(x^* + \Delta x) - w_2 g_j^2(x^* + \Delta x)] \quad (2\text{-}26)$$

Consider a typical term of the summation in (2-26)

$$\beta_j g_j(x^* + \Delta x) - w_2 g_j^2(x^* + \Delta x)$$

$$= [1 - (w_2/\beta_j)\, g_j\,(x^* + \Delta x)]\; \beta_j g_j(x^* + \Delta x) \qquad (2\text{-}27)$$

for $\beta_j > 0$ and $w_2 < \infty$, and for sufficiently small $\|\Delta x\|$, it is clear that the right-hand side of (2-27) is non-negative. Thus,

$$L_a(x^* + \Delta x) \leq f(x^* + \Delta x) \qquad (2\text{-}28)$$

By assumption $L_a(x^*)$ is an unconstrained local minimum of L_a, and thus,

$$f(x^*) = L_a(x^*) \leq L_a(x^* + \Delta x) \qquad (2\text{-}29)$$

and from (2-28) and (2-29), therefore,

$$f(x^*) \leq f(x^* + \Delta x) \qquad (2\text{-}30)$$

Thus, $f(x^*)$ is a constrained local minimum of f.

o

In the above proof if the inequalities in (2-25) and (2-30) can be replaced by strict inequalities, then in Theorem 4 it can be stated that L_a assumes a strict local minimum with respect to x at x^* if and only if x^* is a strict constrained local minimum of f.

2.5 Second-order Sufficient Conditions

Second-order sufficient conditions are developed to measure the curvature of the problem functions near a local minimum. The following developments require second-order differentiability of the problem functions. Before second-order sufficient conditions for constrained local minima are presented, the following definition and theorem are in order.

Definition 5: [Positive definitness]
A real symmetric n x n matrix M is said to be positive definite if for every nonzero vector $y \in E^n$, $y'My > 0$.

Theorem 5: [Sufficient conditions for unconstrained local minima]
Sufficient conditions that a function f of class C^2 have an unconstrained local minimum at x^* are that

$$\nabla f^* = 0 \tag{2-31}$$

and that, for all $\Delta x \neq 0$

$$\Delta x' \nabla^2 f^* \Delta x > 0 \tag{2-32}$$

If $\nabla^2 f$ is continuous at x^*, (2-32) is equivalent to the statement that

$\nabla^2 f^* = [\partial^2 f/\partial x_i \partial x_j]\big|_{x = x^*}$ is positive definite. (For proof, see Appendix B)

Sufficient conditions based on the Kuhn-Tucker conditions of Theorem 2 are given in the following theorem. (For a proof, see [23])

Theorem 6: [A set of sufficient conditions for a constrained local minimum]

If i) x^* satisfies (2-3) and (2-4),

ii) the functions f, $\{h_i\}_{i=1}^{m_1}$, $\{g_j\}_{j=1}^{m_2}$ are of class C^2,

iii) the gradients of the active constraints $\{\nabla h_i\}$, $i = 1, 2, \ldots, m_1$, and $\{\nabla g_j\}$, $j \in S_a$ are linearly independent at x^*,

then sufficient conditions that x^* be a constrained local minimum are that there exist vectors α^* and β^* such that (x^*, α^*, β^*) satisfies conditions (2-19) and that for every nonzero vector Δx satisfying $\Delta x' \nabla h_i^* = 0$, $i = 1, 2, \ldots, m_1$, and $\Delta x' \nabla g_j^* = 0$ for $j \in C_a$, and $\Delta x' \nabla g_j^* \geq 0$ for $j \in S_a - C_a$ at x^*, it follows that

$$\Delta x' \nabla^2 L \Delta x > 0 \qquad (2\text{-}33)$$

When second-order sufficient conditions are satisfied, Theorem 4 can be replaced by the following theorem.

Theorem 7: [Equivalence between NLP solutions and unconstrained minima of L_a; finite weights guaranteed] Given (x^*, α^*, β^*) that satisfy (2-19) and given sufficiently large but finite w_1, w_2, and w_3, L_a satisfies second-order sufficient conditions for an unconstrained local minimum at x^*, if and only if the NLP satisfies second-order sufficient conditions for a constrained local minimum at x^*.

Before proving Theorem 7, we will state and prove Lemma 2, suggested by Pierre and related to a theorem (Theorem 3 of [17]) on quadratic forms.

Let A be an n x n symmetric matrix of reals and let B be an n x m matrix of reals. Further, let $V \triangleq \{\Delta x | B'\Delta x \geq 0, \Delta x \neq 0\}$, and $\hat{V} \triangleq \{\Delta x | B'\Delta x < 0\}$. Also define $\delta(\hat{V})$:

$$\delta(\hat{V}) \triangleq \begin{cases} 1 \text{ if } \Delta x \in \hat{V} \\ \\ 0 \text{ if } \Delta x \in V \end{cases} \qquad (2\text{-}34)$$

In terms of the above notation we can now state and prove Lemma 2.

Lemma 2: [Quadratic forms and linear inequalities] Let V, $\hat{V}$, and $\delta(\hat{V})$ be defined as above. Then for $\Delta x \in V$, $\Delta x' A \Delta x > 0$ (respectively, < 0) if and only if there exists a number λ such that, for all $\Delta x \neq 0$,

$$F \triangleq \Delta x' A \nabla x + \lambda \Delta x' B B' \Delta x \delta(\hat{V}) > 0 \qquad (2\text{-}35)$$

(respectively, < 0).

Proof: First suppose $F > 0$ for all $\Delta x \neq 0$. For $\Delta x \in V$, $F = \Delta x' A \Delta x > 0$ which proves the sufficient part of the Lemma.

Next, suppose that $\Delta x' A \Delta x \geq 0$ for $\Delta x \in V$. For these Δx, it is obvious that for any finite λ, $F > 0$. For $\Delta x \in \hat{V}$ we use the following argument: Let

$$\theta(\Delta x) \triangleq -(\Delta x' A \Delta x / \Delta x' B B' \Delta x) \qquad (2\text{-}36)$$

which is a continuous function over the normalized set $\hat{V}_n$ defined by $\hat{V}_n \triangleq \{\Delta x \mid \Delta x' \Delta x = 1 \text{ and } \Delta x \in \hat{V}\}$. As Δx approaches the boundary of $\hat{V}_n$, i.e., $B' \Delta x \to 0$, the numerator $\Delta x' A \Delta x$ of $\theta(\Delta x)$ is positive, but the denominator approaches $0+$, and therefore $\theta(\Delta x)$ tends to $-\infty$. The preceding statement, with

the fact that $\theta(\Delta x)$ is continuous on $\hat{V}_n$, gives that $\theta(\Delta x)$ assumes a finite maximum value λ^* at some point in $\hat{V}_n$. Thus, for any $\lambda > \lambda^*$,

$$\lambda > -(\Delta x' A \Delta x / \Delta x' B B' \Delta x), \quad \Delta x \in \hat{V}_n \tag{2-37}$$

Rearranging (2-37) we obtain the desired result that

$$\Delta x' A \Delta x + \lambda \Delta x' B B' \Delta x > 0, \quad \Delta x \in \hat{V}_n \tag{2-38}$$

which, when combined with those $\Delta x \in V$, yields (2-35) for all $\Delta x \neq 0$, thus proving the necessary part of the Lemma.

o

We may now utilize the results of Lemma 2 in proving Theorem 7.

Proof: (Theorem 7) Under the assumed conditions of the theorem, it is clear that $\nabla L_a^* = \nabla L^* = 0$. It must be shown that, for all $\Delta x \neq 0$ and finite w_1, w_2, and w_3, $\Delta x' \nabla^2 L_a^* \Delta x > 0$ if and only if $\Delta x' \nabla^2 L^* \Delta x > 0$ for $\Delta x \neq 0$ and Δx such that $\Delta x' \nabla h_i^* = 0$, $i = 1, 2, \ldots, m_1$, and $\Delta x' \nabla g_j^* = 0$, $j \in C_a$, and $\Delta x' \nabla g_j^* \geq 0$, $j \in S_a - C_a$ at x^*. Note that $\Delta x' \nabla h_i^* = 0$ is equivalent to the requirement

that both $\Delta x' \nabla h_i^* \geq 0$ and $-\Delta x' \nabla h_i^* \geq 0$. The proof follows directly from Lemma 2 when the following identities are made:

$$F \equiv \tfrac{1}{2}\Delta x' \nabla^2 L_a^* \Delta x \tag{2-39a}$$

$$A \equiv \tfrac{1}{2}\nabla^2 L^* \tag{2-39b}$$

$$\lambda \geq \max \{w_1, w_2, w_3\} \tag{2-39c}$$

$$\text{Columns of } B \equiv \begin{cases} \sqrt{w_1/\lambda}\ \nabla h_i^* & i = 1, 2, \ldots, m_1 \\ -\sqrt{w_1/\lambda}\ \nabla h_i^* & i = 1, 2, \ldots, m_1 \\ \sqrt{w_2/\lambda}\ \nabla g_j^* & j \in C_a \\ -\sqrt{w_2/\lambda}\ \nabla g_j^* & j \in C_a \\ \sqrt{w_3/\lambda}\ \nabla g_j^* & j \in S_a - C_a \end{cases} \tag{2-39d}$$

With the necessary and sufficient conditions of local minima now established, we may present the multiplier algorithm that operates on the augmented Lagrangian of (2-22) to find these minima.

2.6 The Multiplier Algorithm

When L_a is formed using (x^*, α^*, β^*) which satisfy necessary and sufficient conditions for the constrained

problem, x^* is a constrained local minimum of f if and only if L_a assumes a local minimum with respect to x at x^*. In section 1.2 the problem functions are assumed to be of class C^1. In the posed multiplier algorithm, the DFP conjugate search direction method is used. This method requires the problem functions to be of class C^1: the first derivatives of the problem functions must be explicitly available for operation. However, the method is more efficient in general if the problem functions are of class C^2. Then, if second-order sufficient conditions are satisfied, and if $S_a - C_b$ is empty, each matrix of a sequence generated by the DFP method remains positive definite and the sequence tends to the inverse of $\nabla^2 L_a$ in the vicinity of a local minimum, thus ensuring that a minimum is reached for finite penalty weights $\{w_i\}_{i=1}^{3}$. Even if second-order sufficient conditions are not satisfied at an x^*, which is a local minimum, the algorithm is constructed to give a close approximation to x^* by effectively minimizing L_a with respect to x such that L_a is forced to satisfy the first-order conditions given in (2-19); however, in this case the weights $\{w_i\}_{i=1}^{3}$ may be required to approach infinity to attain convergence.

The algorithm consists of alternating minimization and update phases. During the minimization phase the multipliers and penalty weights are held fixed, and a quadratically convergent search is employed to find x^m that minimizes $L_a(x)$ such that x^m satisfies

$$\nabla L_a(x^m) = 0 \qquad (2\text{-}40)$$

Because of computational roundoff errors and because nonlinear functions are involved, x^m generally satisfies (2-40) only in an approximate sense: the termination of the minimization phase must be based on a set of pragmatic conditions. One condition of the set is as follows: Let ε be a small number and replace (2-40) by

$$\nabla L_a(x^m)' \nabla L_a(x^m) < \varepsilon \qquad (2\text{-}41)$$

At the completion of a minimization phase the multipliers are updated in an attempt to satisfy (2-19). Thus, the multipliers are updated to obtain

$$\underset{\text{(new)}}{\nabla L(x^m)} = \underset{\text{(old)}}{L_a(x^m)} \tilde{=} 0 \qquad (2\text{-}42)$$

The multiplier updating rules that result in (2-42) follow by comparing (2-10) and (2-23):

Equality constraints:

$$\underset{\text{(new)}}{\alpha_i} = \underset{\text{(old)}}{\alpha_i} - 2w_1 h_i(x^m), \qquad i = 1, 2, \ldots, m_1$$

Inequality constraints:

$$\text{for } j \in C_a,$$

$$\beta_{j\,(\text{new})} = \begin{cases} 0, & \text{if } \beta_{j\,(\text{old})} - 2w_2 g_j(x^m) \leq 0 \\ \beta_{j\,(\text{old})} - 2w_2 g_j(x^m), & \text{otherwise} \end{cases} \tag{2-44}$$

$$\text{for } j \in C_b,$$

$$\beta_{j\,(\text{new})} = \begin{cases} 0, & \text{if } g_j(x^m) \geq 0 \\ -2w_3 g_j(x^m), & \text{if } g_j(x^m) < 0 \end{cases} \tag{2-45}$$

The $\{w_i\}_{i=1}^{3}$ values used in (2-43), (2-44) and (2-45) are those values which were used during the preceding minimization phase. After the new β_j's are generated, the indices belonging to the sets C_a and C_b are updated, and are held fixed for the next minimization phase.

Equations (2-44) and (2-45) prescribe a constraint basis for the inequality constraints during a minimization phase. Throughout such a minimization phase, those inequalities having positive multipliers are in a 'locked on' status [56] and are treated the same as equality constraints. Inequality constraints with associated zero-valued multipliers are held from excessive constraint violation, as in penalty function methods, by the exterior

penalty terms associated with w_3. Upon termination of a given minimization phase, the equality constraint multipliers are updated by (2-42), and the inequality constraint multipliers are updated by (2-44) and (2-45) in order to prescribe a new inequality constraint basis for the next minimization phase. If a β_j multiplier from the previous basis tends negative, as in (2-44), the implication is that f will tend to decrease if x is free to move away from the corresponding constraint boundary with $g_j(x)$ increasing, thus relinquishing its locked-on status. The constraint then becomes a free constraint. A free constraint is added to the new constraint basis if the constraint is violated at the end of a minimization phase. Thus, with the violated constraint having a value less than zero, the corresponding multiplier is assigned a positive value, as in (2-45), and the constraint maintains a locked-on status throughout the next minimization phase.

The locked-on feature is especially important when x^m is in the vicinity of a constrained minimum at x^*. If the problem functions are of class C^2 and if strict complimentary slackness holds near x^*, i.e., if $S_a = C_a$, all active inequality constraints (characterized by $g_j(x^*) = 0$) have associated positive multipliers

($\beta_j > 0$), and the matrix of second partial derivatives $\nabla^2 L_a^*$ is continuous. Continuity of second derivatives is a property that facilitates the unconstrained minimization of a function by the use of quadratic convergent techniques.

For most problems the w_i's could be assigned initially and held fixed throughout the algorithm. But because the terms added to the Lagrangian to form L_a are penalty terms, it is more logical to parallel to some degree the procedure used with penalty function methods. Thus, the w_i's are assigned some relatively small positive values and are updated, after the multipliers are updated, by a factor $\gamma \geq 1$. This procedure continues until given upper bounds w_{imax}'s are reached. The penalty weight update rule then is

$$w_{i_{(new)}} = \begin{cases} w_{imax}, & \text{if } \gamma w_{i_{(old)}} \geq w_{imax} \\ \gamma w_{i_{(old)}}, & \text{otherwise} \end{cases} \qquad (2\text{-}46)$$

It is important to note that the initial w_i's must be large enough to prevent a constraint breakthrough from occurring [53].

At x^m, after the multipliers and weights have been updated, the search for a minimum of L_a is conducted initially in the $-\nabla L_a(x^m)$ direction, where

$$\nabla L_a(x^m)\Big|_{(\alpha,\ \beta,\ w)\ =\ (\alpha,\ \beta,\ w)_{(new)}} =$$

$$\left[2w_1 \sum_{i=1}^{m_1} h_i \nabla h_i + 2w_2 \sum_{j \in C_a} g_j \nabla g_j\right]\Bigg|_{x\ =\ x^m} \qquad (2\text{-}47)$$

The search direction of (2-47) is one that tends to move x towards the constraint basis, and in general the corresponding constraints will be more closely satisfied at the end of the next minimization phase.

Pierre [56] shows that this algorithm generates the constrained minimum of f after a finite number of unconstrained quadratic-convergent minimizations under the following conditions:

1) f is linear in x
2) $\{h_i\}_{i=1}^{m_1}$, $\{g_j\}_{j=1}^{m_2}$ are affine linear of the form $b + a'x$ where b and the components of the vector a are real constants
3) a unique and bounded x^* exists

4) $g_j(x^*) > 0$ if $\beta_j^* = 0$ (strict complementary slackness)

Pierre's justification of the above is as follows. At some finite point in the sequence of alternating minimization and update phases, the constraint basis C_a contains only those constraints that bound x^* and the set C_b is empty. A quadratic-convergent minimization is then performed where the last summation in (2-22) remains at zero. The resulting x^m satisfies $\nabla L_a(x^m) = 0$, apart from roundoff errors, because L_a is quadratic in the domain leading from the last x^m to the present x^m. The multipliers obtained at this stage are optimum because the gradients of the linear problem functions are constant vectors. Thus, the new $\nabla L = 0$, and the new ∇L_a is that given in (2-47), with the terms associated with w_1 and w_2 quadratic in x^m. One final quadratic-convergent minimization at this stage yields the true minimum within numerical accuracy limitations.

2.7 Summary

In this chapter we have presented an unconstrained function L_a, that is formulated by augmenting the Lagrangian with weighted problem functions of the constrained

NLP. First-order necessary conditions (Theorem 4) and second-order sufficient conditions (Theorem 7) have been established showing the equivalence between NLP solutions and the unconstrained minima of L_a. The Kuhn-Tucker relations (Theorem 2) which are conditions for the existence of generalized Lagrange multipliers, are the first-order conditions of optimality upon which all multiplier algorithms are formulated.

A multiplier algorithm is presented giving the general procedure for updating the multipliers during the minimization process such that, as the sequence of minimizers $\{x^m\}$ converges to a local minimum x^*, the multipliers also converge to α^* and β^*. Although the algorithm is based and operates on first-order conditions, if the problem functions are of class C^2, such that second-order conditions are satisfied, then the algorithm converges locally without requiring the penalty weights to increase without bound. This provides a numerical stability not found in the usual penalty methods.

In order for the algorithm to work, we must have a means of generating search directions in n-space and a means of searching along this direction for an

unconstrained minimum. The theory and algorithms for these procedures will be discussed in Chapters 4 and 5, but first a practical interpretation of the Lagrange multipliers, which are obtained at the minima of the NLP, will be given in the discussion of sensitivity in Chapter 3.

CHAPTER THREE

SENSITIVITY

3.1 Introduction

A practical question that often arises in solving either a mathematical or a real-world problem is how much the optimum changes when changes are made in the constraints or when changes are made in the optimal parameter values or system elements. Sensitivity analysis allows one to examine the variations in an objective or cost function when such changes are made. Physical systems cannot be constructed or operated exactly as in theory because of the inherent limitations on man's capabilities and limitations caused by a real-world environment. But these systems may be constructed and operated satisfactorily if the elements that influence system operation maintain certain tolerance limits. Tolerance specifications are an important part of a complete solution to a practical problem, and to specify them realistically one must know how changes in the system elements influence the characteristics of the system or problem under consideration. Sensitivity analysis provides the means of obtaining such information.

3.2 Microscopic Sensitivity [53]

The fractional change in a given system characteristic, which is a function of system parameters, can be

estimated on the basis of fractional changes in the parameters. When these fractional changes are required to remain within specified bounded regions or tolerance limits, worst-case sensitivities of system characteristics may be estimated by directly incorporating tolerance limits in the performance measure and constraint equations. For example, system parameters are set to tolerance extremes which are most likely to cause constraint violation and are least favorable to optimal performance. The performance measure is then optimized under these worst-case conditions. A similar analysis could be performed where the tolerance limits are at the opposite extremes and the performance measure is optimized under best-case conditions yielding best-case sensitivities. Based on the above analysis, a designer can choose design-center values for system parameters. Microscopic sensitivity can then be analyzed.

Microscopic (incremental) sensitivity measures are usually based on a truncated Taylor series, since the incremental sensitivity of a system characteristic is applicable to small regions within the tolerance extremes about some design-center. Let us consider the variation of the performance measure $f(x^*)$ where x^* is the design-center

value of the system parameter vector x. If we assume that f(x) is of class C^2, we can expand f(x) about x^* in the truncated Taylor series

$$f(x) \tilde{=} f(x^*) + (x-x^*)' \nabla f(x^*) + \tfrac{1}{2}(x-x^*)' \nabla^2 f(x^*)(x-x^*) \qquad (3\text{-}1)$$

Define Γ and Λ as follows [53]:

$$\Gamma \triangleq \left[\frac{x_1 - x_1^*}{x_1^*}, \frac{x_2 - x_2^*}{x_2^*}, \ldots, \frac{x_n - x_n^*}{x_n^*} \right] \qquad (3\text{-}2)$$

and

$$\Lambda \triangleq \begin{bmatrix} x_1^* & 0 & \cdots & 0 \\ 0 & x_2^* & & \\ \vdots & & \ddots & \\ 0 & & & x_n^* \end{bmatrix} \qquad (3\text{-}3)$$

With these identities, (3-1) can be rearranged to obtain

$$\frac{f(x) - f(x^*)}{f(x^*)} \tilde{=} \Gamma' \frac{\Lambda \nabla f(x^*)}{f(x^*)} + \Gamma' \frac{\Lambda \nabla^2 f(x^*) \Lambda}{2 f(x^*)} = \Gamma' S_x^f + \Gamma' S q_x^f \Gamma \qquad (3\text{-}4)$$

where the column sensitivity matrix S_x^f is defined by

$$S_x^f \triangleq \frac{\Lambda \nabla f(x^*)}{f(x^*)} \qquad (3\text{-}5)$$

and the square sensitivity matrix Sq_x^f is defined by

$$Sq_x^f \triangleq \frac{\Lambda \nabla^2 f(x^*) \Lambda}{2f(x^*)} \tag{3-6}$$

Both $f(x)$ and x^* are assumed known so that S_x^f and Sq_x^f can be evaluated. If x^* happens to be a local minimum (or maximum) $S_x^f = 0$. Equation (3-4) yields the fractional change in $f(x)$ which results in the fractional changes in x as specified in the Γ matrix. If $f(x^*) = 0$ or if any $x_i^* = 0$, however, equations (3-4), (3-5), and (3-6) should not be used--in such cases equation (3-1) should be considered directly with sensitivity measured in magnitude of change rather than in fractional change.

3.3 Macroscopic Sensitivity

Macroscopic sensitivity of a system characteristic is sensitivity with respect to large parameter changes. For example, if a range of values of a particular system parameter gives rise to a range of values for a system characteristic of interest, a design-center value may be selected so that the system characteristic is least sensitive to changes in the parameter. In many applications, tolerance limits ℓ_i are known for design-center values x_i^* such that, in matrix form

$$-L \leq \Gamma \leq L \qquad (3\text{-}7)$$

where $L = [\ell_1, \ell_2, \ldots, \ell_n]'$.

The question that often arises then is: "What is the maximum of the absolute value of $[f(x) - f(x^*)]/f(x^*)$ in (3-4) with respect to allowed variations of Γ as in (3-7)?" This question is posed such that it can be put into the format of the NLP and can be answered by solving the NLP with search techniques such as presented in this thesis. In general, the application of search techniques results in the accumulation of large amounts of macroscopic information.

3.4 Sensitivity and Lagrange Multipliers

Sensitivity in the broad sense of the term may be applied to small changes in the problem functions as compared to changes in the parameters for fixed functions. The Lagrange multipliers that satisfy the Kuhn-Tucker conditions are frequently termed shadow prices or attribute costs, in that they have the property of giving an attributed cost to a constraint. Consider the problem that often arises:

$$\max P(x) \qquad (3\text{-}8)$$

subject to

$$s_i(x) = a_i \qquad i = 1, 2, \ldots, m_1 \qquad (3\text{-}9)$$

and

$$t_j(x) \leq b_j \qquad j = 1, 2, \ldots, m_2 \qquad (3\text{-}10)$$

As a general example, $P(x)$ may be viewed as a profit function for some production process; the constraints of (3-9) may be machine operation and manpower costs needed to produce a line of products from given resources while the constraints of (3-10) may represent the availability or limitation on the total amount of the j^{th} resource. By some investment or reduction in resources, or by effecting a change in personnel or production policy, the manager confronted with the above problem could obtain a displacement d_k, $k = 1, 2, \ldots, m_1 + m_2$, in the constraints (3-9) and (3-10). Assuming that by shifting the right-hand side of the constraints by a small amount d_k, and assuming that there is an associated savings or cost π_k per unit of change, an investment cost of savings $\pi_k d_k$ results. The following question then arises: "Is the investment or policy change profitable or not?"

To answer this question let us first reformulate the management problem into the format of the NLP so that we may directly use the Kuhn-Tucker conditions developed in Section 2.4. Define $f(x) = -P(x)$, $h_i(x) = s_i(x) - a_i$, and $g_j(x) = b_j - t_j(x)$. Viewing the displacements, d_k with these identities the problem becomes

$$\min f(x) \tag{3-11}$$

subject to

$$h_i(x) - d_i = 0, \qquad i = 1, 2, \ldots, m_1 \tag{3-12}$$

and

$$g_j(x) - d_{j+m_1} \geq 0, \qquad j = 1, 2, \ldots, m_2 \tag{3-13}$$

Of primary interest is the local behavior of the minimum value of the objective function as the d_i's are varied. The Kuhn-Tucker conditions for this problem are

$$h_i(x) - d_i = 0, \qquad i = 1, 2, \ldots, m_1 \tag{3-14}$$

$$g_j(x) - d_{j+m_1} \geq 0, \qquad j = 1, 2, \ldots, m_2 \tag{3-15}$$

$$\beta_j(g_j(x) - d_{j+m_1}) = 0, \qquad j = 1, 2, \ldots, m_2 \tag{3-16}$$

$$\beta_j \geq 0, \qquad j = 1, 2, \ldots, m_2 \tag{3-17}$$

$$\frac{\partial f}{\partial x_k} = \sum_{i=1}^{m_1} \alpha_i \frac{\partial h_j}{\partial x_k} + \sum_{j=1}^{m_2} \beta_j \frac{\partial g_j}{\partial x_k}, \quad k = 1, 2, \ldots, n \tag{3-18}$$

Let $\{\nabla h_i\}$ for $i = 1, 2, \ldots, m_1$ and $\{\nabla g_j\}$ for all $j \in S_a$ be linearly independent so that (x^*, α^*, β^*) is the minimum solution satisfying the Kuhn-Tucker conditions at d_i's $= 0$, and assume that the d_i's are small enough so that the functions $x(d)$, $\alpha(d)$, and $\beta(d)$ are of class C^1, where $(x(d), \alpha(d), \beta(d))$ also satisfies the Kuhn-Tucker conditions as the d_i's are varied.

Using the differential chain rule

$$\frac{\partial f(x(d))}{\partial d_k} = \sum_{j=1}^{n} \left(\sum_{i=1}^{m_1} \alpha_i \frac{\partial h_i}{\partial x_j} + \sum_{i=1}^{m_2} \beta_i \frac{\partial g_j}{\partial x_j} \right) \frac{\partial x_j}{\partial d_k} \qquad (3\text{-}20)$$

All equality constraints in equation (3-14) must be satisfied by $x(d)$ in a neighborhood of x^*, i.e.,

$$h_i(x(d)) = d_i \qquad (3\text{-}21)$$

Clearly, from (3-21),

$$\frac{\partial h_i}{\partial d_k} = \delta_{i,k} \qquad (3\text{-}21)$$

where

$$\delta_{i,k} = \begin{cases} 0, & i \neq k \\ 1, & i = k \end{cases} \qquad (3\text{-}23)$$

Using the chain rule on (3-22),

$$\frac{\partial h_i}{\partial d_k} = \sum_{j=1}^{n} \frac{\partial h_i}{\partial x_j} \frac{\partial x_j}{\partial d_k} = \delta_{i,k} \tag{3-24}$$

Suppose that the i^{th} constraint in (3-15) is active at $d = 0$. By assumption, it will remain active at $x(d)$ in a neighborhood of x^* so that

$$g_i(x(d)) = d_{i+m_1} \tag{3-25}$$

and this constraint can be treated in the same way as the equality constraints. Thus,

$$\frac{\partial g_i}{\partial d_k} = \sum_{j=1}^{n} \frac{\partial g_i}{\partial x_j} \frac{\partial x_j}{\partial d_k} = \delta_{k,i+m_1} \tag{3-26}$$

Now suppose the i^{th} constraint in (3-15) is inactive at $d = 0$. By assumption, if the displacement d_i's are small, g_i will remain inactive at $x(d)$ in a neighborhood of x^* so that

$$g_i(x(d)) - d_{i+m_1} > 0 \tag{3-27}$$

Consequently, by (3-16),

$$\beta_i = 0 \tag{3-28}$$

for all inactive constraints.

Substituting (3-24), (3-26), and (3-28) into equation (3-20), we obtain

$$\frac{\partial f(x(d))}{\partial d_k} = \sum_{i=1}^{m_1} \alpha_i \delta_{i,k} + \sum_{i=1}^{m_2} \beta_i \delta_{k,i+m_1} \qquad (3\text{-}29)$$

which reduces to the value of the mulitplier associated with the index value of k. Thus,

$$\frac{\partial f(x(d))}{\partial d_k} = \begin{cases} \alpha_k, & k = 1, 2, \ldots, m_1 \\ \beta_{k-m_1}, & k = m_1+1, m_1+2, \ldots, m_1+m_2 \end{cases} \qquad (3\text{-}30)$$

To first-order terms, (3-29) is used to find the incremental change, $\Delta f(x)$, in $f(x)$, where,

$$\Delta f(x) = \sum_{i=1}^{m_1} \alpha_i^* d_i + \sum_{j=1}^{m_2} \beta_j^* d_{j+m_1} \qquad (3\text{-}31)$$

Thus, in regard to the management question, we see from (3-31) that when the d_i's are small, the answer is obtained by comparing the π_k with the corresponding α_k^* and $\beta_{k-m_1}^*$ values. The practical implication of this result is that we can easily obtain an indication of how the optimal value of the objective function changes as the resource or policy changes. In using a multiplier algorithm, the multipliers for the constraint functions are readily available at the solution of the problem, and the indication of favorable and unfavorable investments or policy changes are obtained without solving numerous NLP problems.

It must be noted, however, that large changes based on incremental indications may lead to results quite contrary to those expected. Incremental changes assume that a fixed set of constraints hold in a small neighborhood of the optimal solution. Large changes may violate this assumption, and the constraint basis may vary with the displacements. Thus, before large scale changes based on incremental indications are implemented, it is wise to solve the NLP under the new conditions.

Summary

The sensitivity analysis presented in this chapter gives a method for finding how the objective or cost-function changes when changes are made in the constraints or when changes are made in the optimal parameter values. Of most significance to the general NLP is the sensitivity information contained in the Lagrange multipliers that satisfy the Kuhn-Tucker conditions. In using a multiplier algorithm for solving a real-world NLP, the multipliers are readily available at the solution. These multipliers have the property of giving an attributed cost to the problem constraints. And with proper interpretation of the multipliers, indications of favorable investments or policy

changes are obtained without solving numerous NLP problems. Several examples of practical application of the NLP and interpretation of the resulting multipliers are given in Chapter 8.

CHAPTER FOUR

GENERATING SEARCH DIRECTIONS

4.1 Introduction

For general comparison purposes sequential search techniques can be divided into two categories: 1) techniques that utilize derivatives of the problem functions, and 2) techniques that do not require derivatives of the problem functions. In general, the best sequential methods which utilize the gradient are more efficient than those that do not [53]. The search techniques of this thesis utilize the gradient in generating the search direction; the techniques are applicable to the case where the gradient is defined analytically at each point, as opposed to the case where derivatives are approximated by differences in the values of the performance measure in the neighborhood of each search point.

Sequential unconstrained minimization procedures locate the minimum x^* as the limit of the sequence $\{x^k\}$, $k = 0, 1, 2, \ldots$, where x^0 is an initial estimate of the minimum, and for each $k \geq 0$, x^{k+1} is the minimum of the performance measure with respect to variations along the line through x^k in some specified direction r^k.

A desirable property for an iterative search method to possess is that of quadratic convergence; meaning that

for quadratic functions, the minimum will be located exactly, apart from roundoff errors, within a finite number of iterations [28]. In general, the unconstrained performance measure f, resulting from the transformation of the constrained NLP into an unconstrained problem, is not quadratic, but in the neighborhood of the minimum x^* it may be expanded in a Taylor series of the form

$$f(x) = f(x^*) + \Delta x' \nabla f(x^*) + \tfrac{1}{2}\Delta x' \nabla^2 f(x^*) \Delta x + \text{higher order terms} \qquad (4\text{-}1)$$

where $\Delta x = x - x^*$, $\nabla f(x^*)$ is the gradient of f evaluated at x^* and is equal to zero, and where $\nabla^2 f(x^*)$, the Hessian matrix of second partials evaluated at x^*, is symmetric and positive definite. For general functions, as the minimum is approached, the function appears more closely quadratic, and the rate of convergence generally increases. Furthermore, in regions remote from the minimum, such methods often account for the curvature of the function and are thereby able to deal with complex situations which are created, for example, when active constraints are incorporated with penalty functions. Information about the local behavior of the function is most efficiently used when iterative search methods calculate each new direction

of search as part of the iteration cycle. In the remainder of this chapter, we investigate the properties of gradient methods used in generating search directions.

4.2 Gradient Techniques

Common to all gradient techniques is the iteration equation (1-4) given here in the form

$$x^{k+1} = x^k - \alpha Hg\Big|_{x = x^k} \tag{4-2}$$

where x^{k+1} is the new value of x obtained from a variation α along the direction $Hg\Big|_{x = x_k}$ through the old point x^k. H is an n x n matrix, g is the gradient† of f, and α is a real number. Gradient methods differ in the way α and H are selected.

In the method of steepest descent H in (4-2) is set equal to the identity matrix I. At the minimum point of a given search direction, the next search direction is set to the negative gradient at that point, which is the direction of greatest incremental decrease of f(x).

---

†In this chapter, g will be used as standard notation for the gradient of f, ∇f, and is not to be confused with the constraint set (1-3).

Evaluating f(x) at the point defined in (4-2) with $H = I$, we have

$$f(x^{k+1}) = f(x^k - \alpha g(x^k)) \triangleq y(\alpha) \qquad (4\text{-}3)$$

which forms a parametric equation in α. The gradient of (4-3) with respect to α is

$$\frac{\partial y}{\partial \alpha} = -g(x^{k+1})' g(x^k) \qquad (4\text{-}4)$$

which is always negative at $\alpha = 0$, since $g(x^k)'g(x^k) > 0$. Therefore, a decrease in f is guaranteed at every step if α is sufficiently small. If α is too small, a decrease is guaranteed, but convergence may be slow, and if α is too large f may actually increase. The greatest decrease occurs from (4-4) at the value of α where $\partial f/\partial \alpha = 0$. This optimum value of α is found by solving (4-3) with a unidirectional search as outlined in Chapter 5.

Convergence, although almost certain with this method, is slow and relatively inefficient when nonlinear problem functions are involved, as oscillatory search directions are often generated. Curry [14] also notes that the convergence rate of this method is affected by scaling of the x_i variables. In E^n, if the function being minimized is spherical, the minimum can be found in one

iteration, because the negative gradient is normal to the spherical surface and points toward the center. If a scale change for any coordinate is made, however, the surface becomes elipsoidal, the negative gradient generally no longer points to the center, and minimization requires more than one iteration, resulting in an increased convergence rate. These problems can be reduced to some extent by the use of conjugate gradient search directions discussed in Section 4.4.

4.3 Newton Search

Suppose that in the vicinity of the current search point x^k, $f(x^{k+1})$ can be adequately represented by the truncated Taylor series

$$f(x^{k+1}) = f(x^k) + \Delta x' g(x^k) + \tfrac{1}{2}\Delta x' A^k \Delta x \qquad (4\text{-}5)$$

where $\Delta x = x^{k+1} - x^k$, and A^k is the n x n matrix of second partial derivatives of f evaluated at x^k. The Δx that minimize (4-5) must satisfy

$$A^k \Delta x = -g(x^k) \qquad (4\text{-}6)$$

Equation (4-6) is in the form of (4-2); where $H = (A^k)^{-1}$ and $\alpha = 1$. If the function to be minimized is quadratic, equation (4-6) gives the minimum in one iteration, provided

that the inverse of the matrix of second partial derivatives is available.

The Newton search is a gradient method that employs the above approach. It has the property of quadratic convergence, but requires the evaluation of second derivatives, and a matrix inversion per iteration. When x^k is near a local minimum of a nonquadratic function, A^k is nearly constant, and convergence is fast: but if x^k is not in the vicinity of a local minimum, such that the higher-order terms in (4-1) tend to dominate, the Newton method may actually diverge. Another problem with this method occurs when A^k is a singular matrix, such that the determinate $|A^k| = 0$. Under these circumstances no inverse of A^k exists. Even when x^k is near a minimum and $|A^k| \neq 0$, it may be close to zero so that A^k is ill-conditioned, leading to erroneous results.

4.4. Conjugate Gradient Techniques

The conjugate gradient method was developed in 1952 by Hestenes and Stiefel [32] to solve a set of simultaneous linear equations having a symmetric positive-definite matrix of coefficients. The equivalence of that problem

and the minimization of a quadratic function can be seen from equation (4-6) in that

$$x^* = x - A^{-1}g(x) \tag{4-7}$$

Since A is a constant positive-definite matrix, the solution is obtained in one step from any point x. Before the presentation of conjugate gradient techniques, some properties of conjugate directions are examined.

Definition 6: [Conjugate vectors]
Let A be a positive-definite matrix. A vector v_1 is said to be conjugate (with respect to A) to a vector v_2 if v_1 and v_2 satisfy

$$v_1'Av_2 = 0 \tag{4-8}$$

When v_1 and v_2 are interpreted as direction vectors in E^n and satisfy (4-8), they are said to be A-conjugate directions. For example, in the method of steepest descent, the vectors $g(x^k)$ and $g(x^{k+1})$ are conjugate with respect to the matrix I because, if the optimum value of α is found

for each search such that $\partial y/\partial \alpha = 0$, equation (4-4) yields

$$-g(x^{k+1})'Ig(x^k) = -g(x^{k+1})'g(x^k) = 0 \qquad (4\text{-}9)$$

Consider the following properties of conjugate directions, which are given in [53] as theorems with proofs. If A is an n x n positive-definite matrix, then

P1: If n nontrivial directions r^i, i = 1, 2, ..., n, are known to be A conjugate to one another, then these r^i's are linearly independent.

P2: When f(x) is quadratic with A equal to the matrix of second partial derivatives, and when nontrivial directions r^i, i = 0, 1, ..., n-1 are mutually conjugate with respect to A, the exact minimum point x^* of f(x) can be obtained by a sequence of n unidirection searches starting at x^0. The x^k are determined from the iteration formula

$$f(x^{k+1}) = \min_{\alpha} f(x^k + \alpha r^k), \quad k = 0, 1, 2, \ldots, n-1 \qquad (4\text{-}10)$$

with the final result being $x^n = x^*$.

Property 2 demonstrates that the method of sequential unidirectional searches is quadratically convergent when using a set of A conjugate search directions.

Equation (4-8) gives a definite criterion for conjugate search directions. In some problem formulations, such as the NLP, only the problem functions and their gradients are defined numerically so that A, the matrix of second partials, is not explicity available. Additionally, the f(x) to be minimized in general is not quadratic, and if x is far from the minimum such that the quadratic approximation to f is not valid, other relations must be used to determine conjugate directions. One such relation can be found by the following theorem. (For proof, see Appendix B)

Theorem 8: [Conjugate directions by linear independence]

Consider the quadratic equation

$$f(x) = f(x^a) + (x - x^a)'g(x^a) + \tfrac{1}{2}(x - x^a)'A(x - x^a) \tag{4-11}$$

where A is n x n and positive definite. If $\partial f(\alpha r^1)/\partial\alpha = 0$ at $\alpha = 0$ and $\partial f(x^a - x^b + \alpha r^1)/\partial\alpha = 0$ at $\alpha = 0$, where $r^1 \neq 0$ and $r^2 = x^a - x^b \neq 0$ are linearly independent, then r^1 is A-conjugate to r^2.

Theorem 8 suggests that conjugate directions can be generated without using the gradient of $f(x)$. Methods of this sort are discussed in [57,66,71]. Two well-known methods for generating conjugate directions when A is not explicitly available are given in the following sections. (See [42] for derivations.)

4.5. The Method of Fletcher and Reeves

In the conjugate gradient method of Fletcher and Reeves [28], the conjugate directions are generated as follows:

$$r^i = -g^i \qquad \text{for} \quad i = j(n+1), \qquad j = 0, 1, 2, \ldots \tag{4-12}$$

and

$$r^{i+1} = -g^{i+1} + \frac{(g^{i+1})' (g^{i+1})}{(g^i)' (g^i)} r^i, \quad \text{otherwise} \tag{4-13}$$

where g^{i+1} equals the gradient of $f(x)$ at the point x^{i+1}, and where x^{i+1} is determined from

$$f(x^{i+1}) = \min_{\alpha} f(x^i + \alpha r^i) \tag{4-14}$$

This method and the DFP method of the next section are adaptive in character in that the search directions are

not precalculated, but generated in the iterative process as the algorithm proceeds.

4.6. The DFP Method

To date, one of the most powerful and sophisticated of the gradient methods is Fletcher and Powell's [27] extension (1963) of Davidon's [15] variable metric method developed in 1959. The method is a conjugate gradient method and is called the DFP method.

In this method the A-conjugate directions are generated by

$$r^i = -H^i g^i \tag{4-15}$$

where

$$H^i = I \qquad \text{for} \quad i = j(n+1), \qquad j = 0, 1, 2, \ldots \tag{4-16}$$

and

$$H^{i+1} = H^i + \frac{(\Delta x^i)(\Delta x^i)'}{(\Delta x^i)'(\Delta g^i)} - \frac{H^i(\Delta g^i)(\Delta g^i)'H^i}{(\Delta g^i)'H^i(\Delta g^i)}, \quad \text{otherwise} \tag{4-17}$$

where $\Delta x^i = x^{i+1} - x^i$, $\Delta g^i = g^{i+1} - g^i$, g^{i+1} is the gradient of $f(x)$ at the point x^{i+1}, and x^{i+1} is found from the relation

$$f(x^{i+1}) = \min_{\alpha} f(x^i - \alpha H^i g^i) \tag{4-18}$$

The sequence $H^0, H^1, \ldots, H^{n-1}$ is a sequence of symmetric positive definite matrices. As x approaches the minimum point x^*, H^i tends to A^{-1}, the inverse matrix of second partial derivatives of the quadratic approximation to f(x) at x^*. It was shown in Section 4.4, equation (4-7), that the minimum of a quadratic function could be attained in one iteration provided that A^{-1} was available. If the DFP method is applied to a quadratic function, the search directions r^i are given by $r^i = -(H^i g^i)$ with the following results: 1) $(r^i)'Ar^j = 0$, $i \neq j$, 2) $H^n = A^{-1}$, and 3) $g^n = 0$. If the function in question is not quadratic, the sequence $\{H^i\}$ still remains positive definite and the unidirectional search is conducted over $\alpha > 0$ only.

Both the method of Fletcher and Reeves and the DFP method are guaranteed, apart from rounding errors, to locate the minimum of any quadratic function of n variables in at most n iterations. For functions that are not quadratic, the processes are iterative rather than n-step, and tests for convergence should be used. Both processes are quadratic convergent; and since nonquadratic functions, when in the proximity of a local minimum, can be adequately

approximated by a quadratic function, rapid convergence often results.

The search directions that are generated are those corresponding to the current local quadratic approximation to the function, and the rate of convergence depends upon the response to changes in the local quadratic approximation from iteration to iteration. Thus, in applying these methods to general functions, consideration must be given to 1) the choice of a starting point x^0, 2) the type of unidirectional search employed to locate each x^{i+1}, 3) the overall efficiency of the algorithm, and 4) the final convergence criterion. These points are discussed in Chapter 5.

4.7. The Method of Generating Search Directions for the Multiplier Algorithm

It was shown that in the DFP method the sequence $\{H^i\} \to A^{-1}$, the inverse matrix of second partial derivatives of $f(x)$, as $x \to x^*$, a local minimum of $f(x)$. Thus, the DFP method yields information about the curvature of $f(x)$ near x^*. This information is, however, obtained at the price of providing storage space for H and time for its manipulation when implemented on a computer. The

method of Fletcher and Reeves is more economical in that the minimum is located but no information about the curvature of f is acquired [53]. Of the methods investigated, this author feels that the DFP method is the best method for the following type of problem: i) the problem functions are given analytically and are of class C^2 with respect to their arguments, and ii) evaluation of the model (problem functions and their gradients) requires relatively large amounts of computation time as to that required for matrix manipulations that are associated with the method.

Because the DFP method generates A^{-1} without the use of second partial derivatives and because the NLP fits the format of problems most suitable to the DFP method, the DFP method is used in implementing the multiplier algorithms of this thesis.

CHAPTER FIVE

THE UNIDIRECTIONAL SEARCH IN E^n

5.1. Introduction

To define a straight line in n-dimensional Euclidean space, E^n, either two points or a point and a direction must be specified.

Definition 6: [Straight line]
If x^a and x^b are specified points in E^n, then

$$x(\alpha) = (1 - \alpha)x^a + \alpha x^b, \qquad -\infty < \alpha < \infty \tag{5-1}$$

defines a straight line which passes through points $x(0) = x^a$ and $x(1) = x^b$. If r is a specified direction vector, then

$$x(\alpha) = x^a + \alpha r, \qquad -\infty < \alpha < \infty \tag{5-2}$$

is a line which passes through the point $x(0) = x^a$ in the line direction r.

It can be shown that these two representations of a line are equivalent by identifying r with $x^b - x^a$ to obtain

$$x(\alpha) = x^a + \alpha(x^b - x^a) = x^a + \alpha r \tag{5-3}$$

Consider $y(\alpha) = f(x^a + \alpha r)$. The first derivative of $y(\alpha)$ with respect to α is

(5-4)

$$\frac{\partial y}{\partial \alpha} = r' \nabla f(x^a + \alpha r) \qquad (5\text{-}4)$$

and the second derivative of $y(\alpha)$ with respect to α is

$$\frac{\partial^2 y}{\partial \alpha^2} = r' \nabla^2 f(x^a + \alpha r) r \qquad (5\text{-}5)$$

Because of the $n(n+1)/2$ functions that must be evaluated to obtain the symmetric matrix $A = \nabla^2 f$, and the additional operations that increase as n^3 to find A^{-1}, as required by second-derivative search methods like the Newton search, it becomes quite evident that the more efficient n-dimensional search techniques make no direct use of second derivatives.

5.2. The Unidirectional Search Problem

In any practical situation, the time spent in evaluating the model (i.e., objective and constraint functions and their gradients) at various points may well dominate the time for the whole minimization process. It is therefore desirable to make the unidirectional search as efficient as possible using the generated data at previous points to the utmost in locating future points, and limiting the number of model evaluations as much as possible.

The unidirectional search problem then is to determine α_s such that†

$$\dot{y}(\alpha_s) = \frac{d}{d\alpha} y(\alpha)\Big|_{\alpha = \alpha_s} = 0 \qquad (5\text{-}6)$$

where

$$y(\alpha) = f(x^i + \alpha r^i) \qquad (5\text{-}7)$$

Since $y(0) = f(x^i)$ and the search directions r^i are generated such that $\dot{y}(0) = (r^i)'\nabla f(x^i) \leq 0$ are readily available from the minimum point of the previous search, it is only natural to use a search technique that utilizes this information.

5.3. A Cubic Convergent Search Without Second Derivatives

A highly successful and recommended method is that proposed by Davidon [15]. The method has three stages of calculation. The first stage establishes the order of magnitude of α_s and is quadratic convergent. The second stage establishes bounds on α_s such that the minimum is bracketed, and the third stage, which is cubic convergent

---

†For the remainder of the thesis, the derivative of a function y with denoted by $\dot{y}$.

interpolates α_s to a desired degree of accuracy from the upper and lower bounds determined from the second stage.

In general, the method proceeds as follows:

Phase I. Given: x^i, $f(x^i)$, $\nabla f(x^i)$, r^i, and f_{est} (the estimated minimum of $f(x)$). Find: d_0, where d_0 is the minimum point of $y(\alpha) = f(x^i + \alpha r^i)$ which is the local quadratic approximation to $f(x)$ and where f_{est} is assumed to be the well defined minimum of $y(\alpha)$.

Phase II. With d_0 obtained from phase I, $y(\alpha)$ is evaluated at $\alpha = d_0, 2d_0, 4d_0, \ldots, d_1, d_2, d_3$ where the step is doubled each time until a step d_3 is determined with the property that $y(d_2) \leq y(d_3)$. At this point the gradient $\dot{y}(\alpha)$ is evaluated at d_2 and, depending upon its sign, the points d_a and d_b are assigned to two of the points from the set $\{d_1, d_2, d_3\}$ which bracket the minimum.

Phase III. This phase performs a cubic interpolation between the points d_a and d_b to locate an approximate minimum α_s of $y(\alpha)$, where $y(\alpha)$ is of the cubic form

$$y(\alpha) = a(\alpha - d_a)^3 + b(\alpha - d_a)^2 + c(\alpha - d_a) + d \qquad (5\text{-}8)$$

The derivatives $\dot{y}(d_a)$ and $\dot{y}(d_b)$ are computed and are used with the values $y(d_a)$ and $y(d_b)$ in (5-8) to determine

the constants a, b, c, and d. With these constants determined, the minimum α_s is given by the following formula, which is found from the solution of (5-8) using ordinary min/max theory.

$$\alpha_s = d_a - \frac{b}{3a} + \text{sgn}(a)\left[\left(\frac{b}{3a}\right)^2 - \frac{c}{3a}\right]^{\frac{1}{2}} \qquad (5\text{-}9)$$

where

$$\text{sgn}(a) = \begin{cases} +1, & a > 0 \\ -1, & a < 0 \end{cases} \qquad (5\text{-}10)$$

Davidon gives a compact representation of (5-9) whereby α_s can be found from

$$\alpha_s = d_b - (d_b - d_a)\left[\frac{\dot{y}(d_b) + u_2 - u_1}{\dot{y}(d_b) - y(d_a) + 2u_2}\right] \qquad (5\text{-}11)$$

where

$$u_1 = \dot{y}(d_a) + \dot{y}(d_b) + 3\left[\frac{y(d_a) - y(d_b)}{d_b - d_a}\right] \qquad (5\text{-}12)$$

and

$$u_2 = \left[u_1^2 - \dot{y}(d_a)\dot{y}(d_b)\right]^{\frac{1}{2}} \qquad (5\text{-}13)$$

If $\dot{y}(\alpha_s)$ is less than both $\dot{y}(d_a)$ and $\dot{y}(d_b)$ then α_s may be accepted as a sufficiently accurate estimate of the minimum point. On the other hand, phase III may be repeated over one of the intervals (d_a, α_s) or (α_s, d_b) that brackets the minimum until the desired accuracy of $\dot{y}(\alpha_s)$ is obtained.

A single application of the interpolation formula (5-11), (5-12), and (5-13) produces the exact value of α_s in the limit as f(x) becomes quadratic in the neighborhood of a local minimum [28]. Further, increased accuracy in the general case is obtained only at the cost of more evaluations of the model. Interpolation is most likely to be inaccurate in regions remote from the minimum, and it is uneconomic to require a high degree of accuracy in such regions. Numerical tests have shown that higher accuracy interpolation does not yield a significant reduction in the number of iterations required in the minimization process. From the standpoint of stability, however, a certain degree of accuracy must be achieved to ensure that the values of $f(x^i)$ do form a decreasing sequence $(\ldots \geq f(x^{i-1}) \geq f(x^i) \geq f(x^{i+1}) \ldots)$. Hence, the provision for repeating the interpolation over smaller intervals.

Since high accuracy is not required in regions remote from a local minimum, and in the vicinity of such a minimum f(x) is adequately represented by the local quadratic approximation, this author feels that a single stage, quadratic convergent search algorithm would be highly efficient in a unidirectional search for a minimum. For the general constrained NLP where the model is complex and the variables are highly interactive, it can be assumed that one gradient evaluation is approximately equivalent to n function evaluations. Thus, the required number of gradient evaluations in the unidirectional search should be held to a minimum. It can be seen that a single application of Davidon's cubic interpolation formula requires three gradient evaluations per search, and an additional gradient evaluation for each repeated interpolation. The search algorithm presented by this author in the next section requires only one gradient evaluation per search. This gradient evaluation occurs at the end of the search when α_s is located, and is used for generating the next search direction, for ensuring that $\partial y(0)/\partial\alpha < 0$ at the beginning of the next search, and for testing the efficiency and overall convergence of the minimization process.

5.4. A Quadratic Convergent Search Without Second Derivatives

Consider again the unidirectional search problem given by (5-6) and (5-7) where $y(\alpha)$ is assumed to be a quadratic of the form

$$y(\alpha) = a(\alpha - d_1)^2 + b(\alpha - d_1) + c, \quad a > 0 \tag{5-14}$$

with a well-defined minimum at

$$\alpha_s = d_1 - \frac{b}{2a} \tag{5-15}$$

Suppose two points, d_1 and d_2, are specified such that $y(d_1)$, $\dot{y}(d_1)$ and $y(d_2)$ can be evaluated from (5-14). Since three data are sufficient to solve for all three constants of the quadratic function, the data $y(d_1)$, $\dot{y}(d_1)$ and $y(d_2)$, where $d_1 = 0$, are used in (5-14) to determine the constants a, b, and c such that α_s may be determined from (5-15). With a, b, and c determined by these three data, the minimum point α_s is

$$\alpha_s = \frac{\frac{1}{2}(d_2)^2 \dot{y}(d_1)}{d_2\dot{y}(d_1) + y(d_1) - y(d_2)} \tag{5-16}$$

Suppose now a second situation where d_1, d_2, and d_3 are specified and $y(d_1)$, $y(d_2)$, and $y(d_3)$ have been evaluated. Again, these three data are sufficient to determine

a, b, and c from (5-14) such that the minimum α_s can be found from (5-15). With a, b, and c determined by these three data, the minimum point α_s is

$$\alpha_s = d_1 + \tfrac{1}{2}\left(\frac{d_{21}^s[y(d_3)-y(d_1)] - d_{31}^s[y(d_2) - y(d_1)]}{d_{21}[y(d_3)-y(d_1)] - d_{31}[y(d_2)-y(d_1)]}\right) \tag{5-17}$$

where

$$\begin{aligned} d_{21} &= d_2 - d_1 & d_{31} &= d_3 - d_1 \\ d_{21}^s &= (d_{21})^2 & d_{31}^s &= (d_{31})^2 \end{aligned} \tag{5-18}$$

Both (5-16) and (5-17) will be used in the following search technique as a means of estimating the minimum of the local quadratic approximation to f(x), depending upon the available data at each point of the unidirectional search. In general, the search proceeds as follows:

Step 1: Given: x^k, $f(x^k)$, $\dot{f}(x^k)$ and r^k.
Find: α_s which is the step in the r^k direction that yields the minimum value of $y(\alpha) = f(x^k + \alpha r^k)$, where $y(\alpha)$ is of the form (5-14). In general, $f(x^k + \alpha r^k)$ is not quadratic but (5-14) is used regardless.

Step 2: Set $d_1 = 0$ and $y(d_1) = f(x^k)$. The search directions generated by the DFP method allow a search over $\alpha > 0$ only, and from the development in Chapter 4 we know that $\dot{y}(0) = (r^k)'\dot{f}(x^k) < 0$. Assign a lower bound, $d_{low} > 0$, and an upper bound, $d_{up} > 1$ on the parameter α. Also assign a reduction (division) constant $\gamma > 1$, and an acceleration (multiplication) constant $\rho > 1$, such that a step d_i may be reduced or accelerated in the r^k direction. The initial step d_2 is assigned by

$\|r^k\| < 200$

$$d_2 = \begin{cases} 0.1, & r^k = -\dot{f}(x^k) \text{ (negative gradient direction)} \\ 1.0, & \text{otherwise (conjugate gradient direction)} \end{cases} \tag{5-19}$$

or

$\|r^k\| \geq 200$

$$d_2 = \begin{cases} 10/\|r^k\|, & 10/\|r^k\| > d_{low}, \\ d_{low}, & \text{otherwise} \end{cases} \tag{5-20}$$

reassign: $d_{low} = .01\, d_{low}$, $\gamma = 2\gamma$

For most searches d_2 is assigned by (5-19), but in the initial search iterations of the minimization process,

the point x^k may be far away from the minimum x^*, and the magnitude of the search direction $\|r^k\|$ may be extremely large. When this situation arises, the fixed step d_2 assigned by (5-19) may lead to a constraint breakthrough. Thus, for $\|r^k\| \geq 200$, d_2, d_{low}, and γ are modified according to (5-20).

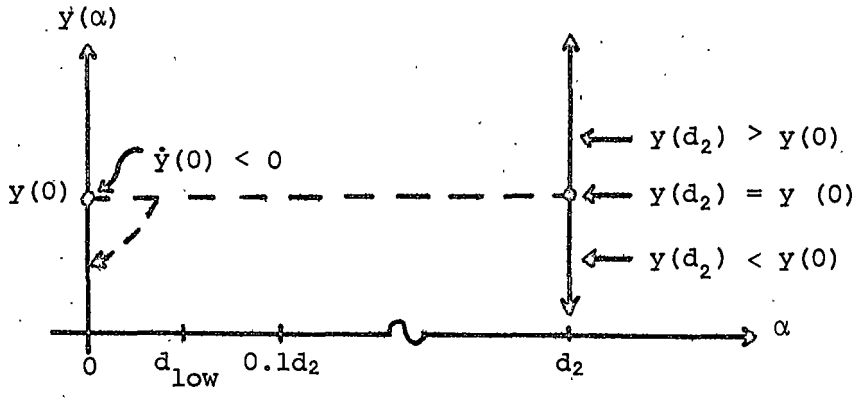


Figure 5-1. Possible situations considered in step 3 of unidirectional search.

Step 3: With d_2 determined, $y(d_2)$ is evaluated and compared to $y(d_1)$. Three possibilities arise from this comparison as shown in Figure 5-1: $y(d_2) > y(d_1)$ is considered in step 4; $y(d_2) = y(d_1)$ is considered in step 5; and $y(d_2) < y(d_1)$ is considered in step 6.

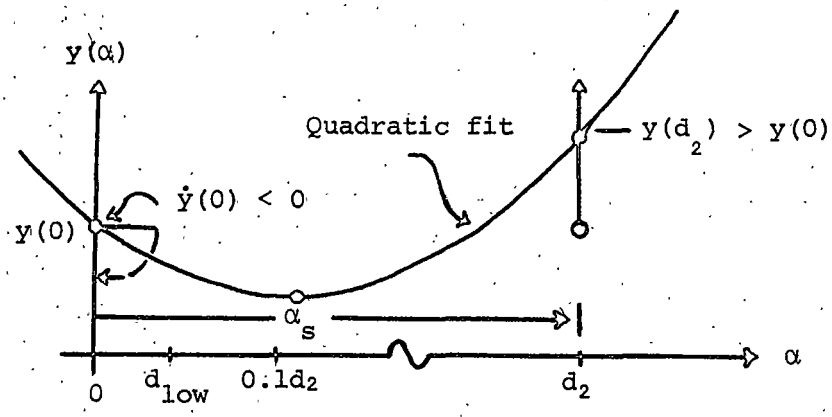


Figure 5-2. Representative case of step 4 where α_s is determined by using $y(0)$, $\dot{y}(0) < 0$, and $y(d_2)$ in equation (5-16).

Step 4: $y(d_2) > y(d_1)$, and $d_1 = 0$. The values $y(d_1)$, $\dot{y}(d_1)$ and $y(d_2)$ are used in (5-16) to estimate α_s. It can be seen from Figure 5-2 that α_s is contained in the open interval $(0, d_2)$.

If $0 < \alpha_s \leq 0.1d_2$ and $d_2 > d_{low}$ then d_2 is reassigned as follows

$$d_2 = \begin{cases} \alpha_s, & d_{low} < \alpha_s \leq 0.1d_2 \\ d_{low}, & 0 < \alpha_s \leq d_{low} \end{cases} \qquad (5\text{-}21)$$

and step 3 is again repeated.

If $0 < \alpha_s < d_2$ and $d_2 \leq d_{low}$ then α_s is the accepted minimum point of the search; evaluate $y(\alpha_s)$ and continue to step 8.

If $0.1d_2 < \alpha_s < d_2$ and $d_2 > d_{low}$, then $y(\alpha_s)$ is evaluated and compared to $y(d_1)$. If $y(\alpha_s) > y(d_1)$, d_2 is reduced by γ, i.e., $d_2 = d_2/\gamma$, and step 3 is repeated. On the other hand, if $y(\alpha_s) \leq y(d_1)$ as shown in Figure 5-3, $y(d_1)$, $y(\alpha_s)$ and $y(d_2)$ are used in (5-17) to yield the quadratic minimum α_{sc} of these three points. The value α_{sc} is calculated as a check on the accuracy on the local quadratic approximation to $f(x)$. Define the ± 10% 'window' interval $w(\alpha)$

$$w(\alpha) = [0.9\alpha, \quad 1.1\alpha] \tag{5-22}$$

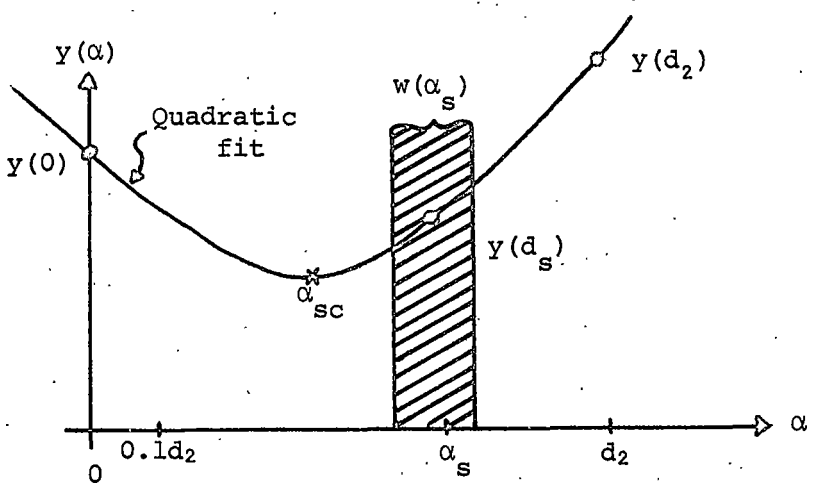


Figure 5-3. Representative case of step 4 where α_{sc} is determined by using $y(0)$, $y(\alpha_s)$, and $y(d_2)$ in equation (5-17). The shaded area represents the window interval $w(\alpha_s)$.

If α_{sc} is contained in $w(\alpha_s)$ then $y(\alpha_s)$ is considered an accurate estimate of the minimum of the search. Continue to step 8. Otherwise α_{sc} is the accepted step and $y(\alpha_s)\Big|_{\alpha_s = \alpha_{sc}}$ is the accepted minimum of the search. Continue to step 8.

Step 5: $y(d_2) = y(d_1)$ where $d_1 = 0$. Assign $d_3 = \rho d_2$. Evaluate $y(d_3)$ as shown in Figure 5-4 and compare it to $y(d_2)$.

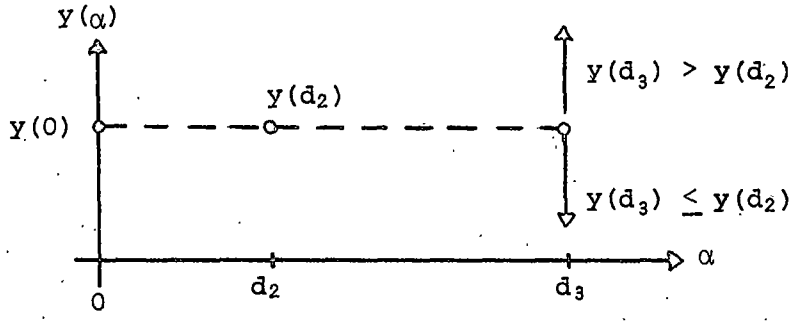


Figure 5-4. Possible situations considered in step 5 of unidirectional search.

If $y(d_3) \leq y(d_2)$ go to step 7. The condition where $y(d_1) = y(d_2) = y(d_3)$ may occur when $\|r^k\|$ is extremely small. In this situation the initial fixed step d_2 from

(5-19), and hence $d_3 = \rho d_2$, are not large enough to cause a significant change in $y(\alpha)$, due to roundoff errors, because $f(x)$ is relatively flat with these step sizes. Thus, acceleration steps are taken to force d_1, d_2, and d_3 away from $\alpha = 0$ as shown in step 7.

If $y(d_3) > y(d_2)$, $y(d_1)$, $y(d_2)$, and $y(d_3)$ are used in (5-17) to estimate α_s; $y(\alpha_s)$ is then the accepted minimum of the search. Continue to step 8.

Step 6: $y(d_2) < y(d_1)$ where $d_1 = 0$. The values $y(d_1)$, $\dot{y}(d_1)$, $y(d_2)$ are used in (5-16) to locate the minimum point a_1 of the resulting quadratic fit, as shown in Figure 5-5.

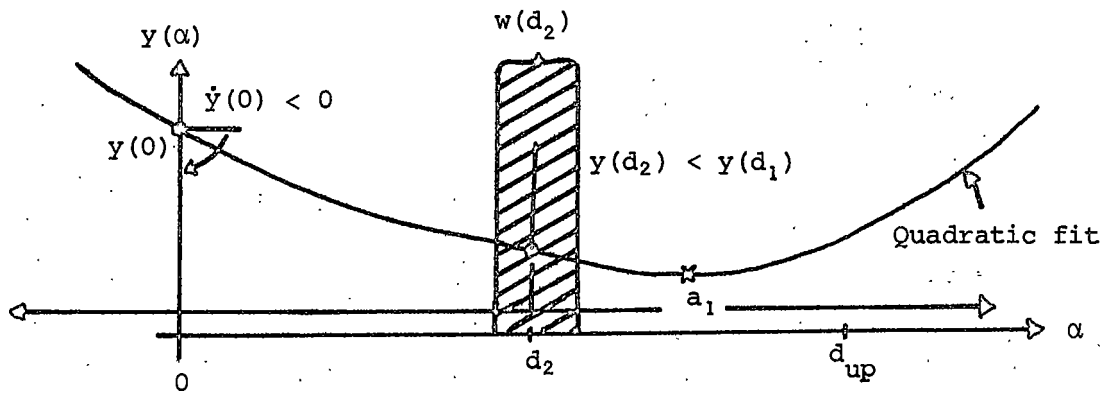


Figure 5-5. Representative case of step 5 where a_1 is determined by using $y(0)$, $\dot{y}(0)$, $y(d_2)$ in equation (5-16).

If a_1 is contained in $w(d_2)$, then the local quadratic approximation to $f(x)$ is accurate, d_2 is the accepted minimum point, and $y(\alpha_s)\Big|_{\alpha_s = d_2}$ is the accepted minimum of the search. Continue to step 8.

If $a_1 \geq d_{up}$, assign $d_3 = d_{up}$ and continue to step 7.

If $1.1d_2 < a_1 < d_{up}$, $y(d_1)$, $y(d_2)$ and $y(a_1)$ are used in (5-17) to estimate α_{sc} as a check on the local quadratic approximation. If $y(a_1) \geq y(d_2)$ then $\alpha_s = \alpha_{sc}$ is the minimum step and $y(\alpha_s)\Big|_{\alpha_s = \alpha_{sc}}$ is the accepted minimum of the search. Continue to step 8. If $y(a_1) < y(d_2)$ and α_{sc} is contained in $w(a_1)$, then $\alpha_s = a_1$ is the minimum step and $y(\alpha_s)\Big|_{\alpha_s = a_1}$ is the minimum of of the search. Continue to step 8.

If α_{sc} is not contained in $w(a_1)$, then assign $d_1 = d_2$, $y(d_1) = y(d_2)$, $d_2 = a_1$, $y(d_2) = y(a_1)$, $d_3 = \rho d_2$, and continue to step 7.

If $0 < a_1 < 0.9d_2$ and if $y(a_1) \leq y(d_2)$, $\alpha_s = a_1$ is the minimum point and $y(\alpha_s)\Big|_{\alpha_s = a_1}$ is the accepted minimum

of the search. Continue to step 8. Otherwise assign $d_2 = a_1$, $y(d_2) = y(a_1)$, and continued to step 7.

If $a_1 < 0$ continue to step 7. This condition occurs when the local quadratic approximation of $f(x)$ is not valid and $y(d_2)$ is below the projected gradient $\dot{y}(0)$ as shown in Figure 5-6. In this situation $y(\alpha)$ has the wrong curvature resulting in $a_1 < 0$.

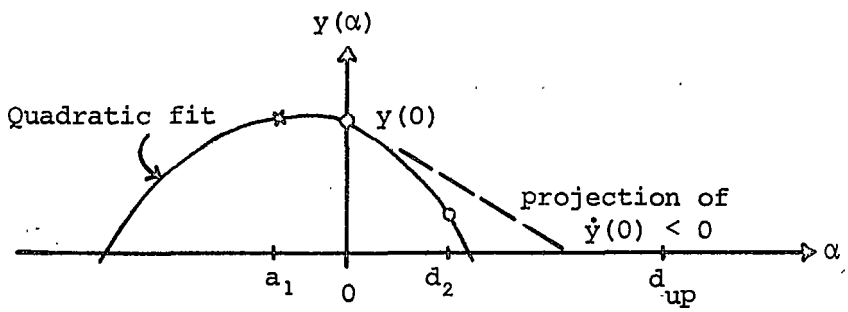


Figure 5-6. Representative case of step 6 where using $y(0)$, $\dot{y}(0)$, $y(d_2)$ in (5-16) gives the wrong curvature to $f(x)$.

Step 7: Evaluate $y(\alpha)$ at the points d_3, ρd_3, $\rho^2 d_3$, $\rho^3 d_3$, ..., d_a, d_b, d_c, where the terminal value d_c is determined on the basis that $y(d_c) \geq y(d_b)$ as shown in Figure 5-7. At each evaluation the points d_1, d_2, d_3 are accelerated by reassigning the values $d_1 = d_2$, $d_2 = d_3$, and $d_3 = \rho d_2$,

such that at the terminal condition $y(d_1) = y(d_a)$, $y(d_2) = y(d_b)$, and $y(d_3) = y(d_c)$. The values $y(d_1)$, $y(d_2)$, and $y(d_3)$ are then used in (5-17) to estimate α_s. If $y(\alpha_s) \leq y(d_2)$, $y(\alpha_s)$ is the accepted minimum of the search. Continue to step 8. Otherwise $\alpha_s = d_2$ is the minimum step and $y(\alpha_s)\Big|_{\alpha_s = d_2}$ is the accepted minimum of the search.

Continue to step 8.

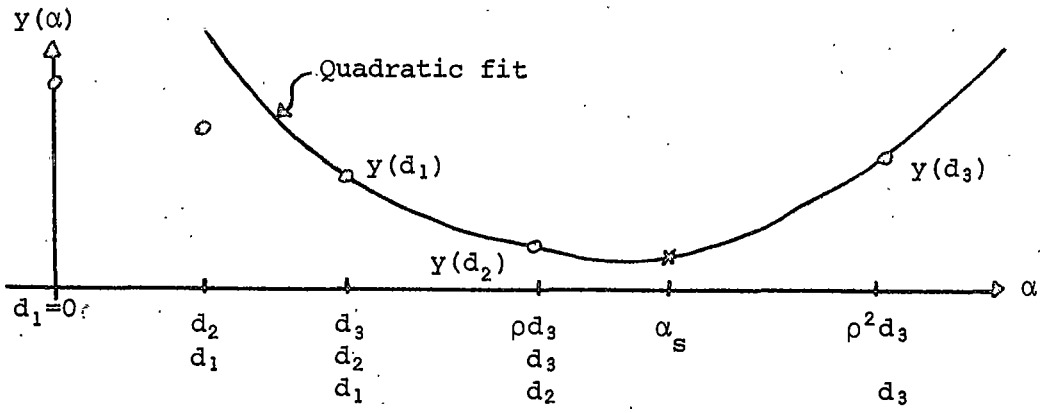


Figure 5-7. Representative case of step 7 showing acceleration of points d_1, d_2, and d_3 away from $\alpha = 0$.

Step 8: With the estimated minimum step α_s and minimum value calculated from the above steps, the new point x^{k+1} is

$$x^{k+1} = x^k + \alpha_s r^k \qquad (5\text{-}23)$$

with the desired result that

$$f(x^{k+1}) \stackrel{\sim}{=} \min_{\alpha} f(x^k + \alpha r^k) \qquad (5\text{-}24)$$

In the above procedure, suppose that the initial value of the function $y(0)$ was saved as the initial best point of the search y_{best}. Then for each step α, $y(\alpha)$ could be compared to y_{best} and if $y(\alpha) < y_{best}$, $y(\alpha)$ would become the new y_{best} of the search. Generally, the point $y(\alpha_s)$, obtained when step 8 is entered, is the best point. But as a check against accepting a greater function value for a search in areas far from x^*, where the quadratic approximation is not valid and L_a is highly nonlinear, $y(\alpha_s)$ should be compared to y_{best} and the smallest value taken for the end point of the search. This additional check is included in the computer implementation of the search procedure, and will be discussed further in Section 6.6 and 6.7.

5.5. The Unidirectional Search for the Multiplier Algorithm

Both search techniques presented in this chapter were implemented and tested on various sample problems of Chapter 7 with the DFP method for generating search directions. The quadratic search technique of Section 5.4

consistently required fewer function evaluations (20% to 40% less) than that of Davidon's, both in function calls per iteration, and in total function calls for the entire minimization process. Only one gradient call, as compared to three (or more) for Davidon's method, per iteration is required for the quadratic search.

From Chapters 2, 4, and 5 we now have a multiplier algorithm, a means of generating search directions in n-space, and an efficient quadratic search technique for locating the minimum along each search direction. Consideration can now be given to the choice of an initial point x^0, the overall efficiency of the algorithm, and the final convergence criterion.

The augmented Lagrangian of (2-22) incorporates both the equality and inequality constraints by exterior penalty functions. Therefore, x^0 need not be in the strictly feasible area of the inequality constraints. Therefore, no additional stage need be incorporated into the algorithm for finding a starting point. The starting point for a particular problem using this algorithm then is selected from the best starting point known from past analysis or on the basis of a sensible guess.

The overall efficiency of this algorithm is based on $\|\nabla L_a\| < \varepsilon_2$, and $\|\Delta x\| < \varepsilon_3$. The final convergence criterion is based on $\|\nabla L_a\| < \varepsilon_1$. These criteria are explained in detail in Chapter 6 where they occur in the algorithm implementation.

CHAPTER SIX

IMPLEMENTATION OF THE ALGORITHM

6.1. Introduction

The multiplier algorithm, like most penalty-function algorithms, solves the constrained minimization problem by a sequence of unconstrained minimization problems. The essential structure of this type of algorithm is shown in Figure 6.1. After initialization it consists of the iterative sequence: 1) generate the search direction r^k from information available at the current search point x^k; 2) find the step α in the search direction which yields the minimum of the augmented function in the r^k direction; 3) calculate and output the status at the new search point x^{k+1}; 4) test conditions for updating multipliers, penalty weights, and other algorithm parameters, and perform the update if conditions are satisfied; 5) test for convergence or stopping conditions and output terminal status if satisfied; 6) if the minimization sequence has not been terminated, repeat the process with the new current search point x^{k+1}. A FØRTRAN code is given in this thesis for solving the NLP by a comprehensive implementation of the above sequential process. Figure 6-2 shows the subroutines used to implement the multiplier algorithm. The general part of the code consists of 13 subroutines which are shown within the dashed lines; the code for these subroutines

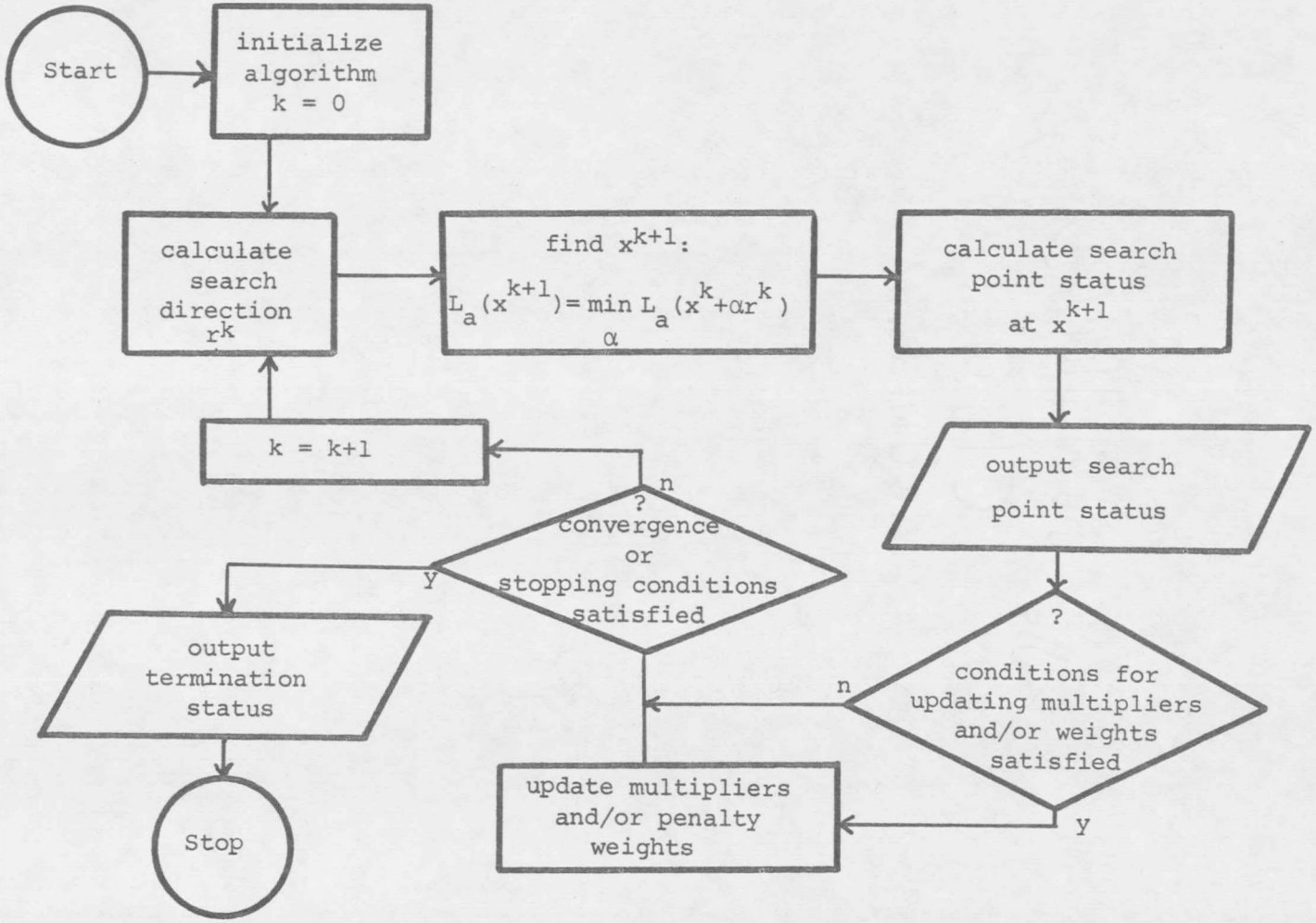


Figure 6-1. Minimization sequence using a multiplier algorithm.

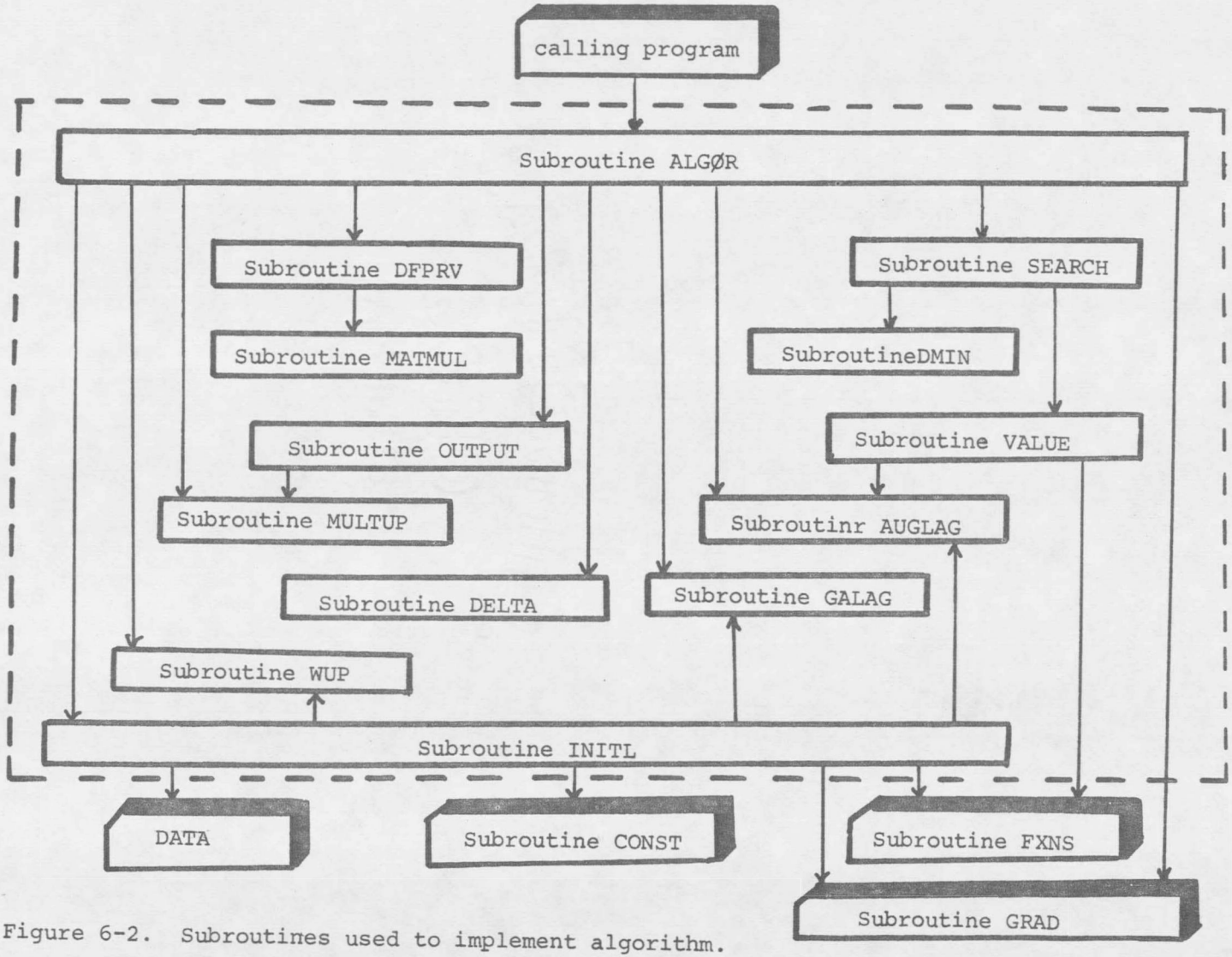


Figure 6-2. Subroutines used to implement algorithm.

remains the same for every problem. Only the 4 remaining routines and data block require modification from problem to problem as they supply the analytical representation of a particular problem and information dictating operating modes of the general routine for that problem. An explanation of each subroutine, to describe its particular function in relation to the total algorithm, is now in order.

6.2. User Supplied Subroutines

Four subroutines for any given problem and one data deck are required to supply the analytical model and parameter settings for the general routine. The short calling program properly dimensions all of the arrays and makes a call to the general working subroutine ALGØR. Subroutine ALGØR in turn directs subroutine INITL to read initial conditions and values for flags, counters, and parameters supplied in DATA, and to set operational modes of the algorithm. Once the algorithm is initialized, it performs the problem minimization sequentially, calling subroutines FXNS and GRAD whenever an evaluation of the problem model is required.

Subroutine FXNS supplies the cost function, f(x), and the equality and inequality constraint functions,

$\{h_i(x)\}_{i=1}^{m_1}$ and $\{g_j(x)\}_{j=1}^{m_2}$, respectively. Subroutine CØNST supplies all nonzero constant gradient components of the problem functions given in subroutine FXNS. It is called only once during the initialization of the algorithm. The dynamic gradient components (those dependent upon x) of the problem functions are assigned in subroutine GRAD.

Reference [40] contains an analytical model for general problems and the information required for dimensioning vectors, calling ALGØR, and supplying the data. For convenience the main body of this information is listed in Appendix C. In subsequent sections the subroutines inside the dashed line of Figure 6-2 are explained. In the flow connections, the numbers that are encircled and assigned to blocks correspond to statement numbers in the program listing in Appendix E. The letters that are enclosed in the small, pointed semi-square boxes are page connectors; some flow diagrams are continued on more than one page. A summary of the key notation used in the FØRTRAN coded program and the flow connections for this program is given in Appendix D.

6.3. Subroutine ALGØR

Subroutine ALGØR is the organization and command program. It directs the minimization sequence as depicted in Figure 6-1. Control is given to this subroutine from the calling program. It then coordinates the operation of all remaining subroutines until the algorithm is terminated, returning control to the calling program which then ends computer execution.

The algorithm has the capability of operating in three modes: 1) straight penalty-function mode with all multipliers equal to zero; 2) a straight multiplier mode where multipliers can be initialized from the data input, or are initialized by the algorithm at the end of the first cycle (each cycle generally consists of between N+1 and NSRCH (usually NSRCH = 2N+1) unidirectional searches): 3) a combined mode in which the penalty-function method operates for a number (NCYCL) of searches, after which the mulitplier method is internally initialized and used for the remainder of the minimization process. KSRCH is the counter for the number of cycles completed during the total process.

The flow connection for subroutine ALGØR is given in Figure 6-3. The algorithm is initialized and the mode

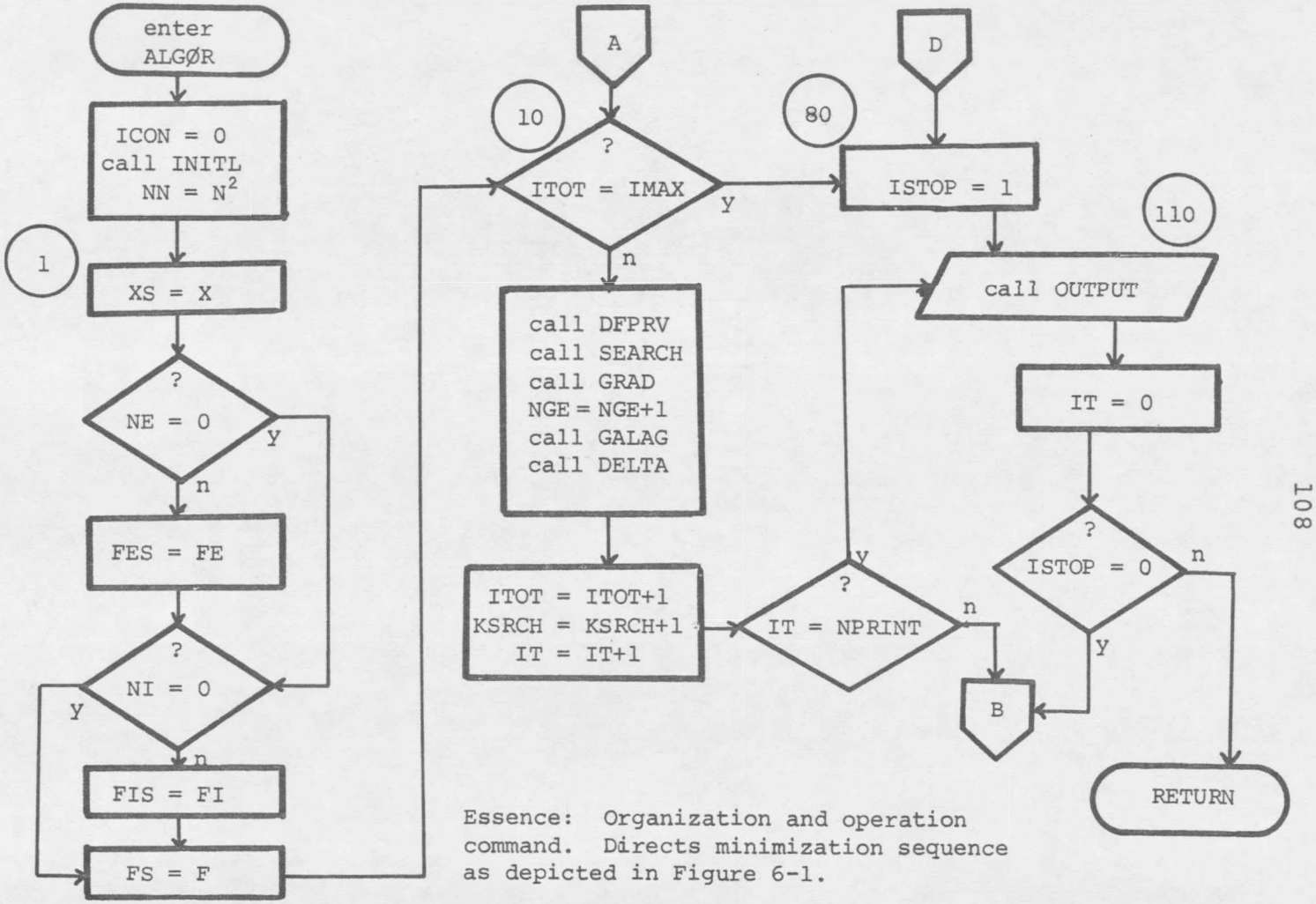


Figure 6-3. Flow connection for subroutine ALGØR.

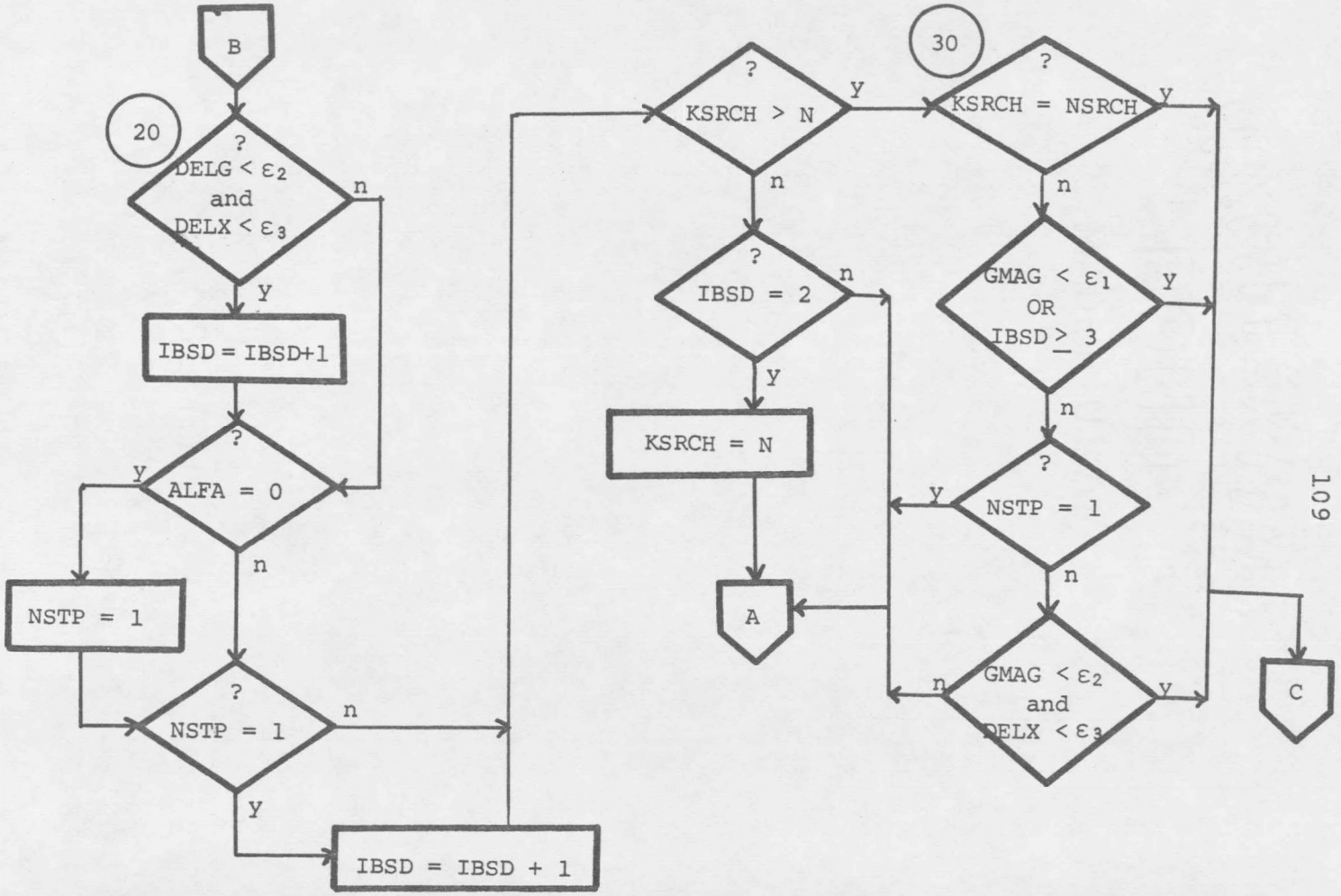


Figure 6-3. (continued)

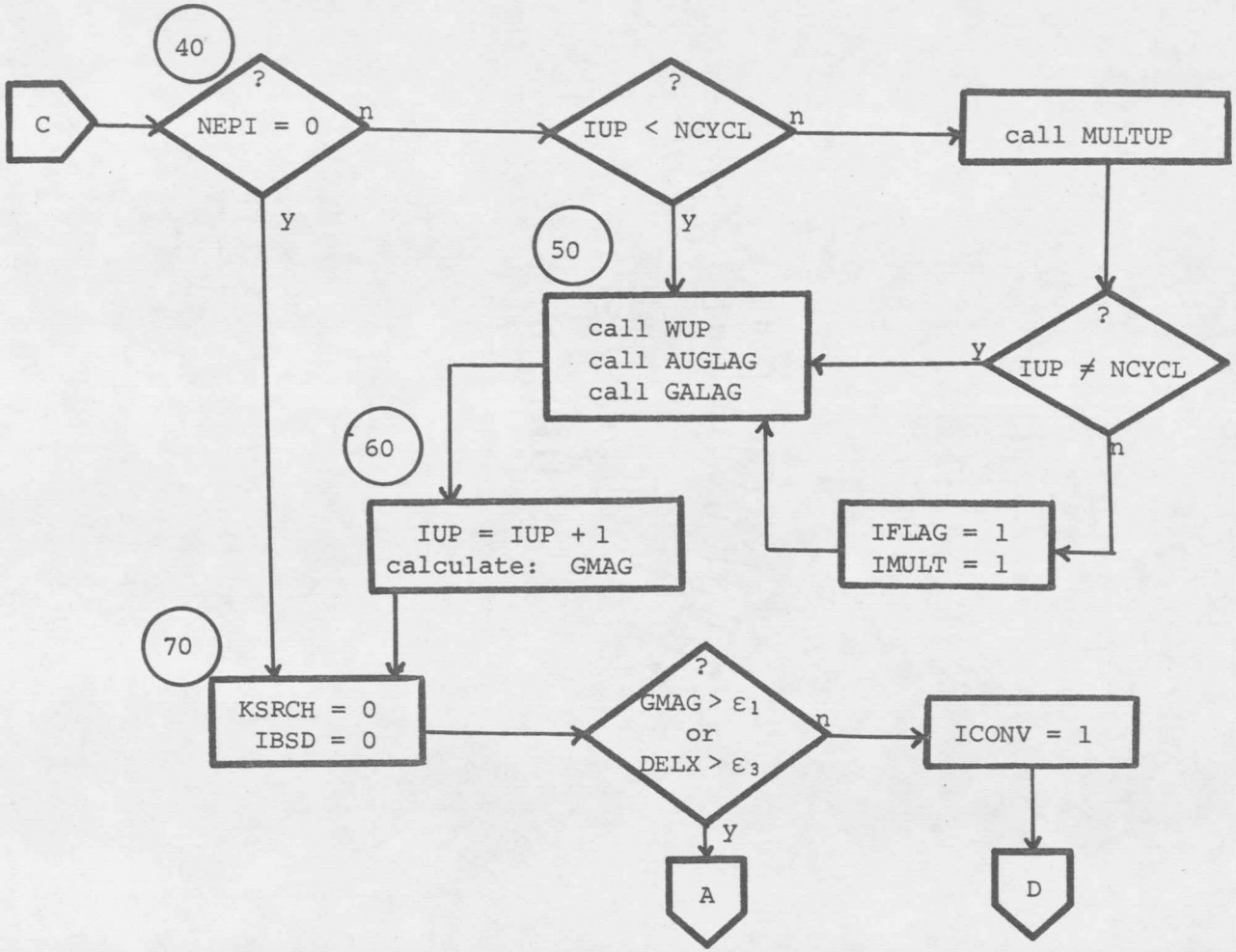


Figure 6-3. (continued)

of operation established by a call to INITL. After the initial starting point information is saved (1), the minimization sequence is then started; the search direction R is generated by DFPRV; the unidirectional search is conducted by SEARCH to find an approximate minimum (ALFA) of the augmented Lagrangian (AL) in the R direction; the gradient of AL (GAL) is then evaluated, and the search point status is updated by DELTA. Appropriate counters are incremented, and the search point status is printed (110) if NPRINT searches have been made since the last output.

Tests to determine if the last unidirectional search made a significant change in the search point are conducted (20):

i) DELG < ε_2 and DELX < ε_3

ii) ALFA = 0

If either of these conditions is satisfied, the last unidirectional search made no significant change in the search point x, and the poor search counter (IBSD) is incremented.

After the above efficiency tests have been conducted the following conditions are tested to determine if the weights or multipliers should be updated (30):

iii) KRSCH > N

iv) IBSD = 2

v) KSRCH = NSRCH vi) NSTP = 1

vii) GMAG < ε_1 or IBSD $\geq$ 3

viii) GMAG < ε_2 and DELX < ε_3

As seen from Figure 6-3, if conditions iv and vi are satisfied, the total number of iterations (ITOT) is compared to the prescribed maximum limit (IMAX) (10). If this limit has been reached without convergence, ISTOP is set equal to 1 (80) and the algorithm is terminated after printing the status of the last point (110). Otherwise, if condition iii and one or more of conditions v, vii, and viii are satisfied, the updating procedure proceeds as follows:

a) if the problem is unconstrained (NEPI = 0) no update is effected

b) if IUP < NCYCL, the weights only are updated by weight update rule (50) (penalty function mode)

c) if IUP = NCYCL, the multipliers are initialized (IMULT = 1), and the weights for the multiplier phase are initialized (IFLAG = 1) (50) (change from penalty-function to multiplier mode)

d) if IUP > NCYCL, both the multipliers and weights are updated by their respective update rules (multiplier mode)

The updating procedure changes the value of the augmented Lagrangian and its gradient. Thus, they are re-evaluated by calling AUGLAG, and GALAG (50). An update also signifies the completion of a cycle, thus IUP is

incremented (60), and the counters KSRCH and IBSD are re-initialized (70).

After the updating procedure, the algorithm is tested for overall convergence as follows:

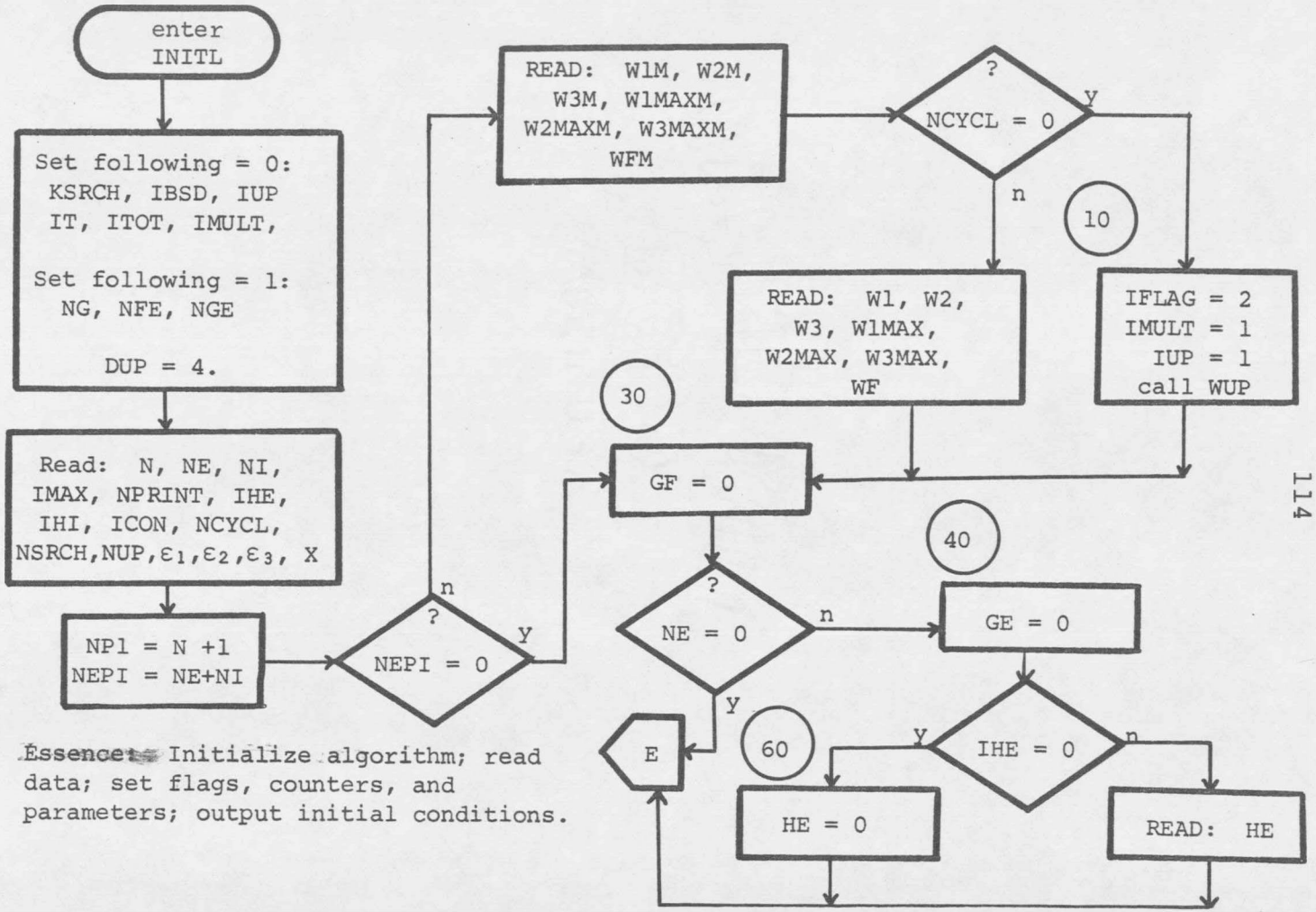
ix) GMAG > ε_1

x) DELX > ε_3

If neither of the conditions ix or x are satisfied, convergence criteria are satisfied, i.e., both GMAG and DELX have been reduced below their respective prescribed values ε_1 and ε_3. When convergence criteria are satisfied, ICONV is set equal to 1, ISTOP is set equal to 1 (80), and the algorithm is terminated after printing the status of the last point (110). If one or both of the conditions ix and x are satisfied, the process is again repeated by returning to (10).

6.4. Subroutine INITL

It is the purpose of this subroutine to effect the initialization of the algorithm. A flow connection for this subroutine is given in Figure 6-4. Flags, counters, and some parameters are set by INITL, and others are read from the data deck (see Appendix C).



Essence: Initialize algorithm; read data; set flags, counters, and parameters; output initial conditions.

Figure 6-4. Flow connection for subroutine INITL.

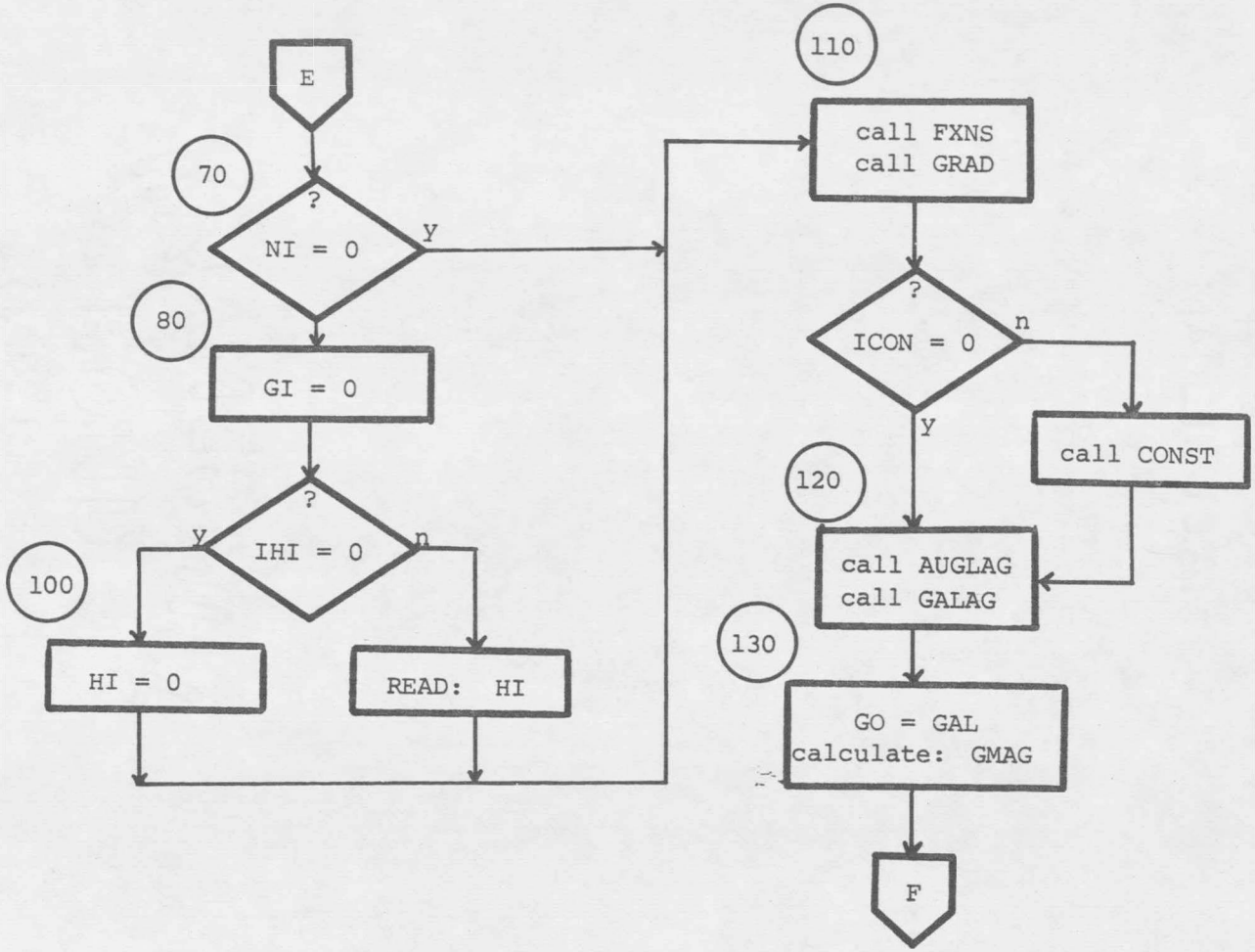


Figure 6-4. (continued)

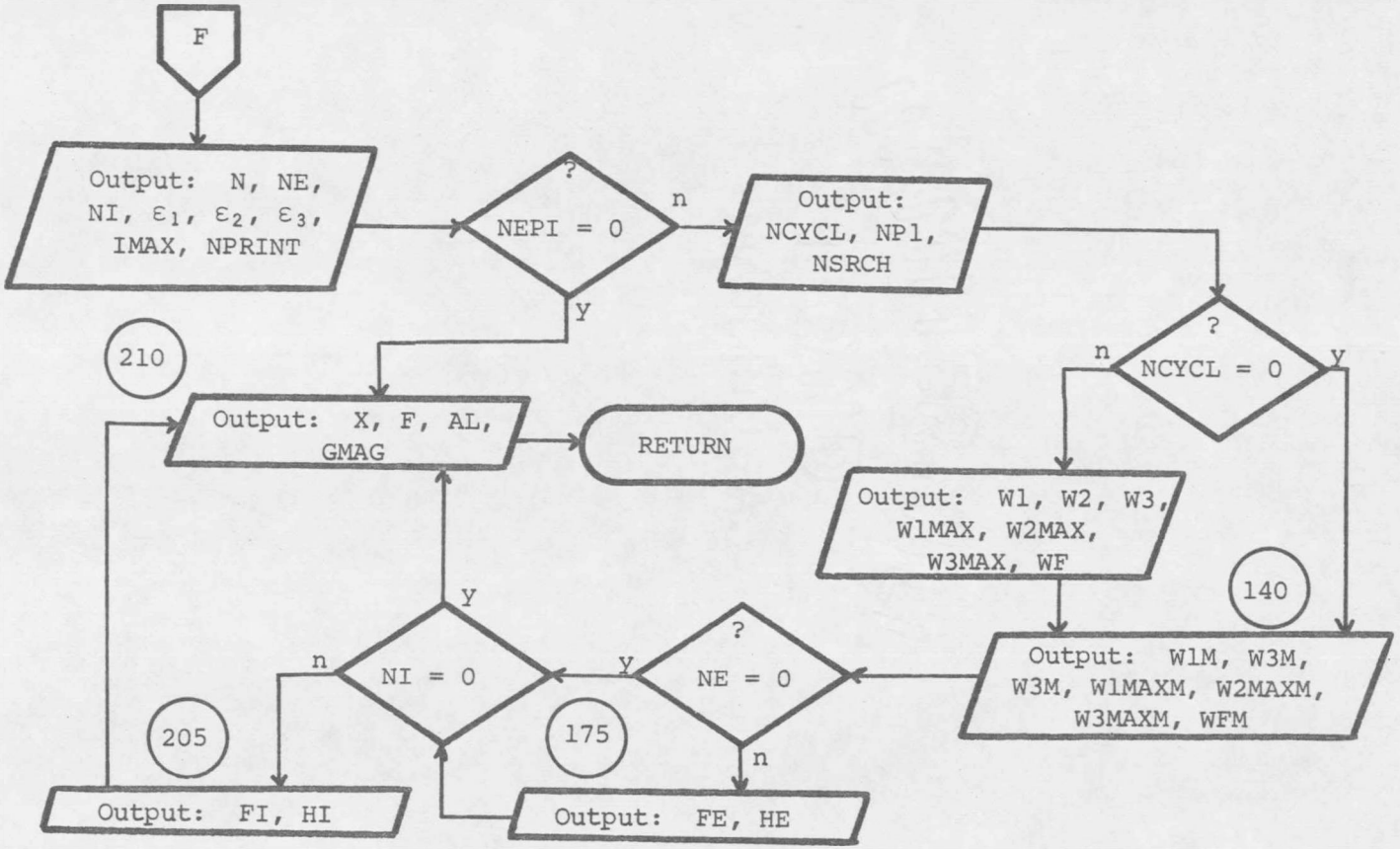


Figure 6-4. (continued)

If the problem is constrained the multiplier weights are read. NCYCL is then tested to determine the operating mode of the algorithm, and if the penalty-function mode or penalty/multiplier mode is specified the penalty-function weights are also read.

Only the nonzero components of the function gradients are listed in subroutine GRAD and CONST. Before these subroutines are called all gradient components of the cost function are zeroed (30). If there are any equality constraints, their gradients are also zeroed (40). If any of the multipliers for the equality constraints are to be initialized to some nonzero constant (IHE = 1) they are read from the data deck. Otherwise they are initialized to zero by INITL (60). A similar procedure is followed for the inequality constraints (80), (100).

The problem model is then evaluated (110), and the augmented Lagrangian and its gradient are formed (120). Subroutine INITL then prints the initial conditions of all important variables, parameters, and flags before returning to subroutine ALGØR as shown in Figure 6-4 by the paths through blocks (140), (175), (205) and (210).

6.5. Subroutines DFPRV and MATMUL

The search direction R is generated by these subroutines. As shown in Figure 6-5 the direction flag (NG) is set to zero. If the last unidirectional search did not find a better point (NSTP = 1), or if this is the first search of a new cycle (KSRCH = 0) the DFP method is reinitialized (10), the H matrix is set to the identity matrix (20), and R is set to the negative of the gradient of L_a (30). The H matrix is formed through a series of 4 calls to subroutine MATMUL which performs vector matrix multiplications (40). One additional call to MATMUL in (80) generates the conjugate direction R.

After the search direction has been generated, the slope and magnitude of R are calculated (90), and are tested under the following conditions:

xi) RMAG > 0.001 GMAG xii) SLOPE $\leq$ 0

The unidirectional search is conducted over positive stepsizes only. Therefore, at the beginning of a search where ALFA = 0, the slope is required to be negative. Also, if the conjugate directions generated are less in magnitude than the magnitude of the gradient of the augmented Lagrangian by a factor of 0.001, a more significant reduction in L_a will usually be produced by a search in the negative

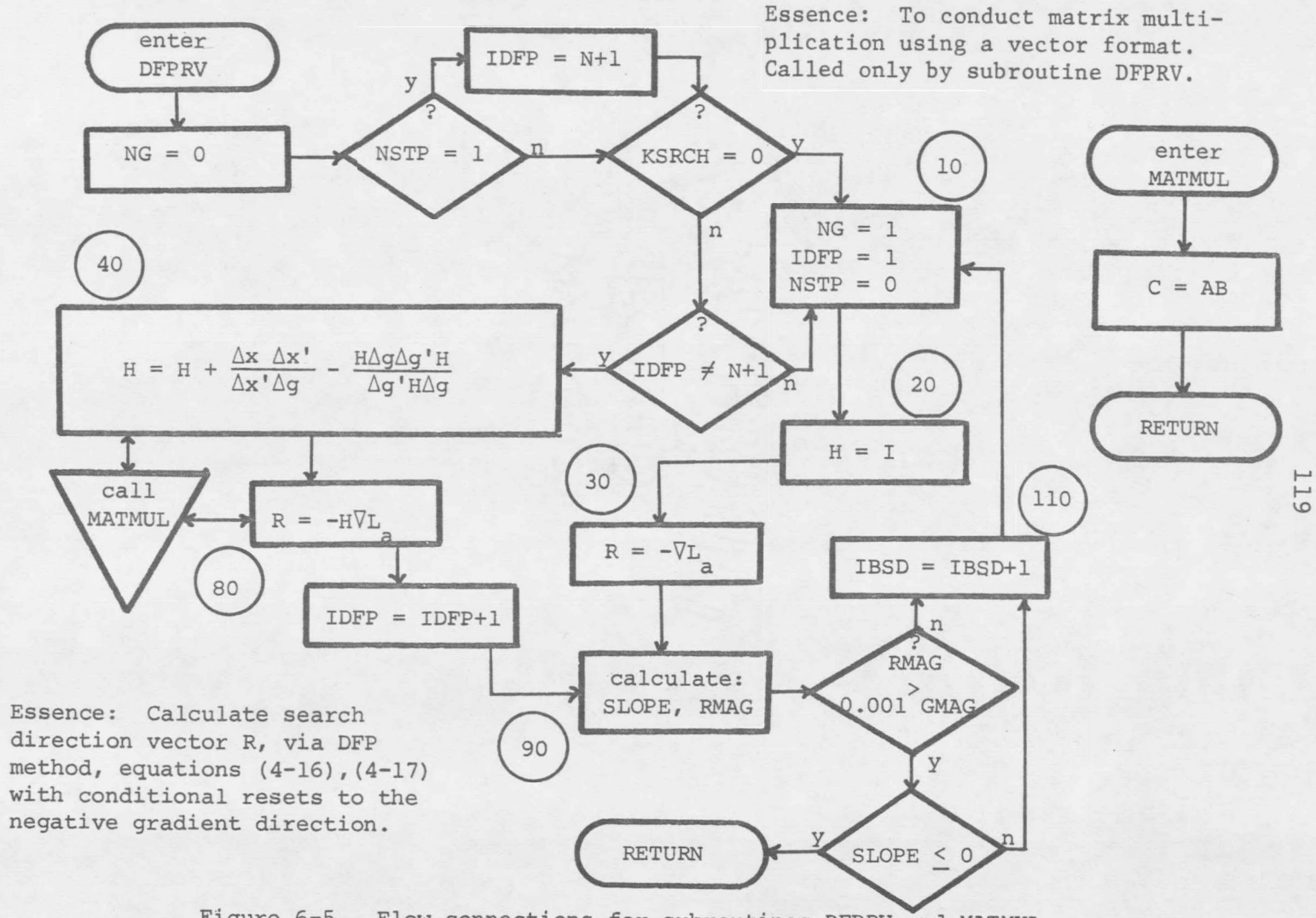
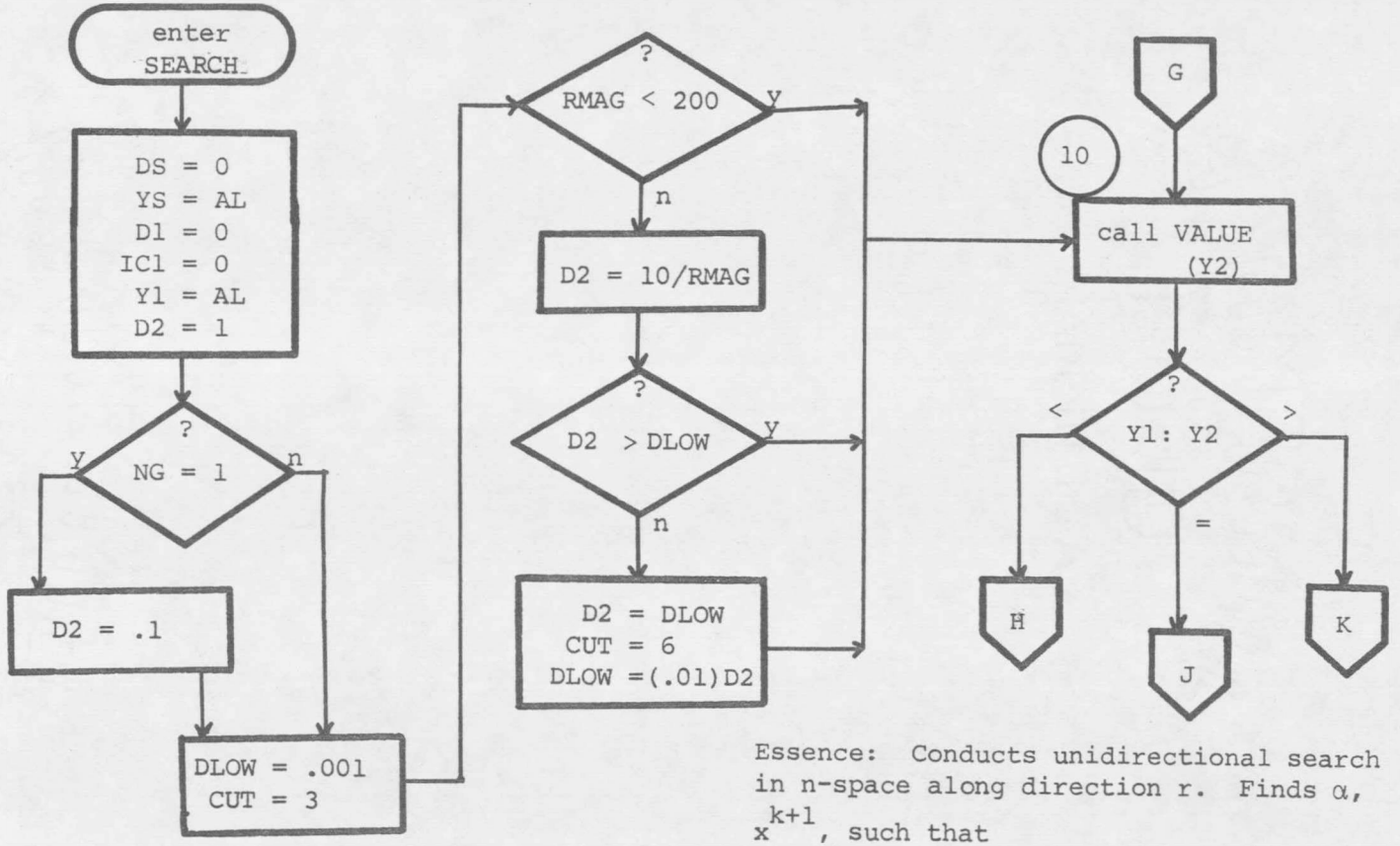


Figure 6-5. Flow connections for subroutines DFPRV and MATMUL.

gradient direction. Thus, if either condition xi or xii is not satisfied, a poor direction has been generated. Therefore, IBSD is incremented (110) and R is set to the negative gradient by returning to block (10). If both conditions xi and xii are satisfied, the generated R is returned as the direction for the next search.

6.6. Subroutine SEARCH

This subroutine conducts the quadratic unidirectional search that is presented in Section 5.4. The flow connection for subroutine SEARCH is shown in Figure 6-6. Initially D1 = 0 and Y1 = AL correspond to the minimum of the last search. This point is saved as the initial best point of the present search. The slope is also known to be negative at D1 = 0 in the R direction given by DFPRV. An initial stepsize D2 is assigned a value of 1 if R is a conjugate gradient direction or a value of 0.1 if R is a negative gradient direction. The stepsize D2 may be modified as shown in Figure 6-6, and a lower bound (DLOW) and a stepsize reduction factor (CUT) are established for the value of D2 finally selected for the initial step. The parametric equation $y(\alpha) = L_a(x^k + \alpha r^k)$, where $y(0) = Y1$, is then evaluated at $\alpha = D2$ to yield Y2 in (10). The value Y2 is then compared to Y1 resulting in one of three



Essence: Conducts unidirectional search in n-space along direction r. Finds α, x^{k+1}, such that

$$L_a(x^{k+1}) = \min_{\alpha} L_a(x^k + \alpha r^k).$$

Figure 6-6. Flow connection for Subroutine SEARCH.

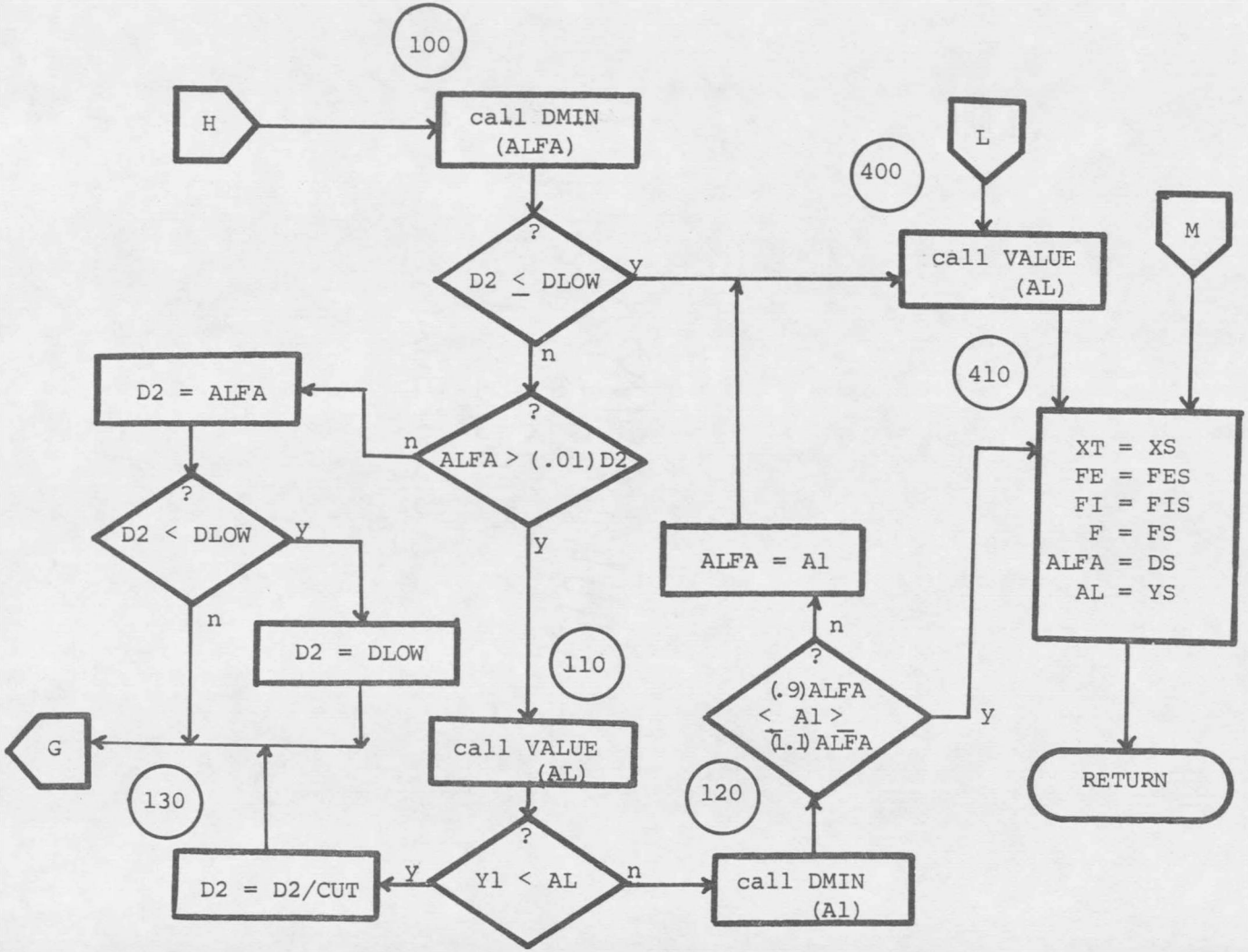


Figure 6-6. (continued)

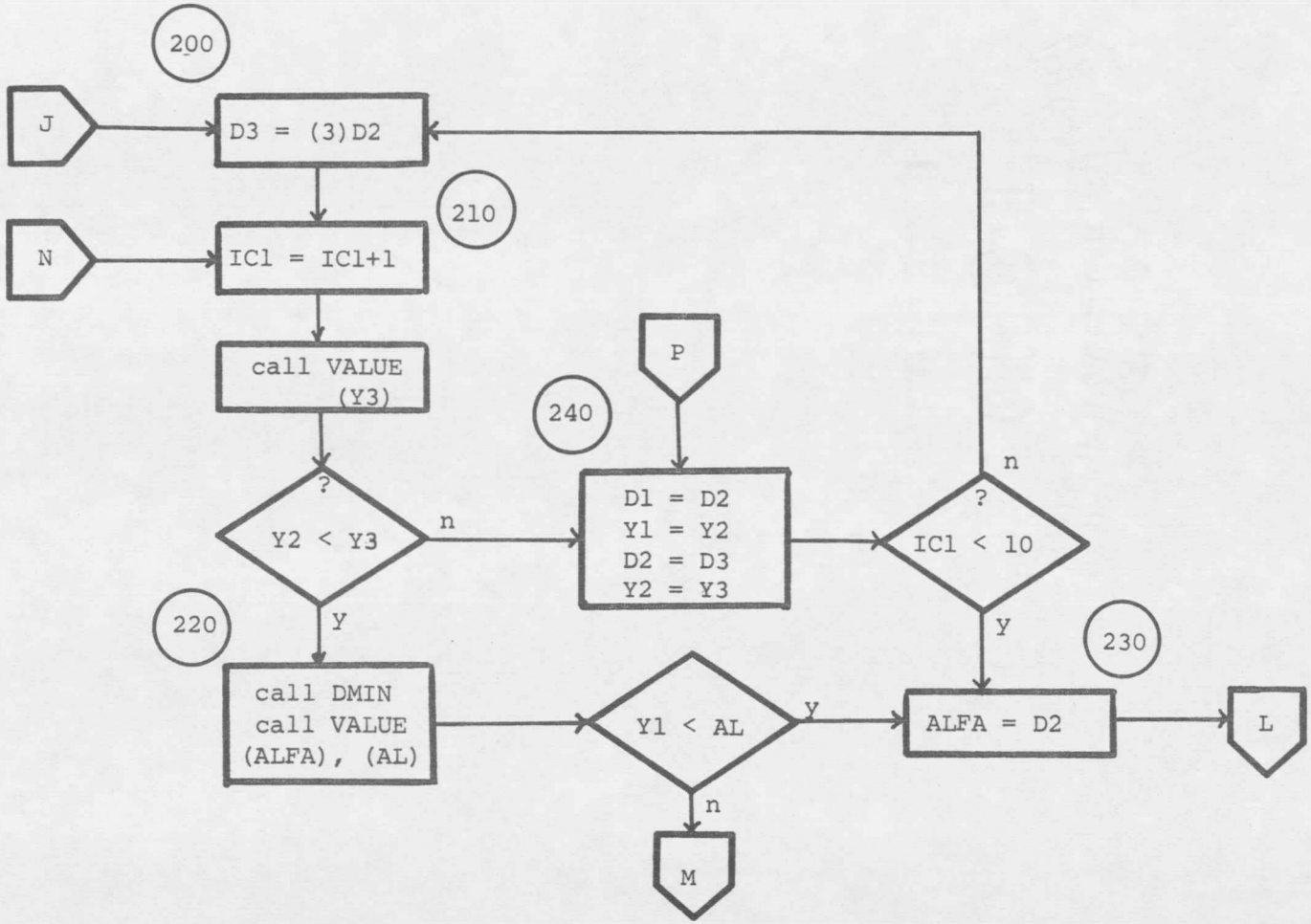


Figure 6-6. (continued)

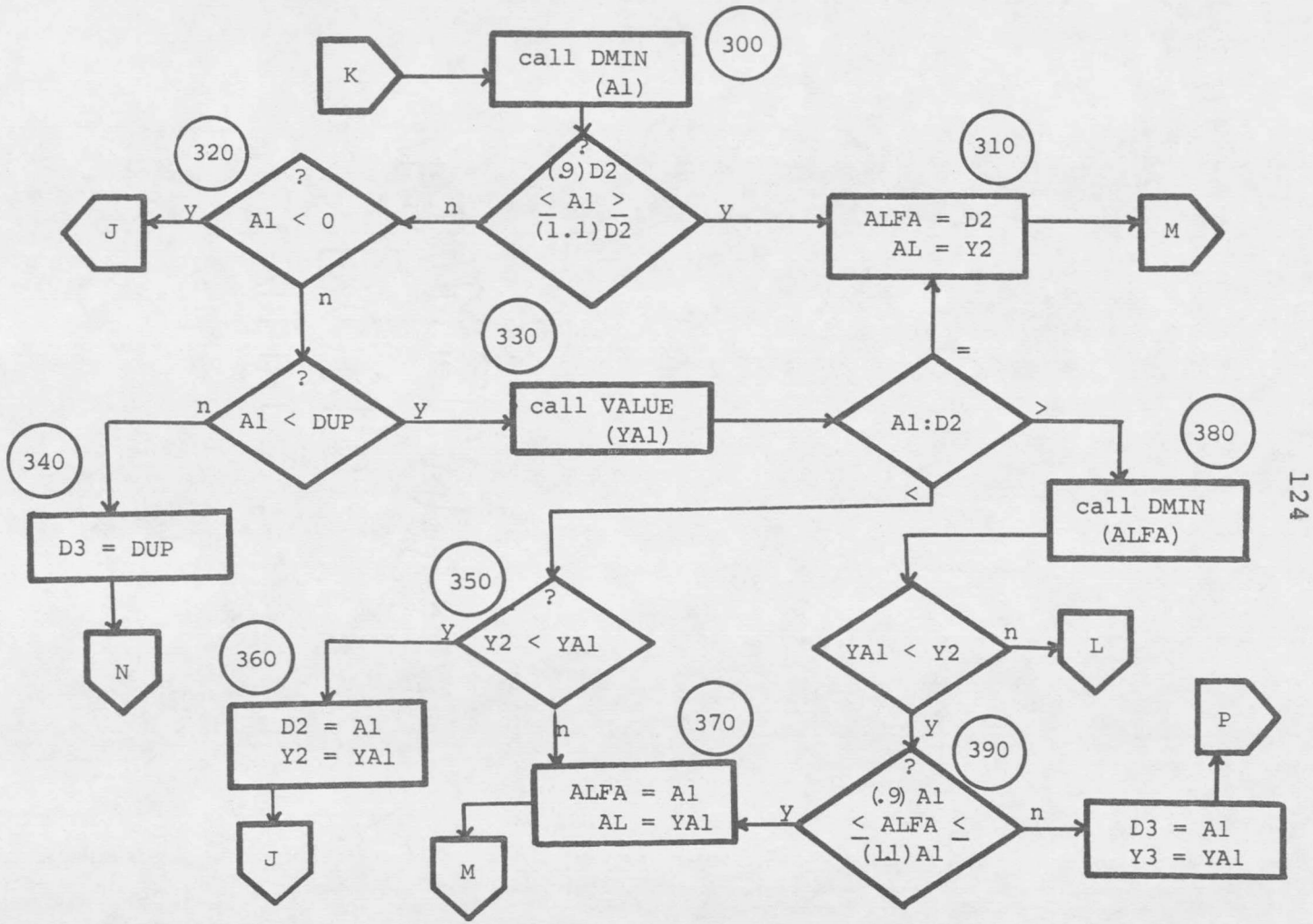


Figure 6-6. (continued)

cases: Case 1) Y1 < Y2; Case 2) Y1 = Y2; and Case 3) Y1 > Y2.

Case 1. The estimated quadratic minimum point ALFA of L_a is obtained by a call to DMIN which finds ALFA using a point-slope-point (PSP) quadratic fit of the values Y1, slope, and Y2 (100). If D2 has been reduced to or below the lower bound DLOW then L_a is evaluated at ALFA (400) and either ALFA or the best point of the search is returned (410) to ALGØR. If D2 is greater than DLOW and ALFA is less than or equal to (0.01) D2, then D2 is reduced to the largest of ALFA and DLOW, Y(D2) is again evaluated (10) and compared to Y1 as above. Otherwise Y(ALFA) = AL is evaluated, and AL is compared to Y1. If AL is greater than Y1, D2 is reduced (130) and the search continues at (10). If AL is less than or equal to Y1, an accuracy check is made on the local quadratic approximation as follows. An estimate A1 of the minimum point is obtained from DMIN (120) which estimates A1 by a point-point-point (3P) quadratic fit of the values of Y1, Y2, AL. If AL is within the ± 10% window of the previous estimate, ALFA, then ALFA or the best point of the search is returned (410). If A1 is not within the window, ALFA is set equal to A1, AL is evaluated (400), and the best point returned (410).

Case 2. A step D3 = (3)D2 is taken and Y3 is evaluated and compared to Y2. If Y3 is not greater than Y2, the search steps are reassigned as in (240), and a step D3 = (3)D2 is again taken. This process is continued for a maximum of 10 steps at which time the best point of the search is returned through blocks (230) and (400). If during this process a value of Y3 is found which is greater than Y2, the values Y1, Y2, and Y3 are passed to DMIN (220), which estimates the minimum ALFA by a 3P quadratic fit. AL is evaluated, and the best point of the search is returned.

Case 3. The quadratic approximation is tested by obtaining an estimate of the minimum A1, found from a PSP fit in DMIN (300). If A1 is within the ±10% window of D2, D2 or the best point is returned through (310). If A1 is negative (320), a step of D3 is taken (200) as in case 2. If A1 is positive and less than DUP then, after YA1 is evaluated (330), A1 is compared to D2. If A1 is less than D2, Y2 and YA1 are compared. If YA1 is less than or equal to Y2, A1 or the best point is returned through (370), otherwise D2 is reassigned as in (360) and the search continues under case 2 at (200). If A1 is greater than D2, a 3P quadratic fit of Y1, Y2, and YA1 is made (380) to

estimate ALFA. If YA1 is greater than Y2, ALFA or the best point is returned (400). If ALFA is within the ±10% window of A1, A1 or the best point is returned through (370), otherwise D3 is reassigned and the search continues under case 2 at (240).

In several situations in this subroutine a test is made to see if an estimated minimum is within ±10% of a point already evaluated. If it is, the evaluated point is returned. This reduces the number of model evaluations for each unidirectional search. Whenever a point is evaluated, it is compared to the best point of the search. If this new point has a smaller value of L_a, it becomes the best point. Since the minimum point of the last search is saved initially, we are guaranteed that the minimum of each search is less than or equal to the value obtained in the previous search. Thus, the sequence $\ldots L_a(x^{k-1}), L_a(x^k), L_a(x^{k+1}) \ldots$ is a monotonic decreasing sequence for each cycle. At the completion of each cycle, the multipliers and weights are updated. If the searches of each cycle have been effective, in that the multipliers are converging to their optimum values, and if there is at least one value of L_a in each cycle that is smaller than

the smallest value of L_a in the previous cycle, the sequence $\{L_a(x^k)\}$ tends to a constrained minimum of the problem as the sequence $\{x^k\}$ tends to x^*.

6.7. Subroutines VALUE and DELTA

Subroutine VALUE is used by subroutine SEARCH in performing a unidirectional search. A flow connection for this subroutine is given in Figure 6-7. This subroutine evaluates the problem functions, by a call to FXNS, at a test point XT, which is a given stepsize D away from the initial point of the search X, along the search direction R (5). The augmented Lagrangian L_a(AL) is formed by calling subroutine AUGLAG, and L_a is compared to the best point of the search YS. If L_a is a better point, it becomes the new best point of the search, and the point XT (10) is stored along with the problem function values at XT, and the values of F, D, and Y (50).

Subroutine DELTA is called by subroutine ALGØR when the best minimum point is found by SEARCH. The flow connection for subroutine DELTA is given in Figure 6-7. DELTA updates the search point status for the new minimum and calculates DELX, DELG, and GMAG which are used by ALGØR for testing convergence and updating criteria.

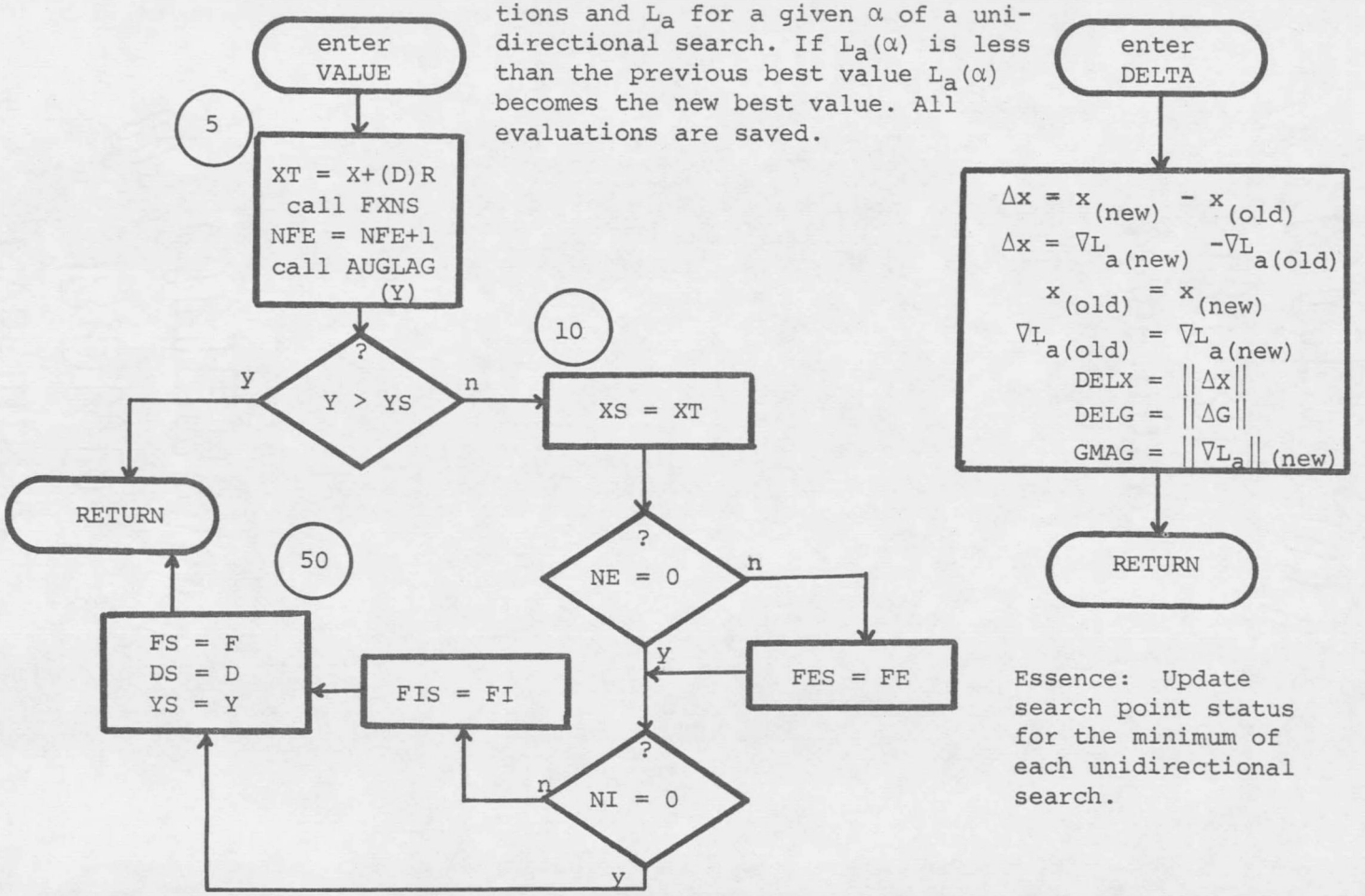


Figure 6-7. Flow connections for Subroutines VALUE and DELTA

6.8. Subroutine DMIN

Subroutine DMIN estimates the minimum step α by making a quadratic fit of 3 data values along the search direction. Depending upon the information available, either a point-slope-point fit is made (10), or a 3-point fit is made (20) as shown in Figure 6-8.

6.9. Subroutine AUGLAG

This subroutine is called by INITL in initializing the algorithm, by SEARCH in conducting a unidirectional search, and by ALGØR after a multiplier and/or weight update. Its purpose is to formulate the augmented Lagrangian L_a(AL) at a given point. For this purpose, it uses the problem function values, multipliers, and weights at that point, as in equation (2-22). A flow connection for this subroutine is given in Figure 6-9.

6.10. Subroutine GALAG

This subroutine is called by INITL in initializing the algorithm and by ALGØR after the best point minimum is found in SEARCH. As seen from the flow connection given in Figure 6-10, its purpose is to formulate the gradient ∇L_a(GAL) of the augmented Lagrangian at this point

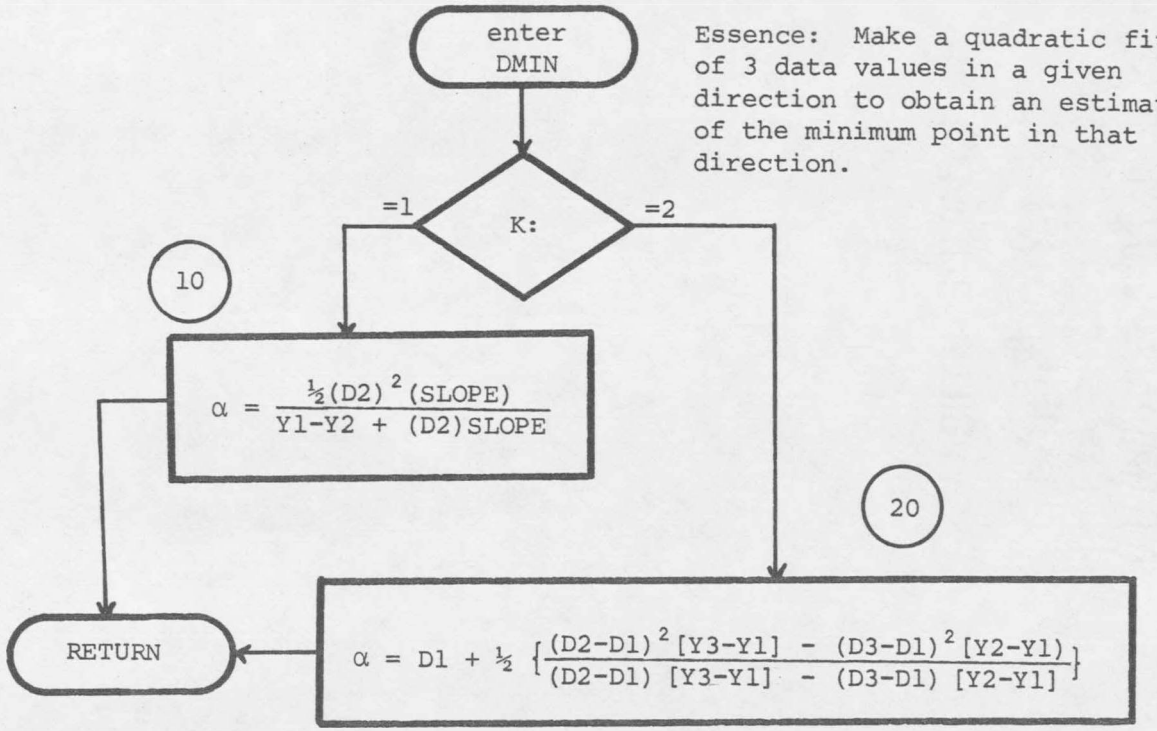


Figure 6-8. Flow connection for Subroutine DMIN.

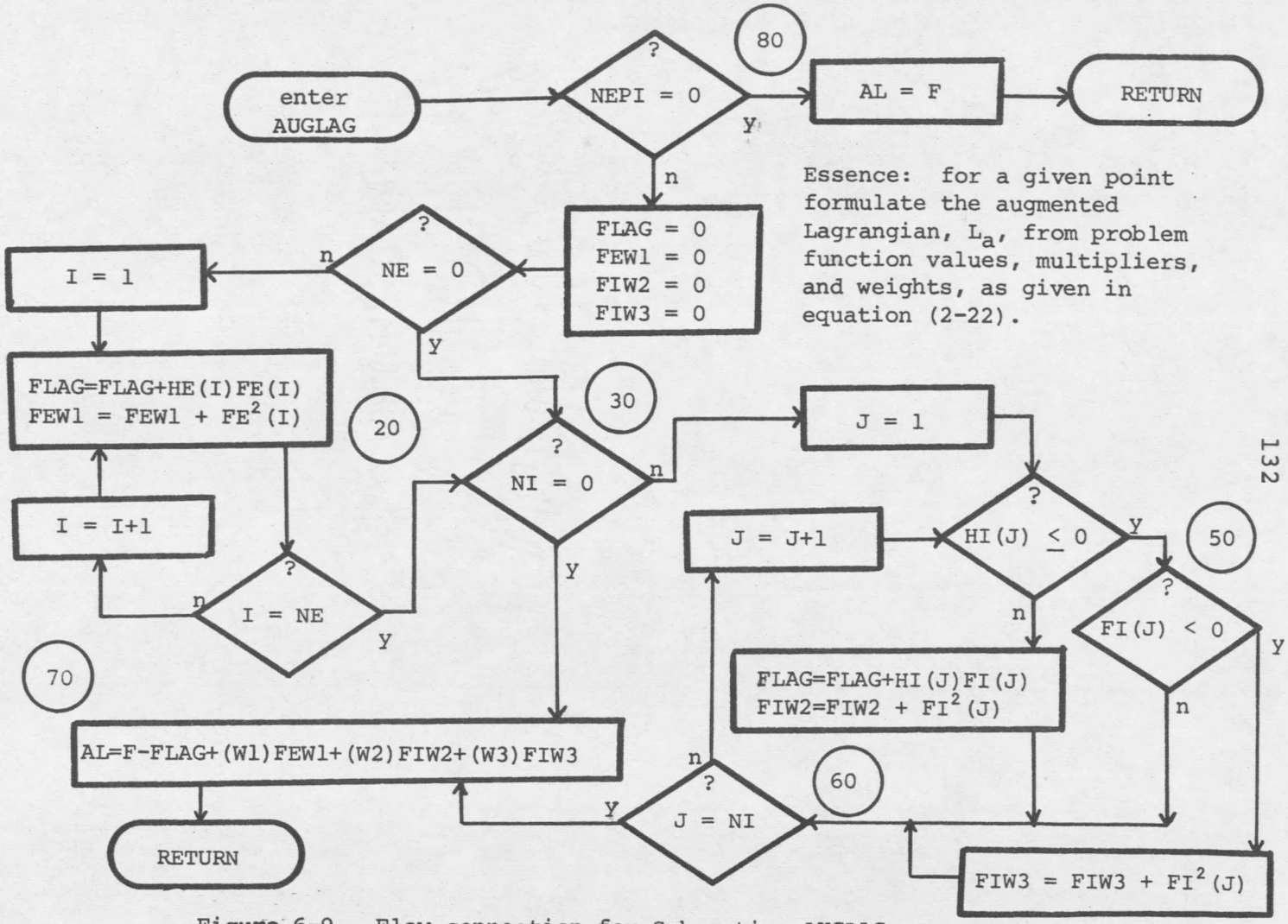


Figure 6-9. Flow connection for Subroutine AUGLAG.

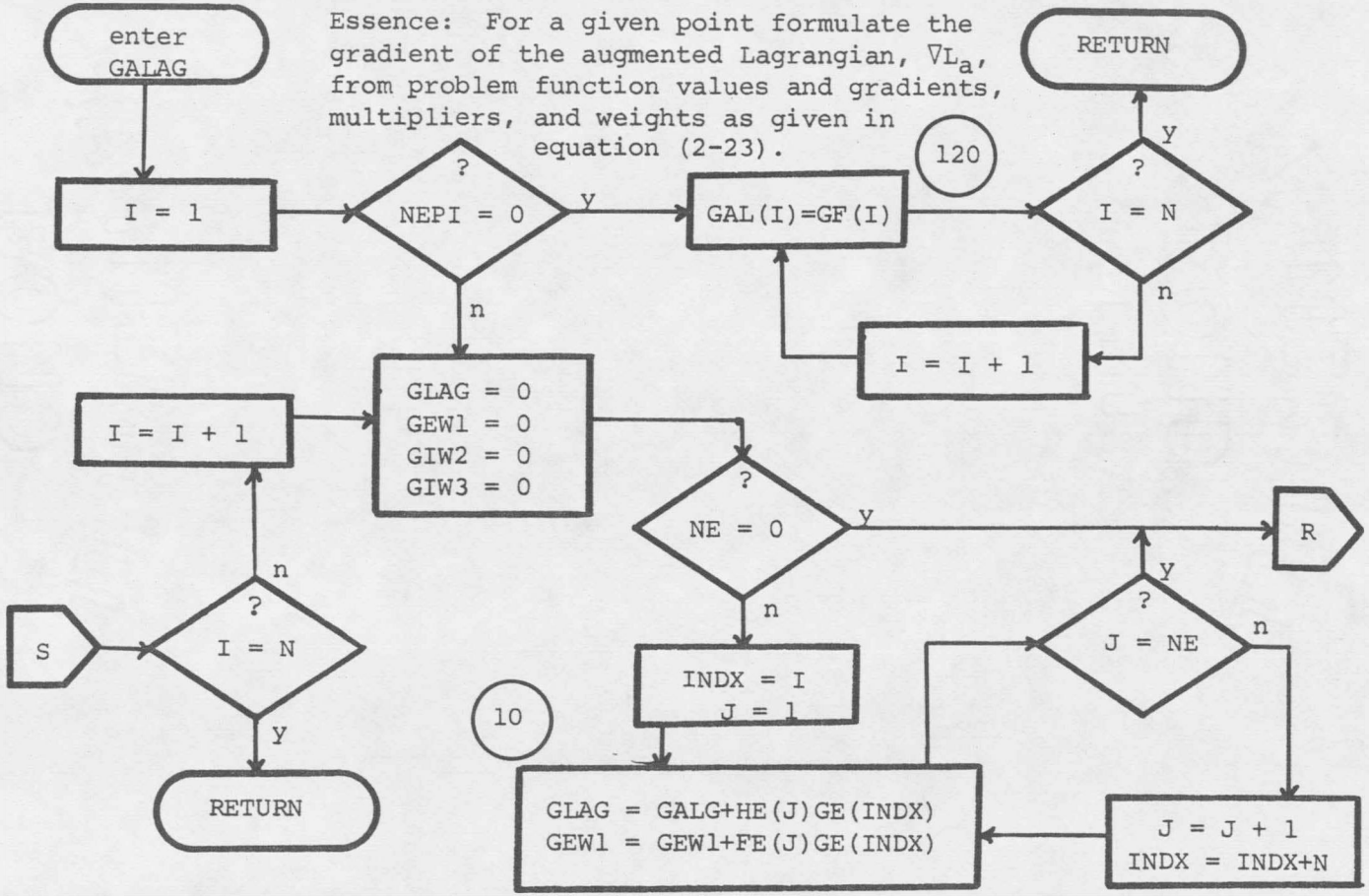


Figure 6-10. Flow connection for Subroutine GALAG.

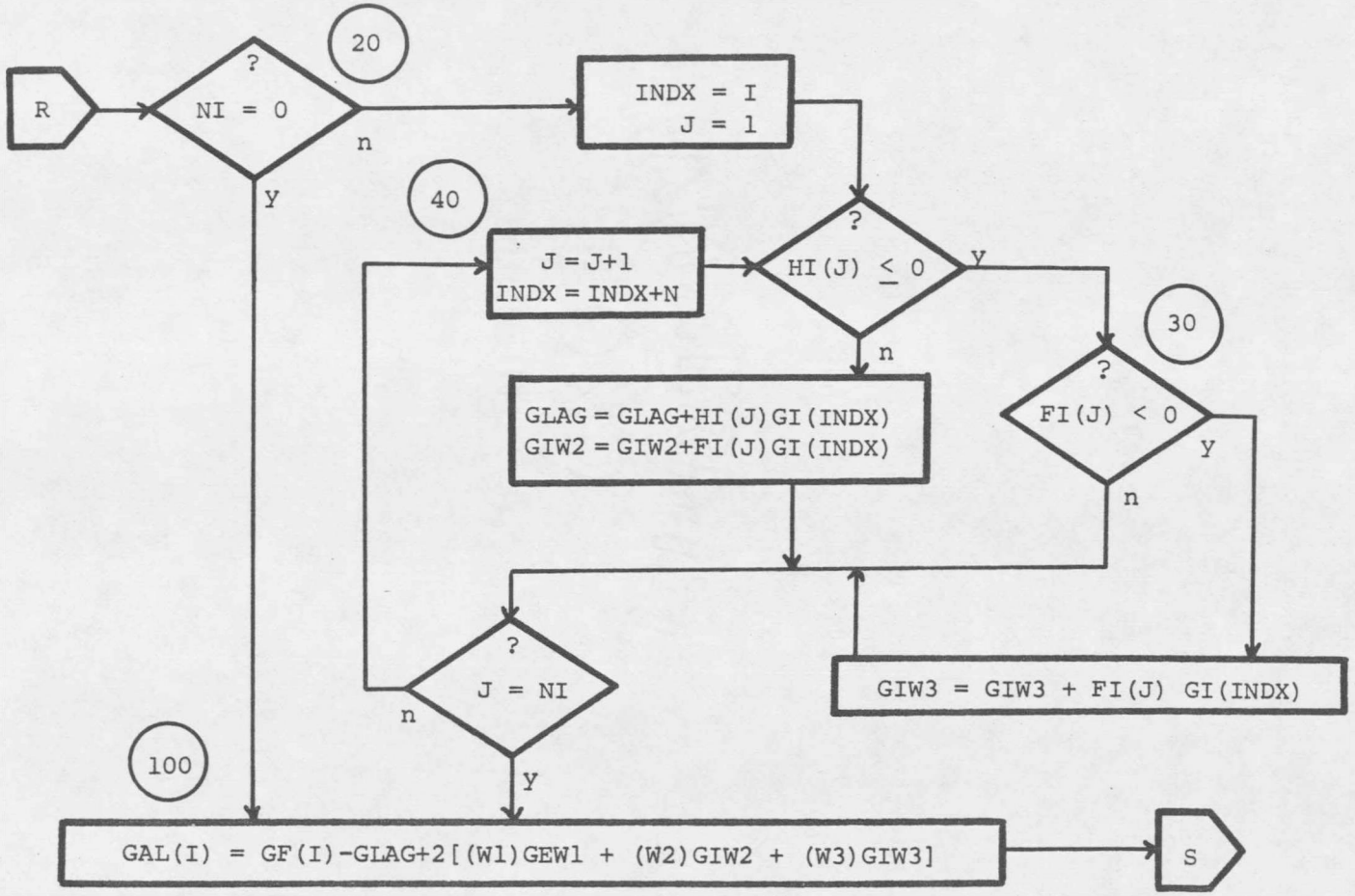


Figure 6-10. (continued)

from problem function values, problem function gradients, multipliers, and weights as in equation (2-23).

6.11. Subroutine MULTUP

The flow connection for this subroutine is given in Figure 6-11. The purpose of this subroutine is to establish the constraint basis for each search cycle by updating the multipliers according to the multiplier rules given in equations (2-43), (2-44), and (2-45). The new multipliers are then printed along with the problem function values at the point of update.

6.12. Subroutine WUP

This subroutine updates the penalty weights according to the weight update rule, equation (2-46). A flow connection is given in Figure 6-12. If the algorithm is used in the multiplier mode, or in the penalty/multiplier mode, this algorithm initializes the multiplier weights (20) when the multiplier phase is introduced.

6.13. Subroutine OUTPUT

A flow connection for this subroutine is given in Figure 6-13. This subroutine prints the general search point information for the minimum of a unidirectional

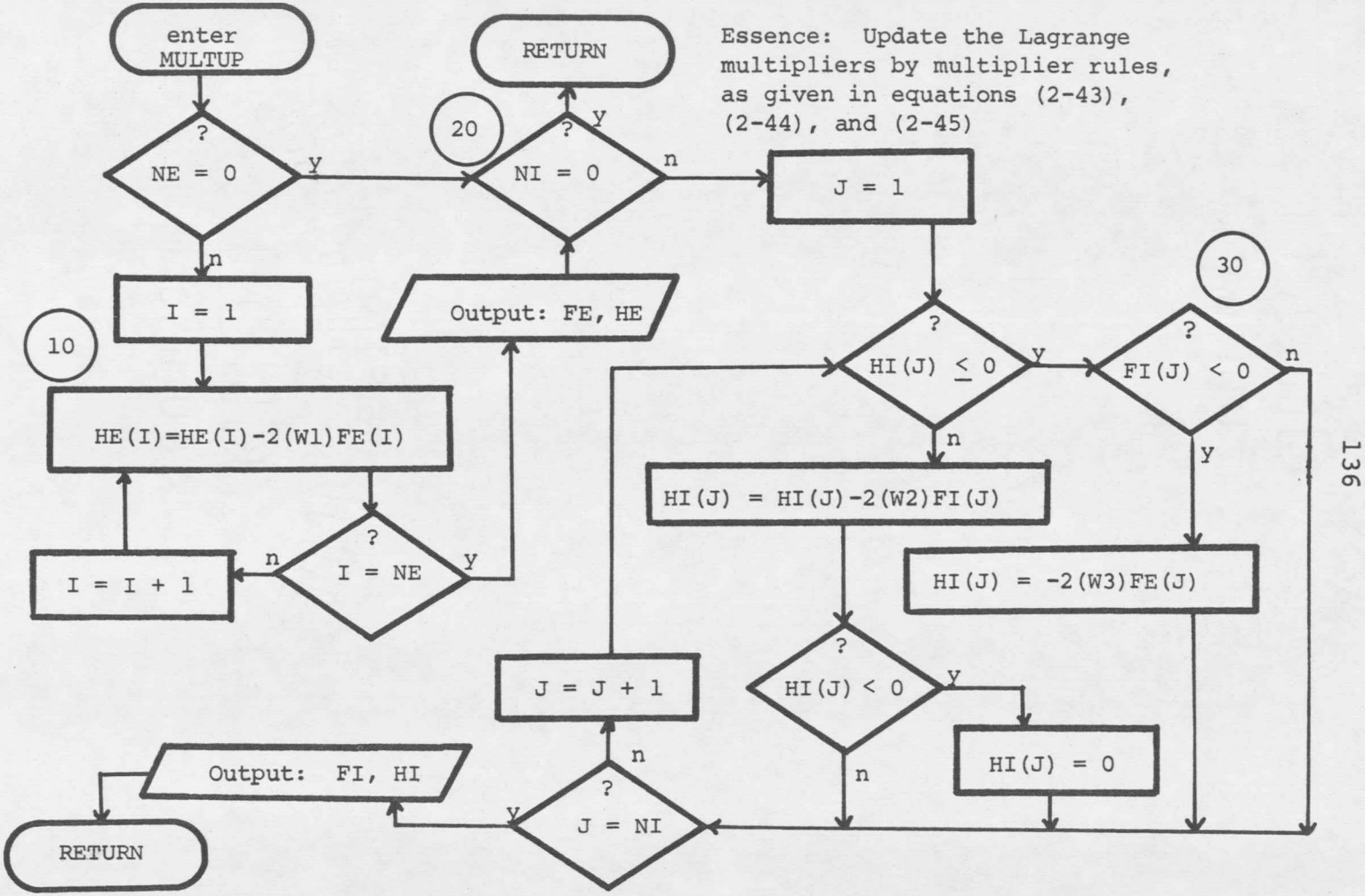


Figure 6-11. Flow connection for Subroutine MULTUP.

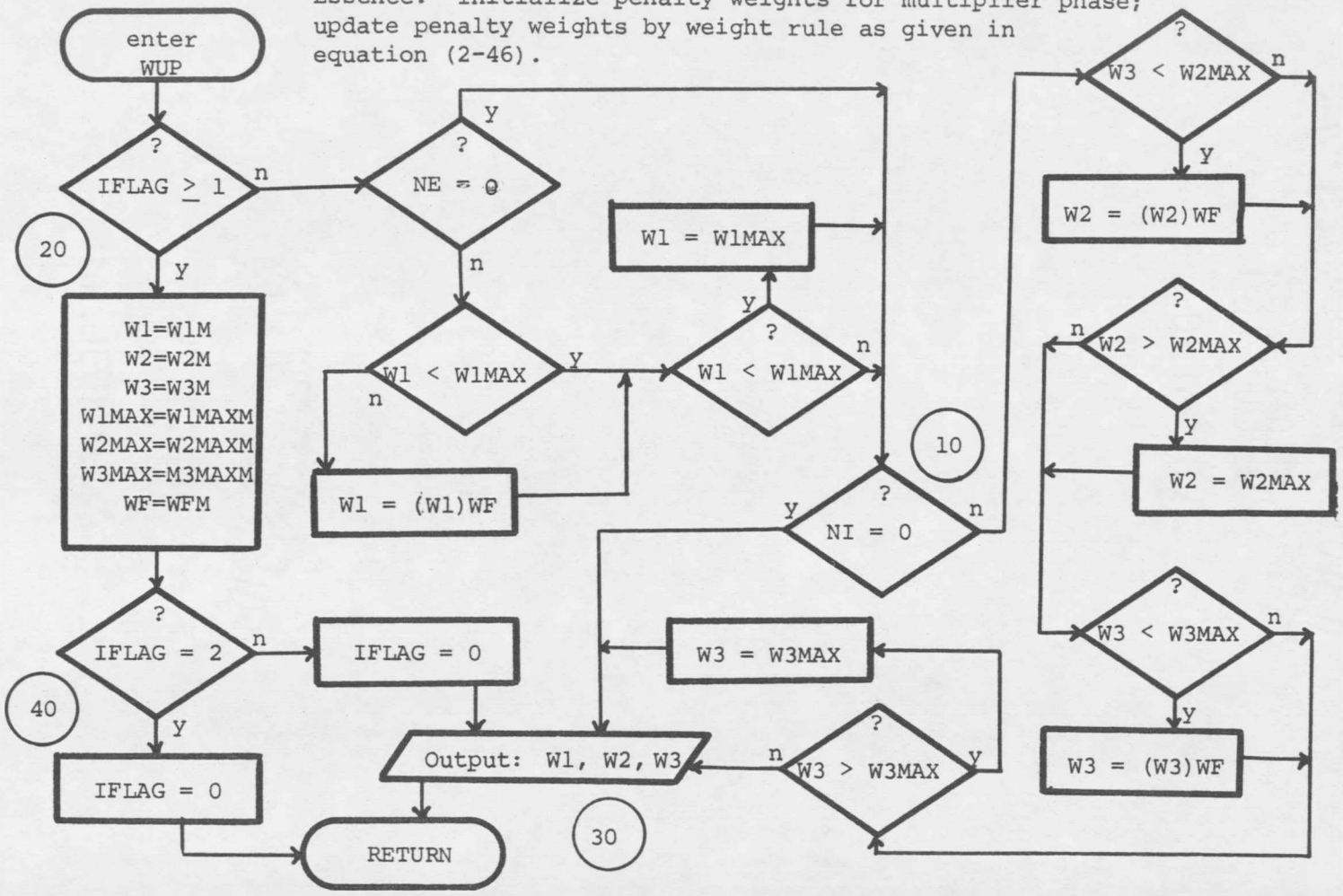


Figure 6-12. Flow connection for Subroutine WUP.

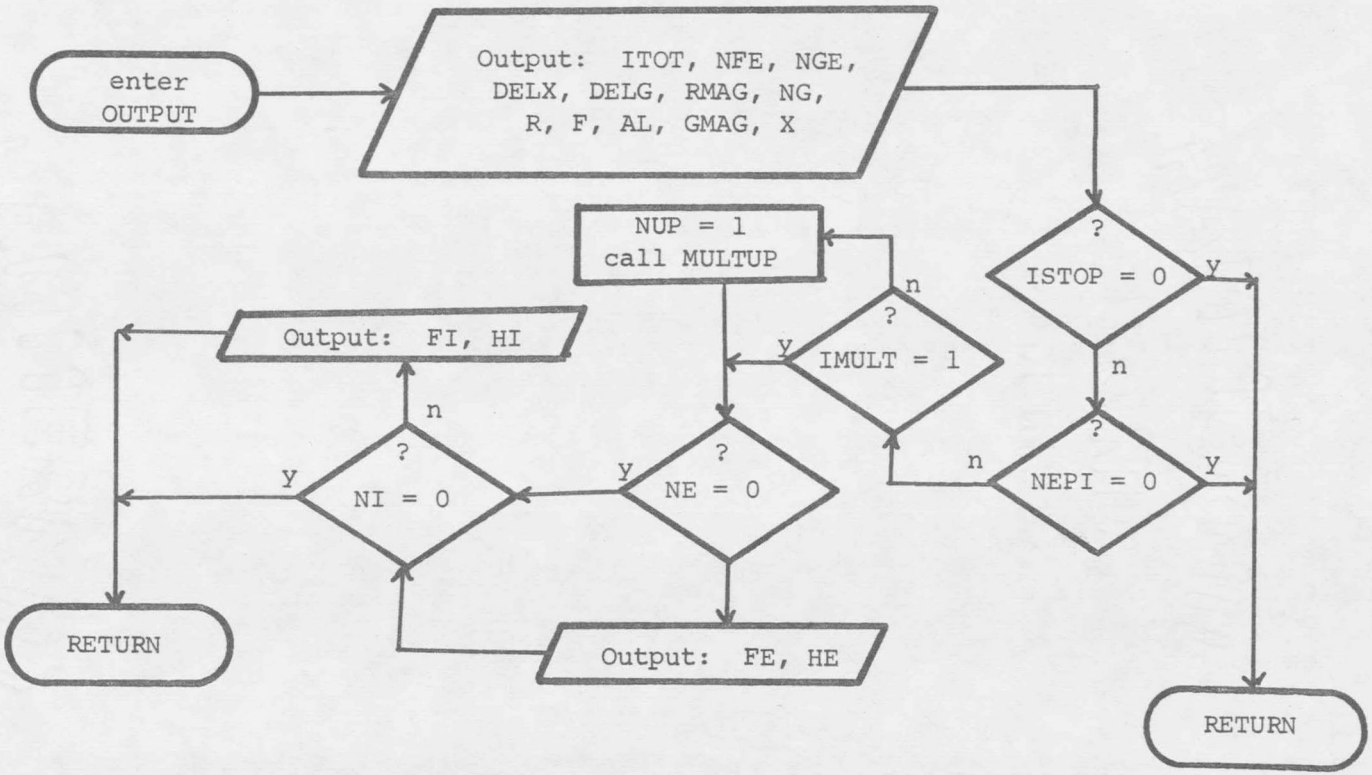


Figure 6-13. Flow connection for Subroutine OUTPUT.

search. At the termination of the algorithm (ISTOP=1), this subroutine calls MULTUP (only if the algorithm has operated in the straight penalty-function mode such that IMULT = 0) to get an estimate of the multipliers. The final values of the problem functions and the values of their associated multipliers are also printed as terminal information.

CHAPTER SEVEN

TEST PROBLEMS

7.1. Introduction

In this chapter the multiplier algorithm, ALGØR, is applied to several standard test problems, and the solutions are compared to those obtained by other methods which solve the NLP. Examples of the formulation of the NLP from selected real-world problem situations, and the solutions of these problems, are presented in Chapter 8.

Execution time to convergence or termination of the algorithms is a good standard of comparison, but is a function of the computer language and the type of computer which is used for solution, both of which vary significantly. Therefore, the comparison is made on the basis of the number of function evaluations required to reduce $|f(x) - f(x^*)|$ to a desired degree of accuracy. The solutions obtained for the problems presented here are the results of double-precision calculations on the XDS Sigma-7 at Montana State University. All results are from ALGØR operating in the straight multiplier mode.

7.2. Test Problems and Results

T1: Banana Valley (Rosenbrock [65])

Rosie (Banana Valley) is a widely used test function for nonlinear programming routines. This unconstrained

function has a steep (banana shaped) valley along $x_2 = x_1^2$.

To be minimized is

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

The optimum point is at $x^* = (1, 1)$ at which $f(x^*) = 0$.

The standard starting point is at $x = (-1.2, 1)$. The parameter settings for the algorithm are $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = \varepsilon_3 = 10^{-6}$, and nsrch = 6. The function $f(x)$ was reduced to below 10^{-5} in 142 function calls and 38 gradient calls, and converged to the exact optimum point in 152 function calls and 44 gradient calls. The user supplied routines, data, and computer results are given for this problem in Appendix C.4.

T2: Around the World (Pierre [55])

The problem is to find the minimum of f on a sphere, subject to remaining on one side of a plane that passes through the sphere.

Minimize

$$f = -x_2$$

subject to

$$x_1^2 + x_2^2 + x_3^2 - 1 = 0$$

$$1 + x_1 - 2x_2 \geq 0$$

The optimum point is $x^* = (0.6, 0.8, 0)$ at which $f(x^*) = -0.8$, $\alpha^* = -0.25$, and $\beta^* = 0.3$.

For a typical solution, the algorithm was initialized as follows: $x = (-0.1, -1.0, 0.1)$, $\varepsilon_1 = 10^{-6}$, $\varepsilon_2 = \varepsilon_3 = 10^{-4}$, $\alpha = \beta = 0$, $w_1 = w_2 = w_3 = 0.25$, $w_{imax's} = 1$, wf = 2, and nsrch = 7. Four place accuracy in $f(x^*)$ was obtained after 42 function calls and 19 gradient calls, and converged to the optimum in 65 function calls and 29 gradient calls. For the same initial values, Pierre's method obtained four place accuracy in $f(x^*)$ after 87 function calls and 17 gradient calls. The user supplied routines, data, and computer results are given for this problem in Appendix C.5.

Pierre also notes that a saddle point x_s exists at $x_s = (-1, 0, 0)$ at which $f(x_s) = 0$, $\alpha = 0.25$, and $\beta = 0.5$. In one test case with the initial $x_3 \equiv 0$, his solution converged to the saddle point. A slight perturbation away from the saddle (initial $x = (-1.0, 0, 0.05)$), however,

caused the solution to converge to the optimum. This test case was not investigated for this thesis.

T3: A Linear Problem (Pierre [55])

Minimize

$$f = -0.5x_1 - x_2 - 0.5x_3 - x_4$$

subject to

$$x_1 + x_2 + x_3 - 2x_4 - 6 = 0$$

$$10 - \sum_{i=1}^{4} x_i \geq 0$$

$$10 - 0.2x_1 - 0.5x_2 - x_3 - 0.2x_4 \geq 0$$

$$x_i \geq 0, \; i = 1, 2, 3, 4$$

The optimum point is at $x^* = (0, 26/3, 0, 4/3)$ at which $f(x^*) = -10$, $\alpha^* = 0$, and $\beta^* = (1, 0, 0, 0.5, 0, 0.5, 0)$, where the last four components of β are the multipliers associated with $x_i \geq 0$, $i = 1, 2, 3, 4$.

For a typical solution the algorithm was initialized as follows: $x_i = 0$, $i = 1, 2, 3, 4$, $\varepsilon_1 = 10^{-6}$, $\varepsilon_2 = \varepsilon_3 = 10^{-4}$, $\alpha = 0$, $\beta_j = 0$, $j = 1, 2, \ldots, 7$, $w_1 = w_2 = w_3 = 1$, $w_{imax's} = 16$, wf = 2, and nsrch = 9. The optimum was obtained after 50 function calls and 22

gradient calls. For the same initial values, Pierre's method obtained the optimum after 118 function calls and 23 gradient calls.

T4: A Seven Variable Problem (Pierre [55])

Minimize

$$f = -5x_1 - 5x_2 - 4x_3 - x_1x_3 - 6x_4 - \frac{5x_5}{1 + x_5}$$

$$- \frac{8x_6}{1 + x_6} - 10(1 + 2e^{-x_7} - e^{-2x_7})$$

subject to

$$2x_4 + x_5 + 0.8x_6 + x_7 - 5 = 0$$

$$x_2^2 + x_3^2 + x_5^2 + x_6^2 - 5 = 0$$

$$10 - \sum_{i=1}^{7} x_i \geq 0$$

$$5 - \sum_{i=1}^{4} x_i \geq 0$$

$$5 - x_1 - x_3 - x_5 - x_6^2 + x_7^2 \geq 0$$

$$x_i \geq 0, \quad i = 1, 2, \ldots, 7$$

A constrained local minimum exists at x^* = (3.24182, 0, 1.63416, 0.124021, 0.889614, 1.24021, 2.87018) at which $f(x^*) = -44.4687$, α^* = (0.317079, -0.185926), and β^* = (1.38658, 5.24758, 0, 0, 1.63416, 0,

0, 0, 0, 0), where the last seven components of β are the multipliers associated with $x_i \geq 0$, $i = 1, 2, \ldots, 7$.

For a typical solution the algorithm was initialized as follows: $x_i = 0.1$, $i = 1, 2, \ldots, 7$, $\varepsilon_1 = 10^{-4}$, $\varepsilon_2 = \varepsilon_3 = 10^{-5}$, $\alpha_i = 0$, $i = 1, 2$, $\beta_j = 0$, $j = 1, 2, \ldots, 10$, $w_1 = w_2 = w_3 = 1$, $w_{imax's} = 32$, wf = 4, and nsrch = 15. Four place accuracy in $f(x^*)$ was obtained after 131 function calls and 52 gradient calls, and converged to the minimum in 179 function calls and 76 gradient calls. For the same initial values, Pierre's method obtained four place accuracy in $f(x^*)$ after 212 function calls and 45 gradient calls.

The following test functions were found in a computational comparison study of some nonlinear programs [5]. Three sequential unconstrained minimization methods, OPTKOV, POWCON, and SUMT, were compared on the basis of computer execution time and number of function evaluations. All three methods were written in the same language, ICL ALGOL 60 and run on the same computer. For the test problems listed here, the conclusions drawn from that study were; considering function evaluations, POWCON was

the best method; considering execution time, SUMT was the best method; OPTKOV was relatively inefficient considering both number of function evaluations and runtime. The convergence test criteria for these three methods are not indicated in [5], and therefore, the comparison with ALGØR is based on the number of function evaluations required to obtain a desired degree of accuracy in f(x). The same starting points were used for each test problem. Function evaluations for OPTKOV, POWCON, AND SUMT are obtained from a table in [5].

T5: (suggested by Fiacco and McCormick)

Minimize

$$f = \frac{1}{3}(x_1 + 1)^3 + x_2$$

subject to

$$x_1 - 1 \geq 0$$

$$x_2 \geq 0$$

This function has an optimum at $x^* = (1, 0)$, at which $f(x^*) = 8/3$, and $\beta^* = (4, 1)$.

The algorithm was initialized as follows:
$x = (1.125, 0.125)$, $\varepsilon_1 = 10^{-6}$, $\varepsilon_2 = \varepsilon_3 = 10^{-4}$, $\beta = (0, 0)$,

$w_1 = w_{1\,max} = 0$, $w_2 = w_3 = 1$, $w_{2\,max} = w_{3\,max} = 64$, wf = 4, nsrch = 5. Five place accuracy in $f(x^*)$ was obtained after 46 function calls and 21 gradient calls, and converged to the optimum in 88 function calls and 29 gradient calls. Five place accuracy in $f(x^*)$ was obtained by OPTKOV after 103 function calls, by POWCON after 134 function calls, and by SUMT after 181 function calls.

T6: (Rosen and Suzuki's problem)

Minimize

$$f = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$$

subject to

$$-\left(\sum_{i=1}^{4} x_i^2\right) - x_1 + x_2 - x_3 + x_4 + 8 \geq 0$$

$$-x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 10 \geq 0$$

$$2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5 \geq 0$$

A constrained local minimum exists at $x^* = (0, 1, 2, -1)$ at which $f(x^*) = -44$, and $\beta^* = (1, 0, 2)$.

The algorithm was initialized as follows:

$x = (0, 0, 0, 0)$, $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 10^{-6}$, $\beta = (0, 0, 0)$, $w_1 = w_{1\,max} = 0$, $w_2 = w_3 = 1$, $w_{2\,max} = w_{3\,max} = 16$, wf = 4,

and nsrch = 9. Five place accuracy in $f(x^*)$ was obtained after 79 function calls and 32 gradient calls, and converged to the minimum in 162 function calls, and 63 gradient calls. Five place accuracy in $f(x^*)$ was obtained by OPTKOV after 198 function calls, and by SUMT after 175 function calls. POWCON stopped prematurely after 68 function calls at the value of f = -44.9307, obtaining only two place accuracy.

T7: (Beale's problem)

Minimize

$$f = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3$$

subject to

$$-x_1 - x_2 - 2x_3 + 3 \geq 0$$

$$x_i \geq 0, \quad i = 1, 2, 3$$

This function has a constrained local minimum at the point $x^* = (4/3, 7/9, 4/9)$ at which $f(x^*) = 1/9$, and $\beta^* = (2/9, 0, 0, 0)$, where the last three components of β are the multipliers associated with $x_i \geq 0$, $i = 1, 2, 3$.

The algorithm was initialized as follows:

$x = (0.5, 0.5, 0.5)$, $\varepsilon_1 = 10^{-6}$, $\varepsilon_2 = \varepsilon_3 = 10^{-5}$,

$\beta = (0, 0, 0)$, $w_1 = w_{1_{max}} = 0$, $w_2 = w_3 = 1$, $w_{2_{max}} = w_{3_{max}} = 16$, wf = 4, and nsrch = 7. Five place accuracy in $f(x^*)$ was obtained after 33 function calls and 17 gradient calls, and converged to the minimum in 51 function calls and 25 gradient calls. Five place accuracy in $f(x^*)$ was obtained by POWCON in 61 function calls, and by SUMT somewhere after 130 function calls. OPTKOV converged to $f(x) = 0.1111$ in 184 function evaluations after obtaining only four places of accuracy.

T8: (suggested by Powell)

Minimize

$$f = \prod_{i=1}^{5} x_i$$

subject to

$$\sum_{i=1}^{5} x_i - 10 = 0$$

$$x_2 x_3 - 5x_4 x_5 = 0$$

$$x_1^3 + x_2^3 + 1 = 0$$

This function has a constrained local minimum at $x^* = (-1.71714, 1.59571, 1.82725, -0.763643, -0.763643)$ at which $f(x^*) = -2.91970$, and $\alpha^* = (-0.744446, 0.703573, -0.096806)$.

The algorithm was initialized as follows:
$x = (-2, 1.5, 2, -1, -1)$, $\varepsilon_1 = 10^{-6}$, $\varepsilon_2 = \varepsilon_3 = 10^{-5}$,
$\alpha = (0, 0, 0)$, $w_1 = 0.5$, $w_{1_{max}} = 8$,
$w_2 = w_3 = w_{2_{max}} = w_{3_{max}} = 0$, wf = 2, and nsrch = 11.
Five place accuracy in $f(x^*)$ was obtained after 75 function calls and 36 gradient calls, and converged to the minimum in 99 function calls and 45 gradient calls. Five place accuracy in $f(x^*)$ was obtained by OPTKOV after 187 function calls, by POWCON after 90 function calls, and by SUMT after 126 function calls.

Powell [58] notes that some difficulty could occur with constraint breakthrough because f is not bounded from below. Therefore, $f = \exp(x_1x_2x_3x_3x_5)$ is the recommended function to be minimized subject to the same constraints as above. The problem was run using this modified cost function. The algorithm was initialized as above except that $w_{1_{max}} = 16$. The minimum of this problem was located at the same point x^* as above at which $f(x^*) = 0.0539498$, and $\alpha^* = (-0.040163, 0.037958, -0.005223)$. Five place accuracy in $f(x^*)$ was obtained after 46 function calls and 19 gradient calls, and converged to the minimum in 101 function calls and 38 gradient calls.

T9: (suggested by Wong)

Minimize

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

subject to

$$-2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 + 127 \geq 0$$

$$-7x_1 - 3x_2 -10x_3^2 - x_4 + x_5 + 282 \geq 0$$

$$-23x_1 - x_2^2 - 6x_6^2 + 8x_7 + 196 \geq 0$$

$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$$

A constrained local minimum of this function exists at x^* = (2.33050, 1.95137, -0.477541, 4.36573, -0.624487, 1.03813, 1.59423) at which $f(x^*)$ = 680.630, and β^* = (1.13972, 0, 0, 0.368615).

The algorithm was initialized as follows: x = (1, 2, 0, 4, 0, 1, 1), $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 10^{-4}$, β = (0, 0, 0, 0), $w_1 = w_{1_{max}} = 0$, $w_2 = w_3 = 0.25$, $w_{2_{max}} = w_{3_{max}} = 16$, wf = 4, and nsrch = 15. Five place accuracy in $f(x^*)$ was obtained after 124 function calls and 36 gradient calls, and converged to the minimum in 179 function calls and 59 gradient calls. Five places of accuracy in $f(x^*)$ was obtained by OPTKOV after 334 function

calls, and by POWCON at convergence in 123 function calls. SUMT failed to find the minimum in 160 function calls as it stopped prematurely at the value f(x) = 680.739, after obtaining only three places of accuracy.

T10: (suggested by Wong)

Minimize

$$f = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 + 2(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7)^2 + 45$$

subject to

$$-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$$

$$-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 4 \geq 0$$

$$-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$$

$$-x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$$

$$-4x_1 - 5x_2 + 3x_7 - 9x_8 + 105 \geq 0$$

$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$$

$$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0$$

$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$$

This function has a constrained local minimum at x^* = (2.17200, 2.36368, 8.77393, 5.09598, 0.990655, 1.43057, 1.32164, 9.82873, 8.28009, 8.37593) at which

$f(x^*) = 24.3062$, and $\beta^* = (0.020546, 0.312029, 0, 0.287049, 1.71653, 0.474520, 0, 1.37593)$.

The algorithm was initialized as follows: $x = (2, 3, 5, 5, 1, 2, 7, 3, 6, 10)$, $\varepsilon_1 = 10^{-4}$, $\varepsilon_2 = \varepsilon_3 = 10^{-5}$, $\beta_j = 0$, $j = 1, 2, \ldots, 8$, $w_1 = w_{1_{max}} = 0$, $w_2 = w_3 = 0.25$, $w_{2_{max}} = w_{3_{max}} = 4$, wf = 4, nsrch = 21. Four place accuracy in $f(x^*)$ was obtained after 185 function calls and 66 gradient calls, and converged to the minimum in 233 function calls and 85 gradient calls. Four place accuracy in $f(x^*)$ was obtained by OPTKOV after 621 function calls, and by SUMT after 532 function calls. POWCON stopped after 272 function calls at the value $f(x) = 24.30$ after obtaining 3 places of accuracy.

T11: (suggested by Wong)

Minimize

$$f = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 + 2(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7)^2 + (x_{11}-9)^2 + 10(x_{12}-1)^2 + 5(x_{13}-7)^2 + 4(x_{14}-14)^2 + 27(x_{15}-1)^2 + x_{16}^4 + (x_{17}-2)^2 + 13(x_{18}-2)^2 + (x_{19}-3)^2 + x_{20}^2 + 95$$

subject to

$$-3(x_1-2)^2 - 4(x_2-3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$$

$$15x_1^2 - 8x_2 - (x_3-6)^2 + 2x_4 + 40 \geq 0$$

$$-0.5(x_1-8)^2 - 2(x_2-4)^2 - 3x_5^2 + x_6 + 30 \geq 0$$

$$-x_1^2 - 2(x_2-2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$$

$$-4x_1 - 5x_2 + 3x_7 - 9x_8 + 105 \geq 0$$

$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$$

$$3x_1 - 6x_2 - 12(x_9-8)^2 + 7x_{10} \geq 0$$

$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$$

$$-x_1 - x_2 - 4x_{11} + 21x_{12} \geq 0$$

$$-x_1^2 - 15x_{11} + 8x_{12} + 28 \geq 0$$

$$-4x_1 - 9x_2 - 5x_{13}^2 + 9x_{14} + 87 \geq 0$$

$$-3x_1 - 4x_2 - 3(x_{13}-6)^2 + 14x_{14} + 10 \geq 0$$

$$-14x_1^2 - 35x_{15} + 79x_{16} + 92 \geq 0$$

$$-15x_2^2 - 11x_{15} + 61x_{16} + 54 \geq 0$$

$$-5x_1^2 - 2x_2 - 9x_{17}^4 + x_{18} + 68 \geq 0$$

$$-x_1^2 + x_2 - 19x_{19} + 20x_{20} + 19 \geq 0$$

$$-7x_1^2 - 5x_2^2 - x_{19}^2 + 30x_{20} \geq 0$$

This function has a constrained minimum at

$x^* = (2.04697, 2.20954, 8.74453, 5.06281, 0.956773, 1.43783,$
$1.33789, 9.97534, 8.17526, 8.45979, 2.31093, 1.35675,$
$6.10588, 14.1647, 0.998377, 0.495293, 1.49227, 2.00033,$

2.64329, 2.02426) at which $f(x^*) = 130.490$, and $\beta^* = (0.071786, 0, 0, 0.291890, 1.47694, 0.526358, 0, 1.45979, 0, 0.146436, 0, 0, 0.007967, 0.008488, 0, 0.134951)$.

The algorithm was initialized as follows: $x = (2, 3, 5, 5, 1, 2, 7, 3, 6, 10, 2, 2, 6, 15, 1, 2, 1, 2, 1, 3)$, $\varepsilon_1 = 10^{-4}$, $\varepsilon_2 = \varepsilon_3 = 10^{-5}$, $\beta_j = 0$, $j = 1, 2, \ldots, 17$, $w_1 = w_{1_{max}} = 0$, $w_2 = w_3 = 0.25$, $w_{2_{max}} = w_{3_{max}} = 8$, wf = 4, nsrch = 41. Four place accuracy in $f(x^*)$ was obtained after 313 function calls and 100 gradient calls, and converged to the minimum in 490 function calls and 166 gradient calls. OPTKOV failed to find the minimum; this method stopped prematurely at the value $f(x) = 133.98$ after 583 function calls. POWCON obtained only three place accuracy at $f(x) = 130.81$ after stopping with 270 function calls. SUMT obtained four place accuracy in $f(x^*)$ after 581 function calls and converged to $f(x) = 130.489$ in 1120 function calls.

The following problem [64] is one that does not satisfy second-order sufficient conditions.

T12: (Rockafellar's problem)

Minimize

$$f = x_1^4 + x_1x_2 + x_2$$

subject to

$$x_2 = 0$$

The Lagrangian for this problem is

$$L = x_1^4 + x_1x_2 + x_2 - \alpha x_2 \qquad (7\text{-}1)$$

If we solve (7-1) by the classical approach, we have that $x^* = (0, 0)$, $f(x^*) = 0$, and $\alpha^* = 1$. The corresponding augmented Lagrangian, L_a is

$$L_a\Big|_{\alpha = \alpha^*} = x_1^4 + x_1x_2 + w_1x_2^2 \qquad (7\text{-}2)$$

The gradient of (7-2) is

$$\nabla L_a\Big|_{\alpha = \alpha^*} = \begin{bmatrix} 4x_1^3 + x_2 \\ x_1 + 2w_1x_2 \end{bmatrix} \qquad (7\text{-}3)$$

If we set (7-3) equal to zero and solve for x, we find that (7-2) has a saddle point at $x = (0, 0)$, and minima at $x = \pm(1/\sqrt{8w_1}, -1\sqrt{32w_1^3})$. As $w_1 \to \infty$ the minima points approach (0, 0). The Hessian of (7-2) is

$$\nabla^2 L_a \Big|_{\alpha = \alpha^*} = \begin{bmatrix} 12x_1^2 & 1 \\ 1 & 2w_1 \end{bmatrix} \qquad (7\text{-}4)$$

For (7-4) to be positive definite $x_1^2 > 1/24w_1$, but x_1 tends to zero only as w_1 tends to infinity, and therefore the second-order sufficient conditions are not satisfied. An attempt was made to solve this problem with ALGØR. Several solutions were obtained, each with a larger value of $w_{1_{max}}$. A maximum iteration limit of 400 was set for each solution, and was initialized each time under the same conditions; x = (0.5, 0.5), $\varepsilon_1 = 10^{-60}$, $\varepsilon_2 = \varepsilon_3 = 10^{-30}$, $w_1 = 1$, $w_2 = w_3 = w_{2_{max}} = w_{3_{max}} = 0$ wf = 2, nsrch = 5. The results are given in Table 7-1.

7.3. Summary

The multiplier program ALGØR was applied to eleven test problems, and the results were compared to Pierre's program MULTMETH [55], and the penalty-function methods OPTKOV, POWCON, and SUMT. The solution statistics of the eleven test problems run under ALGØR are given in Table 7-2. The tabulated run time in seconds was execution time for the problem solution on an XDS Sigma-7 time share

Table 7-1. Results of problem T:12.

w_{max}	x_1	x_2	α	$\|\nabla L_a\|$	f	function calls at search point 400
10	$1.5 \cdot 10^{-1}$	$-1.6 \cdot 10^{-2}$	1.1525	$3.2 \cdot 10^{-1}$	$-1.7 \cdot 10^{-2}$	1360
10^2	$-4.6 \cdot 10^{-2}$	$3.9 \cdot 10^{-4}$	0.9545	$7.8 \cdot 10^{-2}$	$3.7 \cdot 10^{-4}$	1650
10^3	$-1.3 \cdot 10^{-2}$	$1.2 \cdot 10^{-5}$	0.9874	$2.5 \cdot 10^{-2}$	$1.2 \cdot 10^{-5}$	1792
10^4	$-1.1 \cdot 10^{-4}$	$-9.2 \cdot 10^{-11}$	0.9999	$1.8 \cdot 10^{-6}$	$-9.2 \cdot 10^{-11}$	2254
10^5	$-1.2 \cdot 10^{-4}$	$-5.0 \cdot 10^{-12}$	0.9999	$1.0 \cdot 10^{-6}$	$-5.0 \cdot 10^{-12}$	2411
10^6	$-1.1 \cdot 10^{-4}$	$-4.1 \cdot 10^{-13}$	0.9999	$8.2 \cdot 10^{-7}$	$-4.1 \cdot 10^{-13}$	2423
10^{10}	$5.6 \cdot 10^{-10}$	$7.4 \cdot 10^{-27}$	1.0000	$2.2 \cdot 10^{-16}$	$7.4 \cdot 10^{-27}$	1953
10^{20}	$-5.1 \cdot 10^{-12}$	$-4.8 \cdot 10^{-38}$	1.0000	$5.3 \cdot 10^{-34}$	$-4.8 \cdot 10^{-38}$	2777
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
∞	0	0	1	0	0	∞

computer. All problems were stored on file prior to execution, and only initial and final search point status was printed for obtaining run time statistics. All execution was initiated from remote terminal, and the output received on the line printer at the computer center.

Table 7-2. Summary statistics for test problems.

test problem	function calls per search to 4 or 5 place accuracy	function calls per search to convergence	run time to convergence (sec)
T1	3.74	3.45	0.852
T2	2.21	2.24	0.960
T3	--	2.27	1.134
T4	2.52	2.36	3.420
T5	2.19	3.03	0.912
T6	2.47	2.57	1.656
T7	1.94	2.04	0.918
T8	2.08	2.20	1.470
T9	3.44	3.03	2.586
T10	2.80	2.74	5.598
T11	3.13	2.95	33.156
Average	2.652	2.888	

The results of test problems T2, T3, and T4 were compared to the results obtained by the multiplier method of Pierre [55]. ALGØR solved all three problems more efficiently than Pierre's method, which typically requires

5 function calls per search (iteration). The test problems were solved by ALGØR to the same degree of accuracy, in half (on the average) the number of function calls required by Pierre's method, and in approximately the same number of searches (1 gradient call per search required for both methods). The main difference of the two methods appears to be in the efficiency of the unidirectional search. ALGØR requires less than 3 (on the average) function calls per search (see Table 7-2).

In test problems T5 through T11 ALGØR was compared to OPTKOV, POWCON, and SUMT, which are penalty-function methods. A comparative study found in [5] provided the results for comparison for these problems. The number of function evaluations was chosen as the comparison criterion as the three penalty-functions were coded in ALGOL (ALGØR coded in FØRTRAN) and run on a different computer making the runtime to convergence an unfair comparison. The convergence test criteria for these methods was unknown, so the number of function calls required to reach a desired degree of accuracy in $|f(x) - f(x^*)|$ were compared.

The results of POWCON for test problems T9, T10, and T11 were approximately the same as those obtained by

ALGØR. In all other problem runs, ALGØR solved the problems, to a comparable degree of accuracy, much more efficiently than did the three penalty-function methods. The function calls required for these three methods ranged from 2 to 5 times more than the number required by ALGØR.

From the results presented here, the multiplier program ALGØR appears to be a better solution technique than the multiplier program of Pierre, and a much better technique than either POWCON and SUMT, which are two of the leading penalty-function methods in use today.

CHAPTER EIGHT

APPLICATION PROBLEMS

8.1. Introduction

In the first 7 chapters of this thesis the NLP has been given as an explicit set of equations from which the cost function was to be minimized subject to constraints on the x variables. The theoretical framework for problem formulation, solution identification, and an implementation of this theory into a working algorithm have been presented and tested. We now turn to real-world problems, their transformation into an explicit NLP, and their solutions.

The NLP arises in many forms from many unrelated areas. Zangwill [72] treats the NLP formulation from problems that arise in production and inventory control, regression analysis, consumer behavior, chemical condensor design, rocket control, equation solving, cost-benefit analysis, financial analysis, as well as application in the areas of economics, business, government, and engineering. Several texts and monographs have been published to bring the many aspects of problem formulation in a particular area into a stronger and more unified perspective than is offered in most journal articles. Dennis [18] investigates the application of mathematical programming in electrical networks. Trends in automated network design

optimization as applied to computer-aided design of lumped-distributed and microwave networks are reviewed and discussed in [6]. Duffin et al. [19] treat a wide class of engineering problems with both fixed and adjustable parameters from the aspect of geometrical programming which is a special form of NLP particularly useful in engineering and physical sciences. Optimal control is one of the most active research areas in modern technology, having applications in many areas, including aerospace, chemical, nuclear reactor, and transportation technologies. The basic concepts of mathematical programming and the techniques for implementing the solution for optimal control problems are presented in [67], along with many examples and case studies in the sub-areas of optimal control such as linear and nonlinear, continuous, and discrete time systems as well as stochastic and distributed parameter systems.

For many years the practice for on-line control of power systems has been to approximate the nonlinear relations of such systems to obtain linear programming problems. The equations are then solved by well-established linear programming techniques. Application of nonlinear

optimization for on-line control has the advantage of more accurate representation of the system, and the possibility of dealing with larger problems. Recent advances in this area are explored in [10]. Bracken and McCormick [9] present several nonlinear problem formulations including models on weapons assignment, bid evaluation, alkylation process optimization, chemical equilibrium, structural optimization, launch vehicle design and costing, parameter estimation in curve fitting, optimal sample sizes in stratified sampling on several variates, and deterministic nonlinear programming equivalents for stochastic linear programming problems.

Three selected real-world problem situations are now presented. From the problem statement, the NLP model will be formulated and then solved by the multiplier algorithm ALGØR.

8.2. A Circuits Problem

Consider the circuit shown in Figure 8-1 (Problem 6.31 of [53]). The ratio $|V_2(j\omega)/V_1(j\omega)|$ is of interest at two given frequencies $\omega = \omega_1 = 10^6$, and $\omega = \omega_2 = 2 \cdot 10^6$ $|V_2(j\omega_1)/V_1(j\omega_1)|$ is to be maximized subject to the constraint that $|V_2(j\omega_1)/V_1(j\omega_1)| = |V_2(j\omega_2)/V_1(j\omega_2)|$.

Components $R_1 = 10K\Omega$, and $R_2 = 1K\Omega$ are fixed; C_1, C_2, L_1, and L_2 are to be selected to give the constrained optimum. Additional constraints are $L_1 \geq 0$, $L_2 \geq 0$, $C_1 \geq 10$ picofarad, and $C_2 \geq 100$ picofarad.

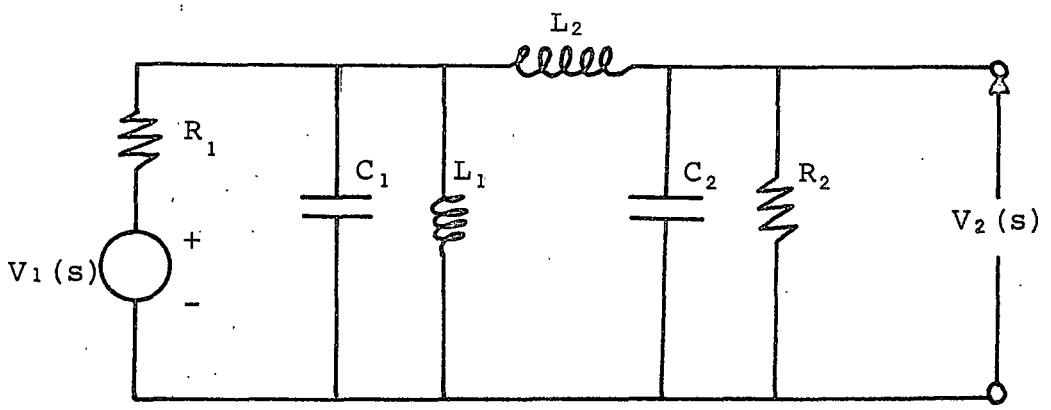


Figure 8-1. Circuit for NLP formulation.

This maximization problem may be put into the NLP format by the following

$$\text{minimize} \left|\frac{V_1}{V_2}\right|^2_{\omega = \omega_1} \tag{8-1}$$

subject to

$$\left|\frac{V_1}{V_2}\right|^2_{\omega = \omega_1} - \left|\frac{V_1}{V_2}\right|^2_{\omega = \omega_2} = 0$$

$$L_1 \geq 0 \qquad L_2 \geq 0$$

$$C_1 - 10^{-11} \geq 0 \qquad C_2 - 10^{-10} \geq 0 \tag{8-2}$$

Relations (8-1) and (8-2) cannot be solved directly in this form. The elements L_1, L_2, C_1, and C_2 must be related to V_1, V_2, and ω through appropriate circuit relations. Proper reduction of this circuit by the use of Laplace transform theory yields the following relation:

$$\frac{1}{H(s)} = \frac{V_1(s)}{V_2(s)} = \frac{(sC_2R_2+1)\,\dfrac{R_1/R_2}{(R_1/sL_1)+(sC_1R_1+1)} + (sC_2R_2+1)\,\dfrac{sL_2}{R_2} + 1}{\dfrac{sL_1}{R_1}\,\dfrac{1}{(R_1/R_2)(s^2L_1C_1)+(sL_1+R_1)+1}} \tag{8-3}$$

Define

$$a \triangleq R_1/R_2 = 10 \qquad b \triangleq \omega_2/\omega_1 = 2$$

$$x_1 \triangleq \omega_1 R_1 C_1 \qquad x_2 \triangleq \omega_1 R_2 C_2$$

$$x_3 \triangleq \omega_1 L_1/R_1 \qquad x_4 = \omega_1 L_2/R_2 \tag{8-4}$$

so that

$$bx_1 \triangleq \omega_2 R_1 C_1 \qquad bx_2 \triangleq \omega_2 R_2 C_2$$

$$bx_3 \triangleq \omega_2 L_1/R_1 \qquad bx_4 \triangleq \omega_2 L_2/R_2$$

Let $s = j\omega$. If we substitute the identities of (8-4) into (8-3) and collect terms we have

$$\frac{V_1(j\omega_1)}{V_2(j\omega_1)} = [a+1-x_1x_4-x_2x_4+\frac{x_4}{x_3}]$$

$$+ j[x_1+ax_2+x_4-x_1x_2x_4+\frac{x_2x_4}{x_3} - \frac{1}{x_3}] \qquad (8\text{-}6)$$

We see from (8-6) that the ratio $V_1(j\omega_1)/V_2(j\omega_1)$ is a complex number, having both real (Re) and imaginary (jIm) components. Similarly, a second ratio is obtained when (8-3) is evaluated at $\omega = \omega_2$ using the identities of (8-5):

$$\frac{V_1(j\omega_2)}{V_2(j\omega_2)} = Re(\omega_2) + jIm(\omega_2) \qquad (8\text{-}7)$$

If we use the fact that

$$\left|\frac{V_1}{V_2}\right| = (Re^2 + Im^2)^{\frac{1}{2}} \qquad (8\text{-}8)$$

then upon substituting (8-6) into (8-8) and squaring we have

$$\left|\frac{V_1}{V_2}\right|^2_{\omega = \omega_1} = r_1^2 + r_2^2 =$$

$$[11-x_1x_4-x_2x_4+\frac{x_4}{x_3}]^2 + [x_1-x_1x_2x_4+10x_2+\frac{x_2x_4-1}{x_3} + x_4]^2$$

(8-9)

Similarly, upon solving (8-7) and substituting into (8-8) and squaring we have

$$\left|\frac{V_1}{V_2}\right|^2_{\omega = \omega_2} = r_3^2 + r_4^2 =$$

$$[11-4x_1x_4-4x_2x_4+\frac{x_4}{x_3}]^2 + [2x_1-8x_1x_2x_4+20x_2+\frac{4x_2x_4-1}{2x_3} + 2x_4]^2 \qquad (8\text{-}10)$$

From the definition of x_1 in (8-4) and from the relations in (8-2) we know that

$$C_1 = \frac{x_1}{\omega_1 R_1} = \frac{x_1}{10^{10}} \geq 10^{-11} \qquad (8\text{-}11)$$

from which

$$x_1 - .1 \geq 0 \qquad (8\text{-}12)$$

Similarly we find that

$$x_2 - .1 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0 \qquad (8\text{-}13)$$

From the above development we can now state the NLP for this problem in the familiar form. We collect equations (8-9), (8-10), (8-12) and (8-13) and upon comparing them with (8-1) and (8-2) we have

A: 8.2 (Circuit Application)

Minimize

$$f = r_1^2 + r_2^2$$

subject to

$$(r_1^2 + r_2^2) - (r_3^2 + r_4^2) = 0$$

$$x_1 - 0.1 \geq 0$$

$$x_2 - 0.1 \geq 0$$

$$x_3 \geq 0$$

$$x_4 \geq 0$$

where

$$r_1 = [11 - x_1x_4 - x_2x_4 + \frac{x_4}{x_3}]$$

$$r_2 = [x_1 - x_1x_2x_4 + 10x_2 + \frac{x_2x_4-1}{x_3} + x_4]$$

$$r_3 = [11 - 4x_1x_4 - 4x_2x_4 + \frac{x_4}{x_3}]$$

$$r_4 = [2x_1 - 8x_1x_2x_4 + 20x_2 + \frac{4x_2x_4-1}{2x_3} + 2x_4]$$

Convergence to a minimum for this problem was very slow in that x_3 was ever so slowly continually increasing without bound, and convergence was obtained only when x_3 reached a large value. The problem was reformulated by replacing x_3 with $1/x_3$ wherever it occurred in the above

problem. The resulting formulation was the same as A:8.2 except that r_1, r_2, r_3, and r_4 were redefined as follows:

$$r_1 = [11 - x_1x_4 - x_2x_4 + x_3x_4]$$

$$r_2 = [x_1 - x_1x_2x_4 + 10x_2 + x_3(x_2x_4-1) + x_4]$$

$$r_3 = [11 - 4x_1x_4 - 4x_2x_4 + x_3x_4]$$

$$r_4 = [2x_1 - 8x_1x_2x_4 + 20x_2 + 0.5x_3(4x_2x_4-1) + 2x_4]$$

A constrained local minimum for this problem was then found at $x^* = (1.91663, 0.1, 0, 1.97181)$ at which $f(x^*) = 69.6755$, $\alpha^* = 0.197551$, and $\beta^* = (0, 10.7851, 11.4656, 0)$.

Interpreting the results of the NLP into terms of the given problem we find that the voltage ratio $|V_1(j\omega)/V_2(j\omega)|^2$ will have the same minimum value of 69.6755 at the two frequencies $\omega_1 = 10^6$ and $\omega_2 = 2 \cdot 10^6$. This minimum value will be for the selection of $C_1 = 192$ picofarads, $C_2 = 100$ picofarads, $L_1 = \infty$, and $L_2 = 1.97$ millihenries. The value $L_1 = \infty$ indicates that the inductor L_1 should not be included in the circuit.

The equality constraint multiplier α^* has no sensitivity meaning if the voltage ratios must be the same at the two given frequencies. However, we see from the small

value of α^* ($\alpha^* \approx 0.2$), that the circuit is relatively insensitive to small differences in the voltage ratios at the optimum. The multiplier associated with L_1 has no sensitivity meaning in that an incremental change of the value of ∞ has no significance. The multiplier associated with the inequality constraint on C_2 does have significance. The incremental sensitivity indication is that if the right side of $x_2 - .1 \geq 0$ is decreased incrementally, a reduction of 10.7851 will occur in the square of the voltage ratio for each unit of reduction in the right side of the constraints. Consider a 0.05 reduction in the right hand side of the constraint. The constraint is then $x_2 - .1 \geq -0.05$, or $x_2 \geq 0.05$. Solving this relation for C_2 as defined in (8-4) we have that $x_2 = \omega_1 R_2 C_2 \geq 0.05$, and $C_2 \geq 50$ picofarads. Thus, is we can reduce the lower bound on C_2 from 100 picofarads to 50 picofarads, we will reduce the square of the voltage ratio by $(10.7851)(0.05) = 0.5392$. This reduction compared to the value of 69.6755 indicates the circuit is insensitive to small variations in the C_2 capacitor around the design center value of 100 picofarads.

8.3. A Geometric Problem

The following problem is a geometric programming problem. It is an adaptation from [19] and is presented in [72].

As vice president of a small but dynamic chemical company, you are to investigate containerization for shipping. Not only should the containers simplify shipment, but also because the containers will be left with the customers, sales should be enhanced. Upon selecting a particular customer as a representative example of the problem, you find that each month 1000 cubic feet of a chemical must be shipped to this customer. The chemical is to be sent in rectangular containers of length x_1, width x_2, and height x_3. The thickness of all material used is negligible. The container seams are fused during the filling process to prevent moisture from entering. The sides and bottom must be made of scrap material for which there is no charge. However, only 10 square feet of this scrap can be guaranteed and processed per container each month from the materials department, but can be processed to any dimension. Material for the ends costs \$2.00 per square foot, while material for the top costs \$3.00 per square foot.

There is also a shipping charge of 20 cents per container sent. You must determine how many and what size of containers are needed to ship the chemical at the lowest possible cost. Thus, the total cost (shipping plus the costs of the two ends and the top for $1000/x_1x_2x_3$ containers) must be minimized subject to the scrap limitation.

The total cost per container is

$$\text{cost} = (0.2 + 4x_2x_3 + 3x_1x_2) \text{ dollars} \qquad (8\text{-}14)$$

subject to the scrap limitation

$$2x_1x_3 + x_1x_2 \leq 10 \qquad (8\text{-}15)$$

Additional constraints must be added to ensure that the dimensions remain nonnegative

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0 \qquad (8\text{-}16)$$

Considering the total cost for all $(1000/x_1x_2x_3)$ containers, equation (8-14) becomes

$$\text{cost}_t = \frac{200}{x_1x_2x_3} + \frac{4000}{x_1} + \frac{3000}{x_3} \text{ (dollars)} \qquad (8\text{-}17)$$

Let $f(x)$ be the total cost in thousands of dollars. Then from (8-15), (8-16), and (8-17) we can formulate the NLP as follows.

A: 8.3 (Container problem)

Minimize

$$f = \frac{0.2}{x_1 x_2 x_3} + \frac{4}{x_1} + \frac{3}{x_3} \text{ (thousands of dollars)}$$

subject to

$$1 - 0.2x_1x_3 - 0.1\, x_1x_2 \geq 0$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

$$x_3 \geq 0$$

The optimum for this problem is at $x^* = (2.37976, 0.316228, 1.94294)$ at which $f(x^*) = 3.36168$, and $\beta^* = (1.81762, 0, 0, 0)$.

Interpreting the optimum of the NLP in terms of the given problem we find that 684 containers of length 2.380 feet, width 0.316 feet, and height 1.943 feet are required to ship 1000 cubic feet of chemical each month at a cost of $3362. The scrap limitation constraint is active (all 10 square feet per container are used) at the solution and has an associated positive multiplier of 1.81762. There is no shadow cost for the free scrap material. Therefore, if by some means, additional scrap material could be obtained at no cost, the incremental sensitivity indication is that

an approximate savings of $182.00 from the total cost could be obtained for each additional square foot of scrap for each of the 684 containers. Reducing the above analysis to a single container basis, each container holds 1.46 cubic feet, and costs $4.92 for construction and shipping. For each additional square foot of scrap per container, approximately 27 cents could be saved.

8.4. Deterministic Nonlinear Programming for a Stochastic Linear Programming Problem

The problem presented here is equivalent to a stochastic linear programming problem of the chance-constrained type. In the chance-constrained problem, the constraint coefficients are normally distributed random variables. The determination of least-cost optimal cattle feed was modeled by van de Panne and Popp [68], and is treated in [9]. The presentation here of a least-cost optimal hog ration parallels that in [9].

The problem concerns the mixing of a number of raw materials in such a way that a hog ration is obtained that satisfies certain specified nutritive and other requirements with minimum cost for the input quantities of raw materials. If the nutritive contents and unit costs of

Table 8-1. Data for deterministic problem.

Fraction x(i)	Constituent	Cost/ton $100	Protein		Calcium		Phosphorus	
			mean	var.	mean	var.	mean	var.
x_1	barley	0.80	11.6	0.4844	0.05	0.0001	0.35	0.0010
x_2	wheat	1.10	13.7	0.3003	0.07	0	0.37	0.0009
x_3	corn	0.85	9.5	0.1444	0	0	0.10	0.0001
x_4	soy	3.45	48.5	0.588	0.33	0	0.62	0.0005
x_5	mustard	2.00	31.9	4.9863	0	0	0	0
x_6	dried milk	2.10	51.1	0.0653	1.27	0.0040	1.03	0.0021
x_7	fish soluables	3.00	65.5	21.0222	1.27	0.1404	1.69	0.0825
x_8	d_i - cal. phos.	0.80	0	0	23.35	1.3631	18.21	0.2073
x_9	limestone	0.45	0	0	35.84	0.5138	0.01	0
x_{10}	molasses	0.72	0	0	0.81	0.0289	0.08	0.0004
x_{11}	dehydrated alfalfa	1.80	21.8	0.2970	1.79	0.0097	0.31	0.0005
x_{12}	shrimp meal	3.00	46.9	9.2933	7.34	0.3893	1.59	0.0107
x_{13}	mono-sodium-phosphate	0.60	0	0	0	0	22.45	1.0206

$i = 1, 2, \ldots, 13$	a_i	b_i	$\sigma_{b_i}^2$	c_i	$\sigma_{c_i}^2$	d_i	$\sigma_{d_i}^2$
Requirement for:	starter	18		1.0		0.9	
	grower	16		0.8		0.7	
	finisher	14		0.7		0.6	

raw materials and the requirements for nutrients are known, the problem can be solved in a straightforward manner by linear programming methods. One problem that arises is that the nutritive content of raw materials varies randomly, so that the solution given by linear programming using expected (mean) values, for instance, does not always satisfy the requirements. That is, using the expected values from a normal distribution, there is only a 50% probability (confidence interval) that the nutritive requirements of the ration will be satisfied.

Table 8-1 gives the data of the problem. The percentage content of protein, calcium, and phosphorous are given for 13 constituents. The cost per ton (in hundreds of dollars) is given for each constituent. The requirments of protein, calcium, and phosphorus are given for starter pigs, growers, and finishers. The problem is to determine the mix with minimum cost per ton that satisfies the given requirements. Since the method of solution is the same for all three rations, only the ration for starter pigs will be given.

Let a_i, $i = 1, 2, \ldots, 13$, denote the cost per ton of each constituent. Let b_i, c_i, and d_i, $i = 1, 2, \ldots, 13$, denote the percentage content for protein, calcium and

and phosphorous, respectively. Let x_i, $i = 1, 2, \ldots, 13$, denote the fraction of the mixture that is composed of each of the constituents. With these definitions the deterministic linear programming model is as follows.

A: 8.4L (linear hog ration)

Minimize

$$f = \sum_{i=1}^{13} a_i x_i$$

subject to

$$\sum_{i=1}^{13} b_i x_i - 18 \geq 0$$

$$\sum_{i=1}^{13} c_i x_i - 1 \geq 0$$

$$\sum_{i=1}^{13} d_i x_i - 0.9 \geq 0$$

$$x_i \geq 0 \qquad i = 1, 2, \ldots, 13$$

$$\sum_{i=1}^{13} x_i - 1 = 0$$

The optimal solution is at $x^* = (0.785587, 0, 0, 0, 0, 0.173919, 0, 0, 0.020643, 0, 0, 0, 0.019853)$ at which $f(x^*) = 1.01490$, $\alpha^* = 0.417284$, and $\beta^* = (0.032743, 0.000911, 0.0081388, 0, 0.231060, 0.120842, 1.143933, 0.538211, 0, 0.423103, 0.213246, 0, 0.301328, 0.664765, 1.02744, 0)$.

Interpreting the optimum of the NLP we find that a mixture of

$x_1 = 0.7856$ (barley)

$x_6 = 0.1739$ (dried milk)

$x_9 = 0.0206$ (limestone)

$x_{13} = 0.0199$ (mono-sodium phosphate)

will satisfy the minimum requirements of the ration for a cost of \$101.49 per ton. But this ration will satisfy the requirement only 50% of the time, since the protein, calcium, and phosphorous percentage contents were calculated from the mean values. In actuality the percentage contents can vary from batch to batch since they are normally distributed independent random variables. The nonlinear equivalent of this stochastic linear problem takes into account the distribution of these random variables by constraints of the form

$$p - \sum_{i=1}^{n} \mu_i x_i + \Psi(\alpha)\left[\sum_{i=1}^{n} \sigma^2(\mu_i) x_i^2\right]^{\frac{1}{2}} \geq 0 \qquad (8\text{-}18)$$

where $\Psi(\alpha)$ is the percentage point corresponding to $(1-\alpha)$. If $\alpha = .95$ (95% confidence interval), $\Psi(\alpha)$ is the .05 fractile. A value of $\alpha = .95$ corresponds to a value of $\Psi(\alpha) = -1.645$.

If we use the means and variances listed in Table 8.1 in inequality (8-18), and a 95% confidence interval, we obtain the following nonlinear model.

A: 8.4N (nonlinear hog ration)

Minimize

$$f = \sum_{i=1}^{13} a_i x_i$$

subject to

$$\sum_{i=1}^{13} b_i x_i + (-1.645)\left(\sum_{i=1}^{13} \sigma_{b_i}^2 x_i^2\right)^{\frac{1}{2}} - 18 \geq 0$$

$$\sum_{i=1}^{13} c_i x_i + (-1.645)\left(\sum_{i=1}^{13} \sigma_{c_i}^2 x_i^2\right)^{\frac{1}{2}} - 1 = 0$$

$$\sum_{i=1}^{13} d_i x_i + (-1.645)\left(\sum_{i=1}^{13} \sigma_{d_i}^2 x_i^2\right)^{\frac{1}{2}} - 0.9 \geq 0$$

$$x_i \geq 0 \qquad i = 1, 2, \ldots, 13$$

$$\sum_{i=1}^{13} x_i - 1 = 0$$

Initial solution of this problem revealed that the percent content of calcium of the ration exceeded a known upper bound set for proper growth regulation of starter pigs. Therefore, rather than add an additional constraint for the upper bound, the requirement for calcium was converted to an equality constraint as evidenced by the above constraint set.

A constrained local minimum for this problem was located at x^* = (0.760844, 0, 0, 0, 0, 0.196658, 0, 0, 0.020830, 0, 0, 0, 0.021669) at which $f(x^*)$ = 1.04403, α^* = (-0.000405, 0.464116), and β^* = (0.031922, 0.006357, 0, 0.196227, 0.819881, 1.43385, 0.517568, 0, 0.434755, 0.229566, 0, 0.255704, 0.638736, 1.03160, 0).

Interpreting the minimum of the NLP we find that a ration mixture of

x_1 = 0.7608 (barley)

x_6 = 0.1967 (dried milk)

x_9 = 0.0208 (limestone)

x_{13} = 0.0217 (mono-sodium phosphate)

will satisfy the minimum requirements for 95% of all batches mixed. The cost per ton is $104.40, which is an increase of approximately $3.00 over the linear program ration. But the $3.00 additional cost has bought a 95% probability that the ration requirements will be satisfied as compared to a 50% probability.

The mixture constituents are the same. The main mixture difference for the nonlinear problem is an increased fraction of dried milk and a decreased fraction of barley. Proper interpretation of the Lagrange multipliers will give an incremental indication of additional costs or savings that would result from the requirement changes for the ration.

The first component of α^* indicates incremental change in $f(x^*)$ for changes in the calcium requirement. Its small negative value has little effect on f for small changes, but indicates a savings if the calcium requirement were increased. The second component has no sensitivity

significance, in that the associated equality constraint is a mathematical law, stating that the percentage fractions of the constituents for the total ration must be 100% and cannot be changed.

The associated multipliers with the $x_i \geq 0$ constraints indicate a savings for a small decrease from 0 in these constraints. But due to the fact that the constituents of the ration cannot have a negative fraction, this sensitivity information is of no value. The first two components of β^* associated with the requirement constraints on protein and phosphorous indicate a small savings could be obtained if these ration requirements could be reduced, i.e., lower the minimum ration requirements for protein and phosphorous. The first component is 5 times larger than the second indicating that the price per ton of the total ration is more sensitive to the requirement for protein than that for phosphorous. But the magnitudes of both multipliers show relative insensitivity to both requirements for small changes.

8.5. Summary

Three interesting real-world application problems are presented in this chapter. The first is a circuit

design problem in which design center values of certain components are to be selected to maximize the output voltage to the same value for two given frequencies. The second is a geometric containerization problem, in which similar rectangular containers are to be designed to ship a given amount of chemical, for minimum total cost. The total cost is a function of a fixed charge for shipping of each container, and the price of materials available for the construction of the containers.

The third problem is a direct application to the livestock industry. A minimum-cost optimal feed ration for hogs is obtained both from the linear programming model, and the nonlinear model. The merits of the results from both models are evaluated.

The three problems are presented to show how the NLP may be formulated from real-world problem situations, and how the results may be interpreted and implemented. The importance of the Lagrange multipliers are seen through the interpretation of the sensitivity information they contain, and how this indicative information may be used in making investment and policy changes without solving numerous NLP problems.

These examples give a minute representation of the type of problems that can occur in almost any area. Problems that arise from other situations could be much larger in scale and complexity. But with proper transformation into the NLP format, they may be solved using nonlinear techniques.

CHAPTER NINE

EVALUATION AND CONCLUSIONS

9.1. Summary

In Chapter 1 the NLP is introduced in its standard form for a constrained minimization problem. The general method of sequential unconstrained minimization is then given, showing how the constrained NLP problem functions may be transformed into a single unconstrained function having the same solutions as the original problem. In Chapter 2 the theoretical framework for the identification and characterization of local unconstrained minima are presented. An unconstrained function, formulated by augmenting the Lagrangian with weighted problem functions of the constrained NLP is presented, and first-order necessary and second-order sufficient conditions are established for this function to possess local minima that minimize f and satisfy the constraints of the NLP. A multiplier algorithm, based on satisfying the Kuhn-Tucker first-order conditions of optimality, is posed. The algorithm parallels that of Pierre [55] and consists of alternating minimization and update phases. It has an added flexibility, however, in that it has three modes of operation. It can be used as: 1) an exterior penalty-function method, where the multipliers are held equal to zero, and the penalty

weights are updated after each cycle, 2) a multiplier method where both multipliers and penalty weights are updated after each cycle, or 3) a combination of both methods with an initial penalty phase and a terminal multiplier phase. An initial premise of this thesis was that an algorithm operating in mode 3 would be most efficient, i.e., using a multiplier method for termination would alleviate the ill-conditioning effects often encountered in straight penalty-function methods. It was supposed that if the algorithm were allowed to operate in the penalty-function phase for several cycles, the multiplier phase could then be initialized with good approximations to x^*, α^* and β^*, and faster convergence would result in the final phases. Computational experience, however, has been that initial penalty phases before the introduction of the multiplier phase are less efficient than utilizing the straight multiplier method, where both multipliers and penalty weights are updated after every cycle.

The general procedure for updating the multipliers and weights during the minimization process is also presented in Chapter 2. This procedure establishes a way to select an active inequality constraint basis, such that

during a minimization phase these inequality constraints exhibit a 'locked on' feature and are treated the same as equality constraints. Although the algorithm is based on first-order conditions, if second-order conditions are satisfied by the NLP problem functions, the algorithm converges locally without requiring the penalty weights to increase without bound. As the sequence of minimizers $\{x^m\}$ converges to a local minimum point x^*, the multipliers also converge to their optimal values α^* and β^*. These optimal multiplier values have a practical interpretation when the NLP is formulated from a real-world problem as shown in the sensitivity analysis given in Chapter 3.

Methods of generating search directions based on the gradient information of the problem functions are discussed in Chapter 4. It was found that the extension of the basic gradient methods to conjugate gradient methods alleviated some of the difficulties presented by steep ridges and ravines that are often encountered in steepest descent search techniques, when the functions are nonlinear. It was shown that if the inverse matrix of second partials A^{-1} was available, the minimum of a quadratic function could be attained in one iteration if the Newton

method was used, and in one cycle if a conjugate gradient method was used. The general NLP is not quadratic, but may be adequately represented by a quadratic approximation when in the vicinity of a local minimum. As the minimum of a problem is approached, such that the local quadratic approximation becomes valid, the DFP method generates a sequence $\{H^i\}$ that tends to A^{-1}, the inverse matrix of second partial derivatives, without requiring the use of second derivatives of the problem functions. Even if the function in question is not quadratic, the sequence $\{H^i\}$ remains positive definite, and the unidirectional search is conducted over $\alpha > 0$ only. The above features influenced the selection of the DFP method to be used for generating conjugate search directions for the multiplier algorithm. Computational experience with both the DFP method and the method of Fletcher and Reeves in the early phases of this research showed the DFP method to be the better method. This is also verified computationally by Pierre [55].

An efficient quadratic convergent unidirectional search technique developed by this author is posed in Chapter 5. High accuracy is not required by a

unidirectional search in regions remote from a local minimum, and since in the vicinity of such a minimum, f(x) can be adequately represented by the local quadratic approximation, a quadratic convergent search technique is highly efficient. The detailed analysis of the search procedure is presented and compared to a popular cubic convergent search technique.

The material of Chapters 2, 3, 4, and 5 is then integrated into a working multiplier algorithm ALGØR. The algorithm is coded in FØRTRAN. The implementation of this algorithm is discussed through the flow connections given in Chapter 6.

The algorithm was applied to several test functions and the results were compared to the results obtained by other well-known NLP solution algorithms. When based on the number of function calls required to obtain a desired degree of accuracy in the cost function, ALGØR was shown to be a more efficient code than the multiplier program of Pierre, and the penalty-function programs OPTKOV, POWCON, and SUMT.

The formulation of three selected real-world problems into the NLP format is presented in Chapter 8, along

with their solutions and interpretations of the sensitivity information possessed by the Lagrange multipliers at the solution points.

9.2. Evaluation of the Multiplier Algorithm

The multiplier algorithm ALGØR, developed in this thesis to solve the general NLP has the following desirable properties:

1. The algorithm is based on the Kuhn-Tucker first-order conditions for optimality.

2. The algorithm converges to constrained local optimum points and tends to avoid other points which may satisfy first-order necessary conditions.

3. The algorithm is quadratic convergent.

4. The algorithm converges in a finite number of steps for a class of linear programming problems.

5. The terms used to augment the Lagrangian are formed from the NLP problem functions in a simple way that requires a minimum amount of computation.

6. Highly nonlinear functions do not present unusual problems, since the minimization procedure is not governed by linear models that apply over small neighborhoods of a given point.

7. The search directions are generated by a conjugate gradient technique. Thus, steep ridges and ravines often caused by nonlinear problem functions do not unduly hinder the algorithm.

8. The search directions are calculated as a part of the iteration process based on accumulated information from past search points rather than following pre-calculated search directions.

9. Only the problem functions and their first derivatives must be specified.

10. The constraints need not be satisfied at initial starting points for x.

11. The algorithm has no problem in dealing with inequality constraints. The simple multiplier updating rules can effectively determine the active inequality constraint basis incorporating the locked-on feature.

12. The algorithm generates multipliers which provide well-known sensitivity information.

13. The selection of search parameters is generally not critical. However, the penalty weights should be selected large enough to avoid constraint breakthroughs and ensure an adequate rate of convergence, but not so large that the solution path is unduly confined to constraint boundaries during the initial interations of the algorithm. This may lead to inefficient optimization and a slow convergence rate.

14. The algorithm requires only 1 gradient evaluation and less than 3 (on the average) function evaluations per iteration. This is important when the evaluation of the model for the NLP requires a relatively large amount of computation time as to that required for the search iteration. The more efficient minimization methods tend to keep the model evaluations to a minimum.

The above properties of the multiplier algorithm developed and presented in this work may hopefully give

impetus to multiplier algorithms in general such that they can displace the currently more popular penalty methods.

9.3. Suggestions for Continued Research

When compared to the penalty-function methods discussed in Chapter 7 on the basis of the number of function evaluations, the multiplier algorithm appears to be a better NLP solution method. A more comprehensive comparison to a wider class of methods, and wider class of problems should be conducted to justify the validity of the above statement. A comparison to other methods should be made on both the number of function evaluations and on execution times for the particular problems. This would require that all methods be coded in the same computer language and run on the same or an identical computer. Such a comparative study has been conducted by Colville [12]. The participants of this study coded their programs in FORTRAN and were required to run a standard timing package on their particular computer, to estimate the computing power of the particular machine used. In this way execution time could be normalized and compared on a fair basis. Consideration should be given to comparing the multiplier technique to the many techniques evaluated in this study.

It was shown that the multiplier algorithm converges in a finite number of steps for a class of linear programming problems. This algorithm should be applied to a wide class of linear problems and the results compared to the results of well known linear programming (LP) techniques. ALGØR may prove to be as efficient as, or perhaps superior to LP techniques as the size of the linear problem increases.

Consideration should also be given to converting the minimization algorithm to one that treats maximization problems in which functions are constrained to be less than or equal to specified values. This could be done without significantly altering the theoretical structure of the algorithm.

The augmented Lagrangian of this work should be studied in the light of duality theory, where the multipliers are the object of the search. During the optimization sequence, as the multipliers converge to their optimal values, the x values are updated such that at convergence the optimal x values are obtained.

The general routine implemented for this research should be maximized for efficiency. This could result in

a reorganization of the program and a reduction in code. Also the convergence rate of the algorithm should be investigated in a more rigorous manner as done in [42].

Several alternative methods for augmenting the Lagrangian with penalty terms and updating multipliers have been suggested in the literature. These should be studied in the light of the algorithm presented in this work. None of the other penalty terms or updating rules are as easily computed as those used here, but some may be developed through an analysis of convergence rates to lead to a significant improvement.

APPENDIX A

LAGRANGIAN FORMS

For the general NLP in Section 1.2, make the following definitions:

$\phi \triangleq \phi(h(x), \alpha, w)$ is a multiplier function associated with the equality constraint functions $h_i(x) = 0$, $i = 1, 2, \ldots, m_1$

$\lambda \triangleq \lambda(g(x), \beta, w)$ is a multiplier function associated with the inequality constraint functions $g_j(x) \geq 0$, $j = 1, 2, \ldots, m_2$

α is an equality constraint multiplier vector of dimension m_1

β is an inequality constraint multiplier vector of dimension m_2

w is a weighting parameter, $w > 0$

θ, t are parameters

Define also the generalized modified Lagrangian, L_m, as follows:

$$L_m(x, \alpha, \beta, w) = f(x) + \sum_{i=1}^{m_1} \phi(h_i(x), \alpha_i, w)$$

$$+ \sum_{j=1}^{m_2} \lambda(g_j(x), \beta_j, w)$$

Examples of multiplier functions for the ϕ class are:

A1: $\alpha h + wh^2$ (Hestenes) (Arrow, Solow)

A2: $-hp + \nabla f + \frac{1}{2}\theta h^2$ (Fletcher)

where θ is a curvature parameter (matrix) > 0 (positive definite), and p is the generalized inverse of $n = \nabla h$, $p = (n'n)^{-1}n'$

A3: $w(h + \theta)^2$ (Powell)

A4: $\frac{1}{tw}((wh + \alpha)^t - \alpha^t)$, (Mangasarian)

where t is an even integer $t \geq 4$

A5: $\cosh(wh + \alpha) - \frac{1}{2}(wh + \alpha)^2 - \cosh\alpha + \frac{1}{2}\alpha^2$

(Mangasarian)

Examples of multiplier functions for the λ class are:

A7: $\frac{\beta(g+1)^{1+t}}{(1+t)}$, (Arrow, Hurwicz)

where t is an even integer

A8: $(\beta/w)\exp(wg)$ (Gould, Howe)

A9: $-\frac{\beta}{4w}$, $g \leq -\frac{1}{2w}$ (Arrow, Gould, Howe)

$\beta wg^2 + \beta g$, $g > -\frac{1}{2w}$

A10: $-\frac{\beta^2}{4w}$, $g \leq -\frac{\beta}{2w}$ (Rockafellar)

$wg^2 + \beta g$, $g \geq -\frac{\beta}{2w}$

$$\beta g + \tfrac{1}{2}g^2 + g^t, \quad g \geq 0$$

A11: $\beta g + \tfrac{1}{2}g^2, \quad -\beta \leq g \leq 0$

$$-\tfrac{1}{2}\beta^2, \quad g \leq -\beta,$$

(Kort, Bertsekas)

where $t > 2$ selected arbitrarily

: in the following $z_+ = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$

A12: $\frac{1}{tw}\left((wg + \beta)_+^t - \beta^t\right)$, (Mangasarian)

where t is an even integer, $t \geq 4$

A13: $\cosh(wg + \beta)_+ - \tfrac{1}{2}(wg + \beta)_+^2 - \cosh\beta + \tfrac{1}{2}\beta^2$

(Mangasarian)

A14: $\tfrac{1}{2}(\cosh(wg + \beta)_+ - 1)^2 - \tfrac{1}{2}(\cosh\beta - 1)^2$

(Mangasarian)

Each of the functions above give the general form of their class of functions from which other multiplier functions may be generated.

APPENDIX B

THEOREM AND LEMMA PROOFS

Proof of Theorem 1.

Let $f(x^*)$ be an unconstrained local minimum. Thus, it may be assumed that $f(x^*) \leq f(x^* + \Delta x)$ for all Δx with $0 < \|\Delta x\| < \varepsilon$. Since f is of class C^1, we can further assume that the limit

$$\lim_{\substack{\Delta x_i \to 0 \\ i=1,\ 2,\ \ldots,\ n}} \frac{f(x^* + \Delta x_i) - f(x^*)}{\Delta x_i} = \nabla f(x^*)$$

exists, where $\Delta x_i = x_i - x_i^*$. Since $f(x^* + \Delta x) - f(x^*)$ is never negative, letting each element of the Δx vector approach zero from above and below, we find that $\nabla f(x^*)$ obeys $\nabla f(x^*) \geq 0$ and $\nabla f(x^*) \leq 0$ so that necessarily

$$\nabla f(x^*) = 0$$

Q

Proof of Lemma 1. [69, 73]

Let $A \triangleq \{z \mid z'v_k \geq 0, \quad k = 1, 2, \ldots, q\}$

If $v_0 = \sum_{k=1}^{q} s_k v_k, \quad s_k \geq 0, \quad k = 1, 2, \ldots, q,$

then for all $z \in A$, $z'v_0 \geq 0$.

Proof: $z \in A$ implies $z'v_k \geq 0$, $k = 1, 2, \ldots, q$. Since $s_k \geq 0$, $k = 1, 2, \ldots, q$, it follows that $s_k z'v_k \geq 0$

$k = 1, 2, \ldots, q$ from which

$$\sum_{k=1}^{q} s_k z' v_k = z' \sum_{k=1}^{q} s_k v_k \geq 0$$

and hence

$$z' v_0 \geq 0$$

Converse: If for all $z \in A$, $z' v_0 \geq 0$, then

$$v_0 = \sum_{k=1}^{q} s_k v_k$$

where $s_k \geq 0$, $k = 1, 2, \ldots, q$.

Proof: Let $$C \triangleq \{y \in E^n \,\Big|\, y = \sum_{k=1}^{q} s_k v_k\}$$

The set C is a convex polyhedral cone in E^n, i.e.,

i) for any $y \in C$ and scalar $\lambda \geq 0$, $\lambda y \in C$

ii) for $y_1, y_2 \in C$, $(y_1 + y_2) \in C$

Now let $y \in C$ and $z' v_0 \geq 0$ for all $z \in A$. Suppose $v_0 \notin C$. Let q be the projection of v_0 onto C such that $q - r = v_0$ and $q + t = y$. Then

$$r' q = 0$$

and

$$r' t \geq 0$$

which imply

$$r'q + r't = r'y \geq 0, \quad \text{for all } y \in C$$

hence,

$$r'v_k \geq 0, \quad k = 1, 2, \ldots, q$$

which implies $r \in A$. Now

$$r'v_0 = r'q - r'r = -r'r < 0$$

which contradicts $z'v_0 \geq 0$ for all $z \in A$. Hence, $v_0 \in C$ and thus

$$v_0 = \sum_{k=1}^{q} s_k v_k$$

$$0$$

Proof of Theorem 2.

For all i write $\{h_i(x^*) = 0\}$ as two inequalities $\{h_i(x^*) \geq 0\}$ and $\{h_i(x^*) \leq 0\}$. Assumption iii that $S_2^* = \phi$ implies that $\Delta x = x - x^*$ belongs to S_1^*. Then by Lemma 1 there exist nonnegative scalar values u_i^*, $i = 1, 2, \ldots, m_1$, v_i^*, $i = 1, 2, \ldots, m_1$, and β_j^*, $j \in S_a$ such that

$$\nabla f^* = \sum_{i=1}^{m_1} (u_i^* - v_i^*) \nabla h_i^* + \sum_{j \in S_a} \beta_j^* \nabla g_j^*$$

Let $\alpha_i^* = (u_i^* - v^*)$, $i = 1, 2, \ldots, m_1$, and $\beta_j^* = 0$ for all $j \notin S_a$, then

$$\nabla f^* - \sum_{i=1}^{m_1} \alpha_i^* \nabla h_i^* - \sum_{j=1}^{m_2} \beta_j^* \nabla g_j^* = 0$$

which is just condition (2-19e), $\nabla L^* = 0$, where $\beta_j^* \geq 0$ for $g_j^* = 0$, and $\beta_j^* = 0$ for $g_j^* > 0$.

o

Proof of Theorem 3.

Let the vectors $\{\nabla h_i^*\}$, $i = 1, 2, \ldots, m_1$ and $\{\nabla g_j^*\}$, $j \in S_a$ be a linearly independent set of vectors v_1, v_2, ..., v_q where $q = m_1$ + total number of indices j, for $j \in S_a$. And let the vectors v_1, v_2, ..., v_q, v_{q+1} be a set of linearly dependent vectors where $v_{q+1} = \nabla f^*$. It is assumed that $\{\nabla h_i^*\} \neq 0$ for all i and $\{\nabla g_j^*\} \neq 0$ for $j \in S_a$. Then for constants $c_1, c_2, \ldots, c_{q+1}$

$$c_1 v_1 + c_2 v_2 + \ldots + c_q v_q + c_{q+1} v_{q+1} = 0$$

where not all c_i are zero. Clearly $c_{q+1} \neq 0$. (Otherwise $v_1, v_2, \ldots, v_q$ become linearly dependent which contradicts that they are linearly independent.) Thus, v_{q+1} can be expressed as a linear combination of ϕ $v_1, v_2, \ldots, v_q$.

Hence

$$v_{q+1} = \sum_{i=1}^{q} t_k v_k$$

where $\{t_k\}$, $k = 1, 2, \ldots, q$ are constants. Given a vector z such that $z'v_k \geq 0$, $k = 1, 2, \ldots, q$, then it follows from Lemma 1 that

$$z'v_{q+1} \geq 0$$

which implies $S_2^* = \phi$

o

Proof of Theorem 5.

Consider the Taylor series expansion of f(x) about the point x^*

$$f(x^* + \Delta x) = f(x^*) + \Delta x' \nabla f(x^*) + \tfrac{1}{2}\Delta x' \nabla^2 f^* \Delta x + \text{higher order terms}$$

Suppose at x^* that $\nabla f(x^*) = 0$, then for sufficiently small Δx, $||\Delta x|| \neq 0$, the higher order terms become negligible and it follows that

$$f(x^* + \Delta x) - f(x^*) \tilde{=} \tfrac{1}{2}\Delta x' \nabla^2 f^* \Delta x$$

Suppose $\nabla^2 f^*$ is positive definite. Then $\Delta x' \nabla^2 f^* \Delta x$ is always positive, independent of Δx. Hence,

$$f(x^* + \Delta x) > f(x^*)$$

and x^* is an unconstrained local minimum of f.

o

Proof of Theorem 8.

Given the equation

$$f(x) = f(x^a) + (x-x^a)'g(x^a) + \tfrac{1}{2}(x-x^a)'A(x-x^a)$$

then

$$\left.\frac{\partial f(\alpha r^1)}{\partial \alpha}\right|_{\alpha = 0} = 0$$

implies that

$$(r^1)'[g(x^a) - Ax^a] = 0$$

Similarly

$$\left.\frac{\partial f(x^a - x^b + \alpha r^1)}{\partial \alpha}\right|_{\alpha = 0} = 0$$

implies that

$$(r^1)'[g(x^a) - Ax^b] = 0$$

Subtracting the second implication from the first yields

$$(r^1)'A(x^a - x^b) = (r^1)'Ar^2 = 0$$

O

APPENDIX C

PROGRAMMING INSTRUCTIONS FOR ALGØR

C.1. General

Subroutine ALGØR solves the general nonlinear problem (NLP) when stated in the following form:

$$\text{minimize } F(x)$$

subject to the constraints

$$FE_i(x) = 0 \qquad i = 1, 2, \ldots, NE$$

and

$$FI_j(x) \geq 0 \qquad j = 1, 2, \ldots, NI$$

where x is an n-dimensional vector, F(x) is the objective or cost function, $FE_i(x)$'s are the equality-constraint functions, and $FI_j(x)$'s are the inequality-constraint functions. The FE_i's, FI_j's, and F are given functions of class C^1, and may be linear or nonlinear. NE and/or NI may be equal to zero. If both NE and NI are zero, the problem is unconstrained.

ALGØR generates conjugate gradient search directions by the Davidon-Fletcher-Powell (DFP) method for each unidirectional search (UDS). The UDS finds the 'best' step-size in a particular search direction by means of a quadratic fit of data generated along that direction.

ALGØR can be used as 1) an exterior penalty function method, 2) a multiplier method, or 3) a combination of both methods with an initial penalty phase and a terminal multiplier phase. The type of method to be employed is controlled through data cards.

C.2. User Supplied Routines and Data

The following four routines must be supplied to ALGØR by the user. A brief description and form listing of each subroutine will be given.

Main Calling Program:

The user supplied main program properly dimensions all vectors and makes the appropriate call to ALGØR. The calling program is:

```
      DIMENSIØN X(N),XT(N),XS(N),GF(N),GØ(N),GAL(N),
     &DLX(N),DLG(N),R(N)
      DIMENSIØN FE(NE),FES(NE),HE(NE)
      DIMENSIØN FI(NI),FIS(NI),HI(NI)
      DIMENSIØN H(N*N),FM1(N*N),FM2(N*N)
      DIMENSIØN GE(N*NE)
      DIMENSIØN GI(N*NI)
      CALL ALGØR(X,XT,XS,FE,FES,HE,FI,FIS,HI,GF,GE,GI,
     &GØ,GAL,DLX,DLG,H,FM1,FM2,R)
      STØP
      END
```

Note: The arguments in the dimension statements are symbolic; specific integer values greater than or equal to

these values must be used in the dimension statements when running a program. N is the number of x_i variables, NE is the number of equality constraints, and NI is the number of inequality constraints. If NE = 0, the vectors FE, FES, HE, and GE need not be dimensioned. If NI = 0, the vectors FI, FIS, HI, and GI need not be dimensioned. Since the argument list of ALGØR is long, the user is cautioned to check that precisely 20 arguments are present.

In the remaining user supplied routines, all vectors are given the dimension 1, as allowed by FORTRAN.

Subroutine for Supplying Problem Functions:

The user supplied subroutine FXNS lists the cost function F to be minimized, and all equality-constraint functions FE, and all inequality-constraint functions FI. The subroutine FXNS is of the form:

```
SUBRØUTINE FXNS(X,F,FE,FI)
DIMENSIØN X(1),FE(1),FI(1)
F =              .............   cost function
FE(1)   =
  .                              NE equality
  .     =        .............   constraints; if
  .                              NE = 0, no listing
FE(NE)  =
FI(1)   =
  .                              NI inequality
  .     =        .............   constraints; if
  .                              NI = 0, no listing
FI(NI)  =
RETURN
END
```

Subroutine for Supplying Nonconstant Gradient Components of Problem Functions:

This user supplied gradient subroutine gives the nonconstant gradient components of all equality, GE, and inequality, GI, constraint functions. (Nonconstant gradient components are those components which are functions of x.) This routine is called after every unidirectional search when a new x is found. If there are no nonconstant gradient components, the SUBRØUTINE, DIMENSIØN, RETURN, and END statements must still be supplied by the user. This routine is of the form:

```
SUBROUTINE GRAD(X,GF,GE,GI)
DIMENSION X(1),GF(1),GE(1),GI(1)
GF(1)     =
  .                          N (gradient) components of F. If a
  .       = .........        given (total) GF component is a
  .                          constant, it need not be listed here.
GF(N)     =                  If there are no nonconstant GF com-
                             ponents, no listing is needed.
GE(1)     =
  .                          N (gradient) components of FE(1),
  .       = .........        then N components of FE(2), ..., then
  .                          N components of FE(NE). If NE=0, no
GE(N*NE)  =                  listing is needed. If a given (total)
GI(1)     =                  GE component is a constant, it need
  .                          not be listed here.
  .       = .........        N (gradient) components of FI(1),then
  .                          N components of FI(2), ..., then N
GI(N*NI)  =                  components of FI(NI). If a given GI
RETURN                       component is a constant, it need not
END                          be listed here.
```

Subroutine for Supplying Constant Gradient Components of Problem Functions:

This user supplied gradient subroutine gives the constant gradient components of the cost function GF, and the constant gradient components of all equality, GE, and inequality, GI, constraint functions. All gradient components equal to zero need not be listed. This routine is called only once by ALGØR, and these components remain fixed throughout the algorithm operation. If there are no constant gradient components, the SUBRØUTINE, DIMENSIØN, RETURN, and END statements must still be supplied by the user. This routine is of the form:

```
SUBROUTINE CONST(GF,GE,GI)
DIMENSION GF(1),GE(1),GI(1)
GF(1)     =
   .
   .      =
   .
GF(N)     =
GE(1)     =
   .
   .      =  ..........
   .
GE(N*NE)  =
GI(1)     =
   .
   .      =
   .
GI(N*NI)  =
RETURN
END
```

Same as SUBRØUTINE GRAD but applies to (total) constant gradient components that are nonzero.

Information and Format of Data Cards:

The data cards communicate to Subroutine ALGØR the initial conditions, convergence and update criteria, print interval, and the type of method to be used in solving the problem: i.e., 1) Penalty method; 2) Multiplier method; and 3) Penalty/Multiplier method.

The correct order and format of the data cards, with parameter definitions and typical values where applicable, are as follows:

DATA CARD 1:

```
      READ 10, (ID(I), I = 1,18)
   10 FØRMAT (18A4)
```

This is an identification card to be printed at the beginning of the output. It is a problem title card containing such information as: problem number, book, author, or any other description such as linear, nonlinear, 6th-order, constrained, unconstrained, etc.

DATA CARD 2:

```
      READ 100,N,NE,NI,IMAX,NPRINT,IHE,IHI,ICØN,NCYCL,
     &NSRCH,NUP
  100 FØRMAT (11I5)
```

N = number of variables

NE = number of equality constraints

NI = number of inequality constraints

IMAX = maximum iteration limit on the number of unidirectional searches; ALGØR stops when IMAX searches have been made.

NPRINT = print interval: initial values and the final search point information are printed automatically by ALGØR. If this is all that is desired set NPRINT = 0. If search point information is desired every J searches, set NPRINT = J, where $J \in \{1, 2, \ldots, IMAX\}$

IHE = 0 if NE = 0 or if all initial equality constraint multipliers for the Multiplier method are initialized to zero
= 1 if NCYCL = 0 and any 1 or all of the equality constraint multipliers are initialized to some nonzero constant.
(multipliers can be initialized only when the Multiplier method is used.)

IHI = 0 if NI = 0 or if all initial inequality constraint multipliers for Multiplier method are initialized to zero.
= 1 if NCYCL = 0 and any 1 or all of the inequality constraint multipliers are initialized to some nonzero constant (multipliers can be initialized only when the Multiplier method is used.)

ICØN = 0 if there are no constant gradient components listed in SUBRØUTINE CØNST
= 1 if there are constant gradient components listed in SUBRØUTINE CØNST

NCYCL = 0 if NE = 0 and NI = 0, (i.e., unconstrained), or if the Multiplier method is used.
= IMAX + 1 if Penalty method is used
= J if J cycles† of Penalty method are desired before the Multiplier phase is introduced, where $J \in \{1, 2, \ldots, IMAX\}$

NSRCH = K, where K is the maximum number of unidirectional searches in each cycle. Typically K = (N+1) or (2N+2) if Penalty method is used, and K = (2N+1) or (2N+2) if Multiplier or Penalty/Multiplier method is used.

---

†A cycle generally consists of between N+1 and # unidirectional searches, where the upper bound # is given by NSRCH.

NUP = 0 causes the update information to be printed when the multipliers and/or weights are updated.
= 1 suppresses update information

DATA CARD 3:

```
      READ 200, EPS1, EPS2, EPS3
  200 FØRMAT (3D10.5)
```

EPS1 = Convergence criterion; ALGØR stops when GMAG < EPS1 and DELX < EPS3, where GMAG is the magnitude of the gradient of the augmented Lagrangian, and DELX is the magnitude change in x between consecutive search points. This test is made at the end of a cycle, after the multipliers and weights have been updated. Typical values of EPS1 are (1.D-06 to 1.D-08)

EPS2,EPS3 = Update criterion; multipliers and/or penalty weights are updated at the end of a cycle or anytime after N+1 searches of a cycle if GMAG < EPS2 and DELX < EPS3. EPS2 does not have to equal EPS3, but typical values are (.001, .001) to (.0001, .0001)

DATA CARD 4:

```
      READ 300,(X(I),(I=1,N)
  300 FØRMAT (8D10.5)
```

This card provides the initial x_i values. List 8 values per card with as many cards as necessary to list all N values X(1), ..., X(N). The last card may have less than 8 values.

DATA CARD 5:

```
      READ 400,W1M,W2M,W3M,W1MAXM,W2MAXM,W3MAXM,WFM
  400 FØRMAT (7D10.5)
```

This card provides the multiplier phase penalty weights. This card must not be included if NE = 0 and NI = 0 (i.e., unconstrained). Otherwise if the Penalty method only is used, a blank card is included. If the Multiplier method or Penalty/Multiplier method is used, the Multiplier-phase penalty weights are initialized as:

W1M = initial weight for all equality constraints
= 0 if NE = 0

W2M = initial weight for inequality constraint functions with associated positive-valued multipliers
= 0 if NI = 0

W3M = initial weight for inequality constraint functions with zero-valued multipliers
= 0 if NI = 0

W1MAXM = maximum value of W1M, W1M is held fixed at W1MAXM when W1M $\geq$ W1MAXM
= 0 if NE = 0

W2MAXM = maximum value of W2M; W2M is held fixed at W2MAXM when W2M $\geq$ W2MAXM
= 0 if NI = 0

W3MAXM = maximum value of W3M; W3M is held fixed at W3MAXM when W3M $\geq$ W3MAXM
= 0 if NI = 0

WFM = factor by which W1, W2, and W3 are updated through multiplication by WFM, i.e., $W1M_{(new)} = WFM*W1M_{(old)}$. Typically $(2.0 \leq WFM \leq 4.0)$
= 1 if weights are to be held fixed at initial values

Note: For a scaled problem usually W1M, W2M, and W3M are initialized in the range from 0.25 to 2.0, and W1MAXM, W2MAXM, and W3MAXM are in the range from 16.0 to 64.0. If a constraint breakthrough occurs, preventing convergence, initial and maximum values of all parameters except WFM should be increased and the problem rerun.

DATA CARD 6:

```
      READ 400,W1,W2,W3,W1MAX,W2MAX,W3MAX,WF
  400 FØRMAT (7D10.5)
```

This card must not be included if NE = 0 and NI = 0 (i.e., an unconstrained problem is to be solved.)

The card must not be included if NCYCL = 0 (i.e., the Multiplier method is used.)

If the Penalty method or the Penalty/Multiplier method is used, the penalty phase penalty parameters are initialized as:

W1 = initial weight for all equality constraints
= 0 if NE = 0

W2 = initial weight for all inequality constraints
= 0 if NI = 0

W3 = W2

W1MAX = maximum value for W1; W1 is held fixed at W1MAX when W1 $\geq$ W1MAX
= 0 if NE = 0

W2MAX = maximum value for W2; W2 is held fixed at W2MAX when W2 $\geq$ W2MAX
= 0 if NI = 0

W3MAX = W2MAX

WF = factor by which W1, W2, and W3 are updated through multiplication by WF (see WFM). Typically ($2.0 \leq WF \leq 4.0$)
= 1 if weights are to be held fixed at initial value.

Note: For a scaled problem, usually W1 = W2 = W3 = 1.0 and W1MAX = W2MAX = W3MAX are in the range from 50.0 to 100.0. If a constraint breakthrough occurs, initial and maximum values of all parameters except WF should be increased, and the problem rerun.

DATA CARD(S) 7:

```
      READ 300,(HE(I),I=1,NE)
  300 FØRMAT (8D10.5)
```

This card must not be included if IHE = 0.

If any one or all equality constraint multipliers are to be initialized to some nonzero constant, all

multipliers must be initialized through data cards. Multipliers equal to zero must be set to zero. List 8 HE values per card with as many cards as necessary to list all NE of the HE values HE(1), ..., HE(NE). The last card may have less than 8 values.

DATA CARDS(S) 8:

```
      READ 300,(HI(J),J=1,NI)
  300 FØRMAT (8D10.5)
```

This card must not be included if IHI = 0.

If any one or all inequality constraint multipliers are to be initialized to some positive nonzero constant, all multipliers must be initialized through data cards. Multipliers equal to zero must be set to zero. List 8 HI values per card with as many cards as necessary to list all NI of the HI values HI(1), ..., HI(NI). The last card may have less than 8 values.

Stacking Data Decks

Data decks may be stacked for as many runs on the same problem as desired, each with different initial conditions, or different methods of solution. Each data deck must contain the correct order and number of data cards as described above. To stack data, a loop must be introduced into the main calling program such that ALGØR is called once for each data deck included.

C.3. Control Cards

The system control cards for the XDS Sigma 7 computer at MSU for any problem to be solved by ALGØR are:

```
!JOB----------
!LIMIT--------
!FØRTRAN LS,GØ,ADP

          user supplied routines

!LOPE (GØ),(EXEC),(EF,(ALGØR, account number for
 ALGØR))

          data deck 1

          data deck 2

          etc.
```

Note: Subroutine ALGØR is written for single precision calculations to work on most computers. The XDS Sigma 7 at MSU has a control statement for "automatic double precision" (ADP) as shown above. If this program is to be run on a different computer system, modifications must be made to comply with that system for double precision, which may mean modifications to the subroutines contained in ALGØR. It is advisable to run under double precision. Problems may be run under single precision, but convergence may be slower, or may not be obtained at all due to the

numerical accuracy required for the unidirectional search to be efficient as the optimum value of x is approached.

If modifications must be made to the subroutines, insertion of the specification statement

IMPLICIT REAL*8 (A-H, Ø-Z)

into each subroutine will allow the algorithm to operate in double precision.

If storage is limiting such that all variables cannot be double precision, the specification statement

IMPLICIT REAL*8 Arg_1, Arg_2, ... etc.

will allow only those variables named to be double precision. In order of importance, the argument list should include: (ALFA, D1,D2,D3,A1)(X,XT,AL,F,FE,FI,HE,HI),(GF, GE,GI,GAL),(R,H,FM1,FM2).

It should be remembered that most of the above are vectors, and storage requirements will change with the size of the problem.

To better illustrate the requirements for proper implementation of the subroutines and data, two sample problems are given (problems T1 and T2) in the next sections, showing proper user supplied routines and data.

C.4. Results of Test Problem T1

The user supplied routines, data, and computer results for a typical run of test problem T1 are given in this section.

C: USER SUPPLIED ROUTINES

```
      DIMENSION X(2),XT(2),GF(2),GO(2),GAL(2),R(2),DLX(2),DLG(2)
      DIMENSION XS(2),H(4),FM1(4),FM2(4)
      CALL ALGOR(X,XT,XS,FE,FES,HE,FI,FIS,HI,GF,GE,GI,GO,GAL,DLX,DLG,
     &H,FM1,FM2,R)
      STOP
      END

      SUBROUTINE FXNS(X,F,FE,FI)
      DIMENSION X(1),FE(1),FI(1)
      F = 100.*(X(1)**2-X(2))**2 + (1.-X(1))**2
      RETURN
      END

      SUBROUTINE GRAD(X,GF,GE,GI)
      DIMENSION X(1),GF(1),GE(1),GI(1)
      A = X(1)**2 - X(2)
      GF(1) = 400.*A*X(1) - 2.*(1.-X(1))
      GF(2) = -200.*A
      RETURN
      END

      SUBROUTINE CONST(GF,GE,GI)
      DIMENSION GF(1),GE(1),GI(1)
      RETURN
      END
```

C: DATA DECK

```
BANANA VALLEY
     2     0     0    60    14     0     0     0     0     6     0
.0000001   .000001    .000001
-1.2        1.
```

BANANA VALLEY

```
############################## INITIAL VALUES ##############################

(X)..N =     2          CONSTRAINTS:    (=)..NE =     0    (>/=)..NI =     0
EPS1 =    .100D-07           EPS2 =    .100D-05          EPS3 =    .100D-05

MAXIMUM ITERATION LIMIT...    60  ; OUTPUT PRINTED EVERY  14 ITERATION(S)
WEIGHT/MULTIPLIER UPDATE INFORMATION IS PRINTED

EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
         UNCONSTRAINED, NO MULTIPLIERS

INEQUALITY CONSTRAINT MULTIPLIERS, HI(1),...,HI(NI)
         UNCONSTRAINED, NO MULTIPLIERS

X VALUES, X(1),...,X(N)
 -.120000D 01        .100000D 01

$$$ F =    .242000D 02    AUG. LAG. =    .242000D 02    GMAG =    .232868D 03
############################################################################
```

```
********************** SEARCH POINT ***     14 **********************
FXNS CALLS ...      64      GRAD CALLS ...      15      ALFA =    .112789D=02
DELX =    .245894D=02       DELG =    .198073D=01       RMAG =    .218012D 01
SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)           NG = 0
  .212378D 01       =.492394D 00
$$$ F =    .120135D 01    AUG. LAG. =    .120135D 01    GMAG =    .227951D 01
X VALUES, X(1),...,X(N)
 =.945753D=01         .146503D=01


********************** SEARCH POINT ***     28 **********************
FXNS CALLS ...     118      GRAD CALLS ...      29      ALFA =    .944467D=02
DELX =    .632900D=02       DELG =    .138707D 01       RMAG =    .670113D 00
SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)           NG = 1
  .631008D 00         .225568D 00
$$$ F =    .192922D 00    AUG. LAG. =    .192922D 00    GMAG =    .121087D 01
X VALUES, X(1),...,X(N)
  .564473D 00         .312940D 00


********************** SEARCH POINT ***     42 **********************
FXNS CALLS ...     151      GRAD CALLS ...      43      ALFA =    .242750D 01
DELX =    .641029D=06       DELG =    .229086D=05       RMAG =    .264070D=06
SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)           NG = C
  .118744D=06         .235866D=06
$$$ F =    .755349D=24    AUG. LAG. =    .755349D=24    GMAG =    .386626D=10
X VALUES, X(1),...,X(N)
  .100000D 01         .100000D 01
```

```
 ################## CONVERGENCE CRITERION SATISFIED ##################
 .................. FINAL SEARCH POINT VALUES TO FOLLOW ................

************************ SEARCH POINT ***     43 ************************

FXNS CALLS ...     152      GRAD CALLS ...      44      ALFA =    .1000000D 01

DELX =    .217515D-12       DELG =    .386626D-10       RMAG =    .217521D-12

SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)            NG = 0
 =.130121D-12       =.174310D-12

$$$ F =    .0000000D 00    AUG. LAG. =    .0000000D 00    GMAG =    .0000000D 00

X VALUES, X(1),...,X(N)
  .1000000D 01        .1000000D 01

*****************************************************************************
```

C.5. Results of Test Problem T2

The user supplied routines, data, and computer results for a typical run of test problem T2 are given in this section. Note that a loop has been introduced into the main calling program calling SUBRØUTINE ALGØR once for each data deck. Although there are 4 stacked data decks, only the printout of the first run is included here.

C: USER SUPPLIED ROUTINES

```
      DIMENSION X(3),XT(3),GF(3),GO(3),GAPF(3),R(3),DLX(3),DLG(3)
      DIMENSION FE(1),HE(1),FI(1),HI(1)
      DIMENSION H(9),FM1(9),FM2(9),GE(3),GI(3)
      DIMENSION XS(3),FES(1),FIS(1)
      DO 10 I=1,4
      CALL ALGOR(X,XT,XS,FE,FES,HE,FI,FIS,HI,GF,GE,GI,GO,GAL,DLX,DLG,
     &H,FM1,FM2,R)
   10 CONTINUE
      STOP
      END

      SUBROUTINE FXNS(X,F,FE,FI)
      DIMENSION X(1),FE(1),FI(1)
      F = -X(2)
      FE(1) = X(1)*X(1) + X(2)*X(2) + X(3)*X(3) -1.
      FI(1) = 1. + X(1) - X(2) - X(2)
      RETURN
      END

      SUBROUTINE GRAD(X,GF,GE,GI)
      DIMENSION X(1),GF(1),GE(1),GI(1)
      GE(1) = X(1) + X(1)
      GE(2) = X(2) + X(2)
      GE(3) = X(3) + X(3)
      RETURN
      END

      SUBROUTINE CONST(GF,GE,GI)
      DIMENSION GF(1),GE(1),GI(1)
      GF(2) = -1.
      GI(1) = 1.
      GI(2) = -2.
      RETURN
      END
```

```
C:   DATA DECKS  ( 4 STACKED;  NOTE LOOP IN CALLING PROGRAM)

AROUND THE WORLD:  MULTIPLIER METHOD, MULTIPLIERS INITIALLY ZERO
     3    1    1    50    7    0    0    1    0    7    1
 .000001    .0001      .0001
 .1         .1         .1
 .25        .25        .25        1.         1.         1.         2.
AROUND THE WORLD:  MULTIPLIER METHOD, MULTIPLIERS INITIALIZED
     3    1    1    50    3    1    1    1    0    7    1
 .000001    .0001      .0001
 .1         .1         .1
 .25        .25        .25        1.         1.         1.         2.
 .2
 .25
AROUND THE WORLD:  PENALTY METHOD
     3    1    1    50    1    0    0    1   51    7    1
 .000001    .0001      .0001
 .1         .1         .1

 1.         1.         1.         100.       100.       100.       4.
AROUND THE WORLD:  PENALTY/MULTIPLIER METHOD;  2 CYCLES PENALTY
     3    1    1    50    1    0    0    1    2    7    1
 .000001    .0001      .0001
 .1         .1         .1
 .25        .25        .25        1.         1.         1.         2.
 1.         1.         1.         16.        16.        16.        4.
```

```
AROUND THE WORLD

############################## INITIAL VALUES ##############################

(X)..N =     3          CONSTRAINTS:    (=)..NE =     1     (>/=)..NI =     1
EPS1 =    .100D-05            EPS2 =    .100D-03            EPS3 =    .100D-03

MAXIMUM ITERATION LIMIT...   50  ; OUTPUT PRINTED EVERY    7 ITERATION(S)
WEIGHT/MULTIPLIER UPDATE INFORMATION IS PRINTED

MULTIPLIER PHASE IS INTRODUCED AFTER    0 PENALTY WEIGHT UPDATE CYCLES;
EACH CYCLE CONSISTS OF BETWEEN     4 AND     7 ONE DIMENSIONAL SEARCHES
WHERE WEIGHTS ARE HELD FIXED

INITIAL MULTIPLIER PHASE WEIGHTS ARE       .25       .25       .25
RESPECTIVELY, AND ARE UPDATED BY A FACTOR OF     2.00 UNTIL THEY EXCEED
VALUES OF    .100D 01     .100D 01     .100D 01

EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
         MULTIPLIERS ARE INITIALLY ZERO

EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
  .2000000D-01

INEQUALITY CONSTRAINT MULTIPLIERS, HI(1),...,HI(NI)
         MULTIPLIERS ARE INITIALLY ZERO

INEQUALITY CONSTRAINT VALUES, FI(1),...,FI(NI)
  .2900000D 01

X VALUES, X(1),...,X(N)
 =.1000000D 00       =.10000CD 01        .1000000D 00

$$$ F =    .1000000D 01    AUG. LAG. =    .1000010D 01    GMAG =    .1020000D 01
############################################################################
```

```
********************** SEARCH POINT ***      7 **************************
FXNS CALLS ...      22      GRAD CALLS ...       8      ALFA =    .204524D 01
DELX =    .794351D-01       DELG =    .559209D-01       RMAG =    .388389D-01
SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)          NG = 0
  .219574D-03       -.352971D-02        .386776D-01
$$$ F =  -.105068D 01    AUG. LAG. =  -.934250D 00    GMAG =    .154093D-01
X VALUES, X(1),...,X(N)
  .584186D 00         .105068D 01       -.724265D-02


##########################################################################
  MULTIPLIERS AND/OR PENALTY WEIGHTS UPDATED AT SEARCH POINT *      7 **
EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
  .445261D 00
EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
 -.222631D 00
INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)
 -.517181D 00
INEQUAILTY CONSTRAINT MULTIPLIERS, HI(1),..,HI(NI)
  .258590D 00
W1 =    .500000D 00      W2 =    .500000D 00      W3 =    .500000D 00
$$$ F =  -.105068D 01    AUG. LAG. =  -.584950D 00    GMAG =    .195506D 01
##########################################################################


##########################################################################
  MULTIPLIERS AND/OR PENALTY WEIGHTS UPDATED AT SEARCH POINT *     13 **
EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
  .257910D-01
```

```
EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
 =.248422D 00

INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)
 =.382410D=01

INEQUAILTY CONSTRAINT MULTIPLIERS, HI(1),..,HI(NI)
  .296831D 00

W1 =    .100000D 01      W2 =    .100000D 01      W3 =    .100000D 01

$$$ F =  =.817839D 00   AUG. LAG. =  =.797953D 00   GMAG =   .237800D 00

##########################################################################


********************** SEARCH POINT ***     14 **********************

FXNS CALLS ...      35      GRAD CALLS ...      15      ALFA =   .725580D=01

DELX =    .172543D=01       DELG =    .680756D=02       RMAG =   .237800D 00

SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)           NG = 1
  .148472D=01       =.237336D 00        .679277D=06

$$$ F =  =.800618D 00   AUG. LAG. =  =.799999D 00   GMAG =   .680663D=02

X VALUES, X(1),...,X(N)
  .598514D 00         .800618D 00       =.108283D=05


##########################################################################
 MULTIPLIERS AND/OR PENALTY WEIGHTS UPDATED AT SEARCH POINT #     17 **

EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
  .775943D=03

EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
 =.249973D 00

INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)
 =.151393D=02

INEQUAILTY CONSTRAINT MULTIPLIERS, HI(1),..,HI(NI)
  .299859D 00
```

```
W1 =    .100000D 01      W2 =    .100000D 01      W3 =    .100000D 01
$$$ F =   -.800648D 00    AUG. LAG. =   -.799997D 00    GMAG =    .862000D-02
##########################################################################################


************************ SEARCH POINT ***     21 ************************
FXNS CALLS ...      48      GRAD CALLS ...      22      ALFA =    .197176D 01
DELX =    .393372D-06       DELG =    .228989D-06       RMAG =    .199503D-06
SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)              NG = 0
 -.108020D-07       -.119369D-08        .199206D-06
$$$ F =   -.800024D 00    AUG. LAG. =   -.800000D 00    GMAG =    .580936D-07
X VALUES, X(1),...,X(N)
  .599980D 00         .800024D 00       -.562614D-08


##########################################################################################
  MULTIPLIERS AND/OR PENALTY WEIGHTS UPDATED AT SEARCH POINT **     21 **
EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
  .142315D-04
EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
 -.250002D 00
INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)
 -.666643D-04
INEQUAILTY CONSTRAINT MULTIPLIERS, HI(1),...,HI(NI)
  .299993D 00
W1 =    .100000D 01      W2 =    .100000D 01      W3 =    .100000D 01
$$$ F =   -.800024D 00    AUG. LAG. =   -.800000D 00    GMAG =    .327576D-03
##########################################################################################
```

```
##########################################################################
  MULTIPLIERS AND/OR PENALTY WEIGHTS UPDATED AT SEARCH POINT *     25 **

EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
 =.785653D-06

EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
 =.250000D 00

INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)
 =.347119D-05

INEQUAILTY CONSTRAINT MULTIPLIERS, HI(1),..,HI(NI)
  .300000D 00

W1 =    .100000D 01      W2 =    .100000D 01      W3 =    .100000D 01

$$$ F =   =.800001D 00   AUG. LAG. =   =.800000D 00   GMAG =    .143952D-04

##########################################################################


********************** SEARCH POINT ***     28 **********************

FXNS CALLS ...      65      GRAD CALLS ...      29      ALFA =    .807440D 01

DELX =    .828246D-06       DELG =    .532798D-05       RMAG =    .102577D-06

SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)          NG = 0
  .977996D-07       .308588D-07       .222711D-08

$$$ F =   =.800000D 00   AUG. LAG. =   =.800000D 00   GMAG =    .681868D-08

X VALUES, X(1),...,X(N)
  .600000D 00       .800000D 00       .136276D-07


##########################################################################
  MULTIPLIERS AND/OR PENALTY WEIGHTS UPDATED AT SEARCH POINT *     28 **

EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
 =.151289D-06

EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
 =.250000D 00
```

```
INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)
 =.222907D-06
INEQUALITY CONSTRAINT MULTIPLIERS, HI(1),...,HI(NI)
  .300000D 00
W1 =    .100000D 01      W2 =    .100000D 01      W3 =    .100000D 01
$$$ F =   -.800000D 00   AUG. LAG. =  -.800000D 00   GMAG =    .905879D-06
##################################################################################
```

```
################## CONVERGENCE CRITERION SATISFIED ##################
.................. FINAL SEARCH POINT VALUES TO FOLLOW ..................


************************ SEARCH POINT ***     28 ************************

FXNS CALLS ...      65     GRAD CALLS ...      29     ALFA =    .807440D 01

DELX =    .828246D-06      DELG =    .532798D-05      RMAG =    .102577D-06

SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)          NG = 0
  .977996D-07       .308588D-07       .222711D-08

$$$ F =   -.800000D 00   AUG. LAG. =   -.800000D 00   GMAG =    .905879D-06

X VALUES, X(1),...,X(N)
  .600000D 00       .800000D 00       .136276D-07

EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)
 =.151289D-06

EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)
 =.250000D 00

INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)
 =.222907D-06

INEQUALITY CONSTRAINT MULTIPLIERS, HI(1),..,HI(NI)
  .300000D 00

*************************************************************************
```

APPENDIX D

SUMMARY OF NOTATION FOR PROGRAM LISTING

1. FLAGS

ICØN: Convergence flag; initialized (=0) and set (=1) when convergence criterion are satisfied.

IFLAG: Multiplier weight flag; initialized (=0) and set (=1) to initialize penalty weights to multiplier phase weights when NCYCL > 0; set (=2) to initialize multipliers phase weights when NCYCL = 0.

IHE,IHI: IHE is multiplier initialization flag for equality constraints; (=0) if all initial multipliers $HE_i = 0$; (=1) if one or more multipliers are to be read in as data; IHI same as IHE but for inequality constraint multipliers, HI.

IMULT: Multiplier update flag; initialized (=0), set (=1) when multipliers have been updated for the first time.

ISTØP: Termination flag; initialized (=0) and set (=1) when convergence or stopping conditions are satisfied.

NG: Search direction flag; (=0) when R is a conjugate gradient direction, (=1) when R is a negative gradient direction.

NSTP: Flag for ineffective unidirectional search; (=0) for stepsize ALFA > 0; (=1) for ALFA = 0.

NUP: Output flag; (=0) output information printed at each multiplier and/or weight update (=1) suppresses all update output.

2. COUNTERS

IBSD: Counter for bad search direction generated by subroutine DFPRV and poor unidirectional search; set (=0) at beginning of each cycle, and incremented when one of the above is encountered.

IC1: Counter for NFE during a single unidirectional search.

IDFP: Counter for conjugate search directions; (=1) when R is a negative gradient direction; incremented after each consecutive generation of a conjugate search direction.

IT: Iteration counter for print interval NPRINT.

ITØT: Iteration counter; incremented after each unidirectional search.

IUP: Cycle update counter; incremented at the end of each cycle after multipliers and/or weights have been updated.

KSRCH: Iteration counter for the number of iterations in each cycle; reset (=0) at the beginning of each cycle.

NFE: Function evaluation counter; incremented each time subroutine FXNS is called.

NGE: Gradient evaluation counter; incremented each time subroutine GRAD is called.

3. PARAMETERS

CUT: Reduction factor for steps taken along R.

DLØW: Lower bound for a step of size D2 along R.

DUP: Upper bound for initial estimate of ALFA in unidirectional search.

EPS1, EPS2, EPS3: Small numbers used in test criterion for convergence, convergence rate, and updating multipliers and/or weights.

IMAX: Maximum iteration limit for ITØT.

N: Number of x_i variables, i = 1, 2, ..., N.

NCYCL: Number of cycles to be completed in the Penalty function mode before Multiplier mode is introduced.

NE: Number of equality constraints (HE_i), i = 1, 2, ..., NE.

NI: Number of inequality constraints (HI_j), j = 1, 2, ..., NI.

NPRINT: Number of iterations between print intervals; search point status is printed at this interval.

NSRCH: Number of unidirectional searches allowed in one cycle.

W1, W1M, etc.: W1, W2, W3 are penalty weights used in Penalty mode; W1M, W2M, W3M are weights used in Multiplier mode.

W1MAX, W1MAXM, etc.: Upper bounds on W1, and W1M, etc.

WF, WFM: Factor by which weights are updated through multiplication in Penalty or Multiplier mode, respectively.

4. VARIABLES

AL: Value of the augmented Lagrangian at a point x.

ALFA: Best step along search direction R; yields minimum value of the augmented Lagrangian (AL) for unidirectional search.

DELG: Magnitude change in the gradient of augmented Lagrangian (GAL) between consecutive search points;
$||\nabla L_a^{k+1} - \nabla L_a^{k}||$.

DELX: Magnitude change in x between consecutive search points;
$|| x^{k+1} - x^{k}||$.

D1, D2, D3, A1: Stepsizes taken along R.

DS: Storage (save) for D1, D2, etc.

F, FS: F is cost function value at x; FS is storage (save) variable for F.

FE, FES: FE is equality constraint function value vector; FES is storage (save) vector for FE.

FI, FIS: FI is inequality constraint function value; FIS is storage (save) vector for FI.

GAL, GØ: GAL is the gradient vector of the augmented Lagrangian (AL); GØ is a storage (save) vector for GAL.

GE, GF, GI: Gradient vector of FE, F, and FI.

GMAG: Magnitude of the gradient vector of the augmented Lagrangian;
$|| \nabla L_a^{k}||$.

HE, HI: Multiplier vectors for FE and FI.

H, FM1, FM2: Matrices used for the generation of conjugate gradient search directions.

R: Search direction vector.

RMAG: Magnitude of the search direction vector; $||R||$.

SLOPE: Slope of the augmented Lagrangian function at the beginning of a unidirection search where ALFA = 0.

X, XØ: X is vector of x_i variables; XØ is storage (save) vector for x.

XT, XS: Test points evaluated along R which yield information used in the estimation of ALFA; XS is storage (save) vector of XT.

Y1, Y2, Y3, YA1: Value of the augmented Lagrangian at a point XT resulting from a step D1, etc., along R; YS is storage (save) for Y_i.

APPENDIX E

PROGRAM LISTING FOR MULTIPLIER ALGORITHM ALGØR

```
      SUBROUTINE ALGOR(X,XT,XS,FE,FES,HE,FI,FIS,HI,GF,GE,GI,GO,GAL,
     &DLX,DLG,H,FM1,FM2,R)
C
CCC   COMMAND PROGRAM: ORGANIZATION AND OPERATION OF ALGORITHM
C
      DIMENSION X(1),XT(1),XS(1),FE(1),FES(1),HE(1),FI(1)
      DIMENSION FIS(1),HI(1),GF(1),GE(1),GI(1),GO(1),GAL(1)
      DIMENSION DLX(1),DLG(1),H(1),FM1(1),FM2(1),R(1)
      COMMON /C1/N,NE,NI,NEPI/C2/NFE,NGE/C3/W1,W2,W3
      COMMON /C4/W1MAX,W2MAX,W3MAX,WF/C6/DELX,DELG
      COMMON /C9/KSRCH,IBSD/C20/NUP
      COMMON /C10/EPS1,EPS2,EPS3/C11/IMULT,ISTOP,ITOT
      COMMON /C12/NCYCL,NSRCH,IT,IUP,NPRINT,IMAX,NP1
      COMMON /C13/RMAG,ALFA,GMAG,SLOPE/C14/IFLAG
      COMMON /C15/W1M,W2M,W3M,W1MAXM,W2MAXM,W3MAXM,WFM
      COMMON /C16/DUP,DLOW/C18/NG/C17/DS,FS,YS/C19/NSTP
C
CCC   INITIALIZATION
C
      ICONV=0
      CALL INITL(X,F,FE,FI,GF,GE,GI,GO,HE,HI,AL,GAL)
      NN=N*N
      DO 1 I=1,N
    1 XS(I)=X(I)
      IF(NE.EQ.0) GO TO 3
      DO 2 I=1,NE
    2 FES(I)=FE(I)
    3 IF(NI.EQ.0) GO TO 5
      DO 4 I=1,NI
    4 FIS(I)=FI(I)
    5 FS=F
C
CCCCC TEST:  ITERATION LIMIT; GENERATE SEARCH DIRECTION R,
      SEARCH ALONG R TO FIND MINIMUM ALFA, COMPUTE GRADIENT,
      UPDATE SEARCH POINT STATUS
C
   10 IF(ITOT.EQ.IMAX) GO TO 80
      CALL DFPRV(GAL,FM1,FM2,H,DLX,DLG,N,NN,R)
      CALL SEARCH(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,AL)
      CALL GRAD(XT,GF,GE,GI)
      NGE=NGE + 1
      CALL GALAG(GF,GE,GI,FE,FI,HE,HI,GAL)
      CALL DELTA(XT,X,GAL,GO,DLX,DLG,N,GMAG)
C
CCC   UPDATE COUNTERS, TEST:  OUTPUT CONDITIONS
C
      ITOT=ITOT + 1
      KSRCH=KSRCH + 1
```

```
      IT=IT + 1
      IF(IT.EQ.NPRINT) GO TO 110
   20 IF(DELG.LT.EPS2.AND.DELX.LT.EPS3) IBSD=IBSD + 1
      IF(ALFA.EQ.0.) NSTP=1
      IF(NSTP.EQ.1) IBSD=IBSD + 1
      IF(KSRCH.GT.N) GO TO 30
      IF(IBSD.EQ.2) KSRCH=N
      GO TO 10
C
CC    TEST:  UPDATE CRITERION FOR MULTIPLIERS AND/OR PENALTY WEIGHTS
C
   30 IF(KSRCH.EQ.NSRCH) GO TO 40
      IF(GMAG.LT.EPS1.OR.IBSD.GE.3) GO TO 40
      IF(NSTP.EQ.1) GO TO 10
      IF(GMAG.LT.EPS2.AND.DELX.LT.EPS3) GO TO 40
      GO TO 10
C
CC    UPDATE MULTIPLIERS AND/OR PENALTY WEIGHTS
C
   40 IF(NEPI.EQ.0) GO TO 70
      IF(NUP.EQ.0) PRINT 1000,ITOT
      IF(IUP.LT.NCYCL) GO TO 50
      CALL MULTUP(FE,FI,HE,HI)
      IF(IUP.NE.NCYCL) GO TO 50
      IFLAG=1
      IMULT=1
   50 CALL WUP
      CALL AUGLAG(F,FE,FI,HE,HI,AL)
      CALL GALAG(GF,GE,GI,FE,FI,HE,HI,GAL)
      GMAG=0.
      DO 60 I=1,N
   60 GMAG=GMAG + GAL(I)*GAL(I)
      GMAG=SQRT(GMAG)
      IF(NUP.EQ.0) PRINT 1010,F,AL,GMAG
      IUP=IUP + 1
   70 KSRCH=0
      IBSD=0
C
CC    TEST:  CONVERGENCE CRITERION
C
      IF(GMAG.GT.EPS1.OR.DELX.GT.EPS3) GO TO 10
      ICONV=1
C
CC    STOP:  CONVERGENCE; ITERATION LIMIT
CC    .. FINAL SEARCH POINT INFORMATION ..
C
   80 ISTOP=1
      IF(ICONV.EQ.1) GO TO 90
```

```
      PRINT 1020
      GO TO 100
   90 PRINT 1030
  100 PRINT 1040
  110 CALL OUTPUT(XT,R,F,AL,FE,FI,HE,HI)
      IT=0
      IF(ISTOP.EQ.0) GO TO 20
      RETURN
C
CC    ********** FORMATS **********
C
 1000 FORMAT(///,20X,'##########################################################
     &##################',/,20X,' MULTIPLIERS AND/OR PENALTY',
     &' WEIGHTS UPDATED AT SEARCH POINT *',I6,' **')
 1010 FORMAT(/,20X,'$$$ F = ',D13.6,3X,'AUG. LAG. = ',D13.6,3X,
     &'GMAG = ',D13.6,//,20X,'##########################################
     &##########')
 1020 FORMAT('1',20X,'########################## MAXIMUM ITERATION LIMIT #
     &##########')
 1030 FORMAT('1',20X,'########################## CONVERGENCE CRITERION ',
     &'SATISFIED ##########')
 1040 FORMAT(20X,'..................... FINAL SEARCH POINT VALUES TO FOLLOW
     & ...............')
      END
```

```
      SUBROUTINE INITL(X,F,FE,FI,GF,GE,GI,GO,HE,HI,AL,GAL)
C
CC    INITIALIZES ALGORITHM PARAMETERS, AND READS DATA
C
      DIMENSION X(1),GF(1),GO(1),GAL(1),FE(1),HE(1)
      DIMENSION FI(1),HI(1),GE(1),GI(1),ID(18)
      COMMON /C1/N,NE,NI,NEPI/C2/NFE,NGE/C3/W1,W2,W3
      COMMON /C4/W1MAX,W2MAX,W3MAX,WF/C9/KSRCH,IBSD
      COMMON /C10/EPS1,EPS2,EPS3/C11/IMULT,ISTOP,ITOT
      COMMON /C12/NCYCL,NSRCH,IT,IUP,NPRINT,IMAX,NP1
      COMMON /C13/RMAG,ALFA,GMAG,SLOPE/C14/IFLAG
      COMMON /C15/W1M,W2M,W3M,W1MAXM,W2MAXM,W3MAXM,WFM
      COMMON /C16/DUP,DLOW/C18/NG/C20/NUP
C
CC    INITIALIZE COUNTERS, FLAGS, PARAMETERS
C
      KSRCH=0
      IBSD=0
      NG=1
      NFE=1
      NGE=1
      IUP=0
      IT=0
      ITOT=0
      IMULT=0
      IFLAG=0
      ISTOP=0
      GMAG=0.
      DUP=4.
C
CC    READ DATA, SET WEIGHTS
C
      READ 1170,(ID(I),I=1,18)
      READ 1000,N,NE,NI,IMAX,NPRINT,IHE,IHI,ICON,NCYCL,NSRCH,NUP
      READ 1010,EPS1,EPS2,EPS3
      READ 1010,(X(I),I=1,N)
      NP1=N+1
      NEPI=NE+NI
      IF(NEPI.EQ.0) GO TO 20
      READ 1010,W1M,W2M,W3M,W1MAXM,W2MAXM,W3MAXM,WFM
      IF(NCYCL.EQ.0) GO TO 10
      READ 1010,W1,W2,W3,W1MAX,W2MAX,W3MAX,WF
      GO TO 20
   10 IFLAG=2
      IMULT=1
      IUP=1
      CALL WUP
C
```

```
C       INITIALIZE ALL GRADIENT COMPONENTS TO ZERO
CC      SET INITIAL MULTIPLIERS
C
   20 DO 30 I=1,N
   30 GF(I)=0.
      IF(NE.EQ.0) GO TO 70
      K=N*NE
      DO 40 I=1,K
   40 GE(I)=0.
      IF(IHE.EQ.0) GO TO 50
      READ 1010,(HE(I),I=1,NE)
      GO TO 70
   50 DO 60 I=1,NE
   60 HE(I)=0.
   70 IF(NI.EQ.0) GO TO 110
      K=N*NI
      DO 80 J=1,K
   80 GI(J)=0.
      IF(IHI.EQ.0) GO TO 90
      READ 1010,(HI(J),J=1,NI)
      GO TO 110
   90 DO 100 J=1,NI
  100 HI(J)=0.
C
CC      EVALUATE $$F, AUGMENTED LAGRANGIAN (AL), AND NONZERO GRADIENTS
C
  110 CALL FXNS(X,F,FE,FI)
      CALL GRAD(X,GF,GE,GI)
      IF(ICON.EQ.0) GO TO 120
      CALL CONST(GF,GE,GI)
  120 CALL AUGLAG(F,FE,FI,HE,HI,AL)
      CALL GALAG(GF,GE,GI,FE,FI,HE,HI,GAL)
      DO 130 I=1,N
      GO(I)=GAL(I)
  130 GMAG=GMAG + GAL(I)*GAL(I)
      GMAG=SQRT(GMAG)
C
CC      OUTPUT INITIAL VALUES
C
      PRINT 1160
      PRINT 1180,(ID(I),I=1,18)
      PRINT 1020
      PRINT 1030,N,NE,NI
      PRINT 1040,EPS1,EPS2,EPS3
      PRINT 1050,IMAX,NPRINT
      IF(NUP.EQ.1) PRINT 1210
      IF(NUP.EQ.0) PRINT 1220
      IF(NEPI.EQ.0) GO TO 150
```

```
      PRINT 1060,NCYCL,NP1,NSRCH
      IF(NCYCL.EQ.0) GO TO 140
      PRINT 1070,W1,W2,W3,WF,W1MAX,W2MAX,W3MAX
  140 PRINT 1080,W1M,W2M,W3M,WFM,W1MAXM,W2MAXM,W3MAXM
  150 PRINT 1090
      IF(NE.EQ.0) GO TO 160
      IF(IHE.EQ.0) GO TO 170
      PRINT 1130,(HE(I),I=1,NE)
      GO TO 175
  160 PRINT 1140
      GO TO 180
  170 PRINT 1150
  175 PRINT 1190
      PRINT 1130,(FE(I),I=1,NE)
  180 PRINT 1100
      IF(NI.EQ.0) GO TO 190
      IF(IHI.EQ.0) GO TO 200
      PRINT 1130,(HI(J),J=1,NI)
      GO TO 205
  190 PRINT 1140
      GO TO 210
  200 PRINT 1150
  205 PRINT 1200
      PRINT 1130,(FI(J),J=1,NI)
  210 PRINT 1110
      PRINT 1130,(X(I),I=1,N)
      PRINT 1120,F,AL,GMAG
      PRINT 1160
      RETURN
C
CC    ********** FORMATS **********
C
 1000 FORMAT(11I5)
 1010 FORMAT(8D10.5)
 1020 FORMAT(//,20X,'############################## INITIAL VALUES ######
     &##################')
 1030 FORMAT(/,20X,'(X)..N = ',I4,9X,'CONSTRAINTS:    (=)..NE = ',I4,
     &4X,'(>/=)..NI = ',I4)
 1040 FORMAT(20X,'EPS1 = ',D10.3,10X,'EPS2 = ',D10.3,10X,'EPS3 = ',
     &D10.3)
 1050 FORMAT(/,20X,'MAXIMUM ITERATION LIMIT...',I5,2X,
     &'; OUTPUT PRINTED EVERY',I4,' ITERATION(S)')
 1060 FORMAT(/,20X,'MULTIPLIER PHASE IS INTRODUCED AFTER ',I3,
     &' PENALTY WEIGHT UPDATE CYCLES;',/,20X,'EACH CYCLE CONSISTS OF ',
     &'BETWEEN ',I4,' AND ',I5,' ONE DIMENSIONAL SEARCHES',/,20X,
     &'WHERE WEIGHTS ARE HELD FIXED')
 1070 FORMAT(/,20X,'INITIAL PENALTY PHASE WEIGHTS ARE ',3(F7.2,2X),/,
     &20X,'RESPECTIVELY, AND ARE UPDATED BY A FACTOR OF ',F7.2,
```

```
     &' UNTIL THEY EXCEED',/,20X,'VALUES OF ',3(D10.3,2X))
1080 FORMAT(/,20X,'INITIAL MULTIPLIER PHASE WEIGHTS ARE ',3(F7.2,2X),/,
     &20X,'RESPECTIVELY, AND ARE UPDATED BY A FACTOR OF ',F7.2,
     &' UNTIL THEY EXCEED',/,20X,'VALUES OF ',3(D10.3,2X))
1090 FORMAT(/,20X,'EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)')
1100 FORMAT(/,20X,'INEQUALITY CONSTRAINT MULTIPLIERS, HI(1),...,HI(NI)')
1110 FORMAT(/,20X,'X VALUES, X(1),...,X(N)')
1120 FORMAT(/,20X,'$$$ F = ',D13.6,3X,'AUG. LAG. = ',D13.6,3X,
     &'GMAG = ',D13.6,/,20X,'#####################################
     &##################')
1130 FORMAT(20X,D13.6,5X,D13.6,5X,D13.6,5X,D13.6)
1140 FORMAT(29X,'UNCONSTRAINED, NO MULTIPLIERS')
1150 FORMAT(29X,'MULTIPLIERS ARE INITIALLY ZERO')
1160 FORMAT('1')
1170 FORMAT(18A4)
1180 FORMAT(20X,18A4)
1190 FORMAT(/,20X,'EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)')
1200 FORMAT(/,20X,'INEQUALITY CONSTRAINT VALUES, FI(1),...,FI(NI)')
1210 FORMAT(20X,'WEIGHT/MULTIPLIER UPDATE INFORMATION SUPPRESSED')
1220 FORMAT(20X,'WEIGHT/MULTIPLIER UPDATE INFORMATION IS PRINTED')
     END
```

```
      SUBROUTINE DFPRV(GN,FM1,FM2,H,DLX,DLG,N,NN,R)
C
CC    COMPUTES SEARCH DIRECTION R, VIA DFP METHOD WITH RESETS OF
CC    R = -GAL BASED UPON UPDATE CRITERION, ETC.
C
      DIMENSION GN(1),DLX(1),DLG(1),R(1),H(1),FM1(1),FM2(1)
      COMMON /C9/KSRCH,IBSD/C13/RMAG,ALFA,GMAG,SLOPE
      COMMON /C18/NG/C19/NSTP
C
      NG=0
      IF(NSTP.EQ.1) IDFP=N+1
      IF(KSRCH.EQ.0) GO TO 10
      IF(IDFP.NE.(N+1)) GO TO 40
   10 NG=1
      IDFP=1
      NSTP=0
C
CC    R = -GAL
C
      DO 20 I=1,NN
   20 H(I)=0.
      INDX=0
      DO 30 I=1,N
      H(INDX+I)=1.
      R(I)= -GN(I)
   30 INDX=INDX+N
      GO TO 90
C
CC    R VIA DFP
C
   40 DEN1=0.
      DO 50 I=1,N
   50 DEN1=DEN1+DLX(I)*DLG(I)
      CALL MATMUL(DLX,DLX,FM1,N,1,N)
      CALL MATMUL(DLG,H,DLX,1,N,N)
      DEN2=0.
      DO 60 I=1,N
   60 DEN2=DEN2+DLX(I)*DLG(I)
      CALL MATMUL(H,DLG,R,N,N,1)
      CALL MATMUL(R,DLX,FM2,N,1,N)
      DO 70 ID=1,NN
   70 H(ID) = H(ID) + FM1(ID)/DEN1 - FM2(ID)/DEN2
      CALL MATMUL(H,GN,R,N,N,1)
      DO 80 I=1,N
   80 R(I)=-R(I)
      IDFP=IDFP +1
C
CC    TEST FOR A GOOD R; IF R BAD SET R = -GAL
```

```
C
   90 RMAG=0.
      SLOPE=0.
      DO 100 I=1,N
      RMAG=RMAG+R(I)*R(I)
  100 SLOPE=SLOPE+R(I)*GN(I)
      RMAG=SQRT(RMAG)
      IF(RMAG.LT.(.001*GMAG)) GO TO 110
      IF(SLOPE.LE.0.) RETURN
  110 IBSD=IBSD + 1
      GO TO 10
      END
```

```
      SUBROUTINE MATMUL(A,B,C,N1,N2,N3)
C
CC    MATRIX MULTIPLICATION       C = A*B    ... VECTOR FORMAT
CC    USED IN DFPRV..A(N1,N2), B(N2,N3), C(N1,N3)
C
      DIMENSION A(1),B(1),C(1)
C
      II1=0
      II2=0
      DO 30 I=1,N1
      DO 20 J=1,N3
      SUM=0.
      IND=0
      DO 10 L=1,N2
      SUM=SUM + A(II1+L)*B(IND+J)
   10 IND=IND+N3
   20 C(II2+J)=SUM
      II1=II1+N2
      II2=II2+N3
   30 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE SEARCH(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,AL)
C
C     CONDUCTS SEARCH ALONG R TO FIND STEPSIZE ALFA THAT MINIMIZES
C     THE AUGMENTED LAGRANGIAN  AL = F(X + ALFA*R)
C
      DIMENSION X(1),XT(1),XS(1),FE(1),FES(1),HE(1)
      DIMENSION FI(1),FIS(1),HI(1),R(1)
      COMMON /C1/N,NE,NI,MEPI/C2/NFE,NGE/C3/W1,W2,W3
      COMMON /C13/RMAG,ALFA,GMAG,SLOPE/C18/NG
      COMMON /C16/DUP,DLOW/C17/DS,FS,YS
C
C     SAVE STARTING POINT (D1,Y1), SET INITIAL STEP D2, EVALUATE Y2
C
      DS=0.
      YS=AL
      D1=0.
      IC1=0
      Y1=AL
      D2=1.
      IF(NG.EQ.1) D2=.1
      DLOW = .001
      CUT=3.
      IF(RMAG.LT.200.) GO TO 10
      D2=10./RMAG
      IF(D2.GT.DLOW) GO TO 10
      D2=DLOW
      CUT=6.
      DLOW = .01*D2
   10 CALL VALUE(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,D2,Y2)
      IF(Y1-Y2)100,200,300
C
C     SEARCH SCHEME WHEN Y2 > Y1    AND    D1=0.
C
  100 CALL DMIN(D1,D2,D3,Y1,Y2,Y3,SLOPE,1,ALFA)
      IF(D2.LE.DLOW) GO TO 400
      IF(ALFA.GT.(.1*D2)) GO TO 110
      D2=ALFA
      IF(D2.LT.DLOW) D2=DLOW
      GO TO 10
  110 CALL VALUE(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,ALFA,AL)
      IF(Y1-AL)130,120,120
  120 CALL DMIN(D1,ALFA,D2,Y1,AL,Y2,SLOPE,2,A1)
      IF(A1.GE.(.9*ALFA).AND.A1.LE.(1.1*ALFA)) GO TO 410
      ALFA=A1
      GO TO 400
  130 D2=D2/CUT
      GO TO 10
C
```

```
C       SEARCH SCHEME WHEN Y2 <= Y1
C
  200 D3=3.*D2
  210 IC1=IC1+1
      CALL VALUE(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,D3,Y3)
      IF(Y2-Y3)220,240,240
  220 CALL DMIN(D1,D2,D3,Y1,Y2,Y3,SLOPE,2,ALFA)
      CALL VALUE(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,ALFA,AL)
      IF(Y1-AL)230,410,410
  230 ALFA=D2
      GO TO 400
  240 D1=D2
      Y1=Y2
      D2=D3
      Y2=Y3
      IF(IC1.LT.10) GO TO 200
      GO TO 230
C
CC      SEARCH SCHEME WHEN Y2 < Y1   AND   D1=0.
C
  300 CALL DMIN(D1,D2,D3,Y1,Y2,Y3,SLOPE,1,A1)
      IF(A1.GE.(.9*D2).AND.A1.LE.(1.1*D2)) GO TO 310
      GO TO 320
  310 ALFA=D2
      AL=Y2
      GO TO 410
  320 IF(A1.LT.0.) GO TO 200
      IF(A1-DUP)330,340,340
  330 CALL VALUE(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,A1,YA1)
      IF(A1-D2)350,310,380
  340 D3=DUP
      GO TO 210
  350 IF(Y2-YA1)360,370,370
  360 D2=A1
      Y2=YA1
      GO TO 200
  370 ALFA=A1
      AL=YA1
      GO TO 410
  380 CALL DMIN(D1,D2,A1,Y1,Y2,YA1,SLOPE,2,ALFA)
      IF(YA1-Y2)390,400,400
  390 IF(ALFA.GE.(.9*A1).AND.ALFA.LE.(1.1*A1)) GO TO 370
      D3=A1
      Y3=YA1
      GO TO 240
C
CC      OPTIMUM STEPSIZE ALFA FOUND FOR THIS SEARCH
C
```

```
400 CALL VALUE(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,ALFA,AL)
410 DO 420 I=1,N
420 XT(I) = XS(I)
    IF(NE.EQ.0) GO TO 440
    DO 430 I=1,NE
430 FE(I) = FES(I)
440 IF(NI.EQ.0) GO TO 460
    DO 450 I=1,NI
450 FI(I) = FIS(I)
460 F=FS
    ALFA=DS
    AL=YS
    RETURN
    END
```

```
      SUBROUTINE DMIN(D1,D2,D3,Y1,Y2,Y3,SLOPE,K,ALF)
C
CC    ASSUMES QUADRATIC FORM  AL = A*(D1-ALF)**2 + B*(D1-ALF) + C
CC    MINIMUM ALF = D1 - B/(2*A)  FOUND BY QUADRATIC FIT OF
CC    DATA:  (POINT/SLOPE,POINT) OR (3 POINTS)
C
      GO TO (10,20),K
C
CC    QUADRATIC FIT; PSP
C
   10 TP1=-SLOPE*D2
      ALF=(.5*TP1*D2)/(Y2-Y1+TP1)
      RETURN
C
CC    QUADRATIC FIT; 3P
C
   20 D21=D2-D1
      D31=D3-D1
      TP1=(Y2-Y1)/D21
      TP2=(Y3-Y1)/D31
      ALF=D1-.5*(D21*TP2-D31*TP1)/(TP1-TP2)
      RETURN
      END
```

```
      SUBROUTINE VALUE(X,XT,XS,FE,FES,HE,FI,FIS,HI,R,F,D,Y)
C
CC    EVALUATES AUGMENTED LAGRANGIAN Y, AT STEPSIZE D,
CC    ALONG SEARCH DIRECTION R; SAVES POINT IF BETTER
C
      DIMENSION X(1),XT(1),XS(1),FE(1),FES(1),HE(1)
      DIMENSION FI(1),FIS(1),HI(1),R(1)
      COMMON /C1/N,NE,NI,NEPI/C2/NFE,NGE/C3/W1,W2,W3
      COMMON /C17/DS,FS,YS
C
      DO 5 I=1,N
    5 XT(I) = X(I) + D*R(I)
      CALL FXNS(XT,F,FE,FI)
      NFE=NFE+1
      CALL AUGLAG(F,FE,FI,HE,HI,Y)
      IF(Y.GT.YS) RETURN
      DO 10 I=1,N
   10 XS(I)=XT(I)
      IF(NE.EQ.0) GO TO 30
      DO 20 I=1,NE
   20 FES(I)=FE(I)
   30 IF(NI.EQ.0) GO TO 50
      DO 40 I=1,NI
   40 FIS(I)=FI(I)
   50 FS=F
      DS=D
      YS=Y
      RETURN
      END
```

```
      SUBROUTINE DELTA(XN,XO,GN,GO,DLX,DLG,N,GMAG)
C
CC    COMPUTES:  NORM(XNEW-XOLD), NORM(GNEW-GOLD), NORM(GNEW)
C
      DIMENSION XN(1),XO(1),GN(1),GO(1),DLX(1),DLG(1)
      COMMON /C6/DELX,DELG
C
      DELX=0.
      DELG=0.
      GMAG=0.
      DO 10 I=1,N
      DLX(I)=XN(I)-XO(I)
      DLG(I)=GN(I)-GO(I)
      DELX=DELX+DLX(I)*DLX(I)
      DELG=DELG+DLG(I)*DLG(I)
      GMAG=GMAG+GN(I)*GN(I)
      XO(I)=XN(I)
   10 GO(I)=GN(I)
      DELX=SQRT(DELX)
      DELG=SQRT(DELG)
      GMAG=SQRT(GMAG)
      RETURN
      END
```

```
      SUBROUTINE MULTUP(FE,FI,HE,HI)
C
CC    UPDATES MULTIPLIERS
C
      DIMENSION FE(1),HE(1),FI(1),HI(1)
      COMMON /C1/N,NE,NI,NEPI/C3/W1,W2,W3/C20/NUP
C
      IF(NE.EQ.0) GO TO 20
      IF(NUP.EQ.1) GO TO 5
      PRINT 1070
      PRINT 1020,(FE(I),I=1,NE)
    5 TW1=W1+W1
      DO 10 I=1,NE
   10 HE(I)=HE(I)=TW1*FE(I)
      IF(NUP.EQ.1) GO TO 20
      PRINT 1000
      PRINT 1020,(HE(I),I=1,NE)
   20 IF(NI.EQ.0) RETURN
      IF(NUP.EQ.1) GO TO 25
      PRINT 1090
      PRINT 1020,(FI(J),J=1,NI)
   25 TW2=W2+W2
      TW3=W3+W3
      DO 40 J=1,NI
      IF(HI(J).LE.0.) GO TO 30
      HI(J)=HI(J)=TW2*FI(J)
      IF(HI(J).LT.0.) HI(J)=0.
      GO TO 40
   30 IF(FI(J).LT.0.) HI(J) = =TW3*FI(J)
   40 CONTINUE
      IF(NUP.EQ.1) RETURN
      PRINT 1010
      PRINT 1020,(HI(J),J=1,NI)
      RETURN
C
CC    ********** FORMATS **********
C
 1000 FORMAT(/,20X,'EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)')
 1010 FORMAT(/,20X,'INEQUAILTY CONSTRAINT MULTIPLIERS, HI(1),..,HI(NI)')
 1020 FORMAT(20X,D13.6,5X,D13.6,5X,D13.6,5X,D13.6)
 1070 FORMAT(/,20X,'EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)')
 1090 FORMAT(/,20X,'INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)')
      END
```

```
      SUBROUTINE WUP
C
CC    UPDATES PENALTY WEIGHTS AND SETS MULTIPLIER WEIGHTS
CC    AT BEGINNING OF MULTIPLIER PHASE
C
      COMMON /C1/N,NE,NI,NEPI/C14/IFLAG/C20/NUP
      COMMON /C3/W1,W2,W3/C4/W1MAX,W2MAX,W3MAX,WF
      COMMON /C15/W1M,W2M,W3M,W1MAXM,W2MAXM,W3MAXM,WFM
C
      IF(IFLAG.GE.1) GO TO 20
C
CC    PENALTY WEIGHT UPDATE
C
      IF(NE.EQ.0) GO TO 10
      IF(W1.LT.W1MAX) W1=WF*W1
      IF(W1.GT.W1MAX)W1=W1MAX
   10 IF(NI.EQ.0) GO TO 30
      IF(W2.LT.W2MAX) W2=WF*W2
      IF(W2.GT.W2MAX) W2=W2MAX
      IF(W3.LT.W3MAX) W3=WF*W3
      IF(W3.GT.W3MAX) W3=W3MAX
      GO TO 30
C
CC    INITIALIZE MULTIPLIER WEIGHTS
C
   20 W1=W1M
      W2=W2M
      W3=W3M
      W1MAX=W1MAXM
      W2MAX=W2MAXM
      W3MAX=W3MAXM
      WF=WFM
      IF(IFLAG.EQ.2) GO TO 40
      IFLAG=0
   30 IF(NUP.EQ.0) PRINT 1000,W1,W2,W3
      RETURN
   40 IFLAG=0
      RETURN
C
CC    ********** FORMATS **********
C
 1000 FORMAT(/,20X,'W1 = ',D13.6,5X,'W2 = ',D13.6,5X,'W3 = ',D13.6)
      END
```

```
      SUBROUTINE AUGLAG(F,FE,FI,HE,HI,AL)
C
CC    COMPUTES AUGMENTED LAGRANGIAN (AL)
C
      DIMENSION FE(1),HE(1),FI(1),HI(1)
      COMMON /C1/N,NE,NI,NEPI/C3/W1,W2,W3
C
      IF(NEPI.EQ.0) GO TO 80
      FLAG=0.
      FEW1=0.
      FIW2=0.
      FIW3=0.
      IF(NE.EQ.0) GO TO 30
      DO 20 I=1,NE
      FEW1=FEW1 + FE(I)*FE(I)
   20 FLAG=FLAG + HE(I)*FE(I)
   30 IF(NI.EQ.0) GO TO 70
      DO 60 I=1,NI
      IF(HI(I).LE.0.) GO TO 50
      FLAG=FLAG + HI(I)*FI(I)
      FIW2=FIW2 + FI(I)*FI(I)
      GO TO 60
   50 IF(FI(I).LT.0.) FIW3=FIW3 + FI(I)*FI(I)
   60 CONTINUE
   70 AL = F = FLAG + W1*FEW1 + W2*FIW2 + W3*FIW3
      RETURN
   80 AL=F
      RETURN
      END
```

```
      SUBROUTINE GALAG(GF,GE,GI,FE,FI,HE,HI,GAL)
C
CC    COMPUTES N COMPONENTS OF THE GRADIENT OF THE AUGMENTED
CC    LAGRANGIAN (GAL)
C
      DIMENSION GF(1),GAL(1),GE(1),FE(1),HE(1),GI(1),FI(1),HI(1)
      COMMON /C1/N,NE,NI,NEPI/C3/W1,W2,W3
C
      IF(NEPI.EQ.0) GO TO 110
      DO 100 I=1,N
      GLAG=0.
      GEW1=0.
      GIW2=0.
      GIW3=0.
      IF (NE.EQ.0) GO TO 20
      INDX=I
      DO 10 J=1,NE
      GLAG=GLAG+HE(J)*GE(INDX)
      GEW1=GEW1+FE(J)*GE(INDX)
   10 INDX=INDX+N
   20 IF(NI.EQ.0) GO TO 100
      INDX=I
      DO 40 J=1,NI
      IF(HI(J).LE.0.) GO TO 30
      GLAG=GLAG+HI(J)*GI(INDX)
      GIW2=GIW2+FI(J)*GI(INDX)
      GO TO 40
   30 IF(FI(J).LT.0.)GIW3=GIW3+FI(J)*GI(INDX)
   40 INDX=INDX+N
  100 GAL(I) = GF(I) - GLAG + 2.*(W1*GEW1 + W2*GIW2 + W3*GIW3)
      RETURN
  110 DO 120 I=1,N
  120 GAL(I) = GF(I)
      RETURN
      END
```

```
      SUBROUTINE OUTPUT(X,R,F,AL,FE,FI,HE,HI)
C
CC    OUTPUTS GENERAL AND TERMINAL SEARCH POINT INFORMATION
C
      DIMENSION X(1),R(1),FE(1),HE(1),FI(1),HI(1)
      COMMON /C1/N,NE,NI,NEPI/C2/NFE,NGE/C3/W1,W2,W3
      COMMON /C6/DELX,DELG/C11/IMULT,ISTOP,ITOT
      COMMON /C13/RMAG,ALFA,GMAG,SLOPE/C18/NG
C
CC    GENERAL SEARCH POINT INFORMATION
C
      PRINT 1000,ITOT
      PRINT 1010,NFE,NGE,ALFA
      PRINT 1020,DELX,DELG,RMAG
      PRINT 1030,NG
      PRINT 1040,(R(I),I=1,N)
      PRINT 1050,F,AL,GMAG
      PRINT 1060
      PRINT 1040,(X(I),I=1,N)
      IF(ISTOP.EQ.0) RETURN
C
CC    TERMINAL INFORMATION
C
      IF(NEPI.EQ.0) GO TO 30
      IF(IMULT.EQ.1) GO TO 10
      NUP=1
      CALL MULTUP(FE,FI,HE,HI)
   10 IF(NE.EQ.0) GO TO 20
      PRINT 1070
      PRINT 1040,(FE(I),I=1,NE)
      PRINT 1080
      PRINT 1040,(HE(I),I=1,NE)
   20 IF(NI.EQ.0) GO TO 30
      PRINT 1090
      PRINT 1040,(FI(J),J=1,NI)
      PRINT 1100
      PRINT 1040,(HI(J),J=1,NI)
   30 PRINT 1110
      PRINT 1120
      RETURN
C
CC    **********  FORMATS  **********
C
 1000 FORMAT(///,20X,'************************ SEARCH POINT ***',I6,
     &' ************************')
 1010 FORMAT(/,20X,'FXNS CALLS ... ',I6,5X,'GRAD CALLS ... ',I6,5X
     &'ALFA = ',D13.6)
 1020 FORMAT(/,20X,'DELX = ',D13.6,6X,'DELG = ',D13.6,6X,
```

```
     &'RMAG = ',D13.6)
1030 FORMAT(/,20X,'SEARCH DIRECTION TO THIS POINT, R(1),...,R(N)',
     &10X,'NG = ',I1)
1040 FORMAT(20X,D13.6,5X,D13.6,5X,D13.6,5X,D13.6)
1050 FORMAT(/,20X,'$$$ F = ',D13.6,3X,'AUG. LAG. = ',D13.6,3X,
     &'GMAG = ',D13.6)
1060 FORMAT(/,20X,'X VALUES, X(1),...,X(N)')
1070 FORMAT(/,20X,'EQUALITY CONSTRAINT VALUES, FE(1),...,FE(NE)')
1080 FORMAT(/,20X,'EQUALITY CONSTRAINT MULTIPLIERS, HE(1),...,HE(NE)')
1090 FORMAT(/,20X,'INEQUALITY CONSTRAINT VALUES, FI(1),...FI(NI)')
1100 FORMAT(/,20X,'INEQUALITY CONSTRAINT MULTIPLIERS, HI(1),..,HI(NI)')
1110 FORMAT(/,20X,'**********************************************************
     &**********')
1120 FORMAT('1')
     END
```

REFERENCES

REFERENCES

1. J. Abadie, "On the Kuhn-Tucker Theorem," Chapter 2 of Nonlinear Programming, J. Abadie (Editor), North-Holland Publishing Company, Amsterdam, 1967.

2. K.J. Arrow, F.J. Gould, and S.M. Howe, "A General Saddle Point Result for Constrained Optimization," Institue of Statistics Mimeo Series No. 774, University of North Carolina, Chapel Hill, September 1971.

3. _____, and L. Hurwicz, "Reduction of Constrained Maxima to Saddle Point Problems," in Third Berkeley Symposium on Mathematical Statistics and Probability, J. Neyman (Editor) University of California Press, Berkeley, 1956.

4. _____, and R.M. Solow, "Gradient Methods for Constrained Maxima, with Weakened Assumptions," in Studies in Linear and Nonlinear Programming, K. Arrow, L. Hurwicz, and H. Uzawa (Editors), Stanford University Press, Stanford, 1958.

5. J. Asaadi, "A Computational Comparison of Some Nonlinear Programs," Mathematical Programming, Vol. 4, No. 2, pp. 144-154, April 1973.

6. J.W. Bandler, and R.E. Seviora, "Current Trends in Network Optimization," IEEE Transactions on Microwave Theory and Techniques, Vol. MIT-18, No. 12, pp. 1159-1170, December 1970.

7. E.J. Beltrami, "A Constructive Proof of the Kuhn-Tucker Multiplier Rule," Journal of Mathematical Analysis and Applications, Vol. 26, pp. 297-306, 1969.

8. M.J. Box, D. Davies, and W.H. Swann, Nonlinear Optimization Techniques, (IMPERIAL Chemical Industries LIMITED MONOGRAPH No. 5), Oliver and Royd, Edinburgh 1, 1969.

9. J. Bracken, and G.P. McCormick, Selected Applications of Nonlinear Programming, John Wiley and Sons, New York, 1968.

10. A. Brameller, and K.L. Lo, "A Review of Minimization Techniques with Reference to Power System Engineering," in Real-Time Control of Electrical Power Systems, Symposium Proceedings, Edmund Handschin, (Editor), Brown, Boveri and Company, Ltd., Baden, Switzerland, 1971.

11. C.W. Carroll, "The Created-Response-Surface Technique for Optimizing Nonlinear Restrained Systems," Operations Research, Vol. 9, pp. 169-184, March-April 1961.

12. A.R. Colville, "A Comparative Study on Nonlinear Programming Codes," IBM New York Scientific Center Report No. 320-2949, June, 1968.

13. R. Courant, "Variational Methods for the Solution of Problems of Equilibrium and Vibrations," Bulletin of the American Math Society, Vol. 49, pp. 1-23, January 1943.

14. H.B. Curry, "The Method of Steepest Descent for Nonlinear Minimization Problems." Quarterly of Applied Mathematics, Vol. 2, pp. 258-261, October 1944.

15. W.C. Davidon, "Variable Metric Method for Minimization," Atomic Energy Commission Research and Development Report, ANL-5990, 1959.

16. D. Davies and W.H. Swann, "Review of Constrained Optimization," Chapter 12 of Optimization, R. Fletcher (Editor), Academic Press, London, 1969.

17. G. Debreu, "Definite and Semidefinite Quadratic Forms," Econometrica, Vol. XX, pp. 295-300, 1952.

18. J.B. Dennis, Mathematical Programming and Electrical Networks, Massachusetts Institute of Technology Press, 1959.

19. R.J. Duffin, E.L. Peterson, and C. Zener, Geometric Programming - Theory and Application, John Wiley and Sons, Inc., New York, 1967.

20. A.V. Fiacco, and G.P. McCormick, "The Sequential Unconstrained Minimization Technique for Nonlinear Programming, A primal-Dual Method," Management Science, Vol. 10, No. 2, pp. 360-366, January, 1964.

21. _____, and _____, "Computational Algorithm for the Sequential Unconstrained Minimization Technique for Nonlinear Programming," Management Science, Vol. 10, No. 4, pp. 601-617, July, 1964.

22. _____, and _____, "Extensions of SUMT for Nonlinear Programming; Equality Constraints and Extrapolation," Management Science, Vol. 12, No. 11, pp. 816-828, July, 1966.

23. _____, and _____, Nonlinear Programming: Sequential Unconstrained Minimization Techniques, Wiley, New York, 1968.

24. R. Fletcher, Optimization, Academic Press, London, 1969.

25. _____, "A Class of Methods for Nonlinear Programming with Termination and Convergence Properties," Chapter 6 of Integer and Nonlinear Programming, J. Abadie (Editor), North-Holland Publishing Company, Amsterdam, 1970.

26. _____, and S.A. Lill, "A Class of Methods for Nonlinear Programming, II; Computational Experience," in Nonlinear Programming, J.B. Rosen, O.L. Mangasarian, and K. Ritter (Editors), Academic Press, pp. 67-92, 1971.

27. _____, and M.J.D. Powell, "A Rapidly Convergent Descent Method for Minimization," The Computer Journal, Vol. 7, pp. 163-168, July, 1963.

28. _____, and C.M. Reeves, "Function Minimization by Conjugate Gradients," The Computer Journal, Vol. 7, pp. 149-154, July, 1964.

29. P.C. Haarhoff, and J.D. Buys, "A New Method for the Optimization of a Nonlinear Function to Nonlinear Constraints," The Computer Journal, Vol. 13, pp. 178-184, May, 1970.

30. M.R. Hestenes, "Multiplier and Gradient Methods," Journal of Optimization Theory and Applications, Vol. 4, pp. 303-320, November, 1969.

31. _____, "Multiplier and Gradient Methods," in Computing Methods in Optimization Problems - 2, pp. 143-163, L.A. Zadeh, L.W. Neustadt, and A.V. Balakrishman (Editors), Academic Press, New York, 1969.

32. _____, and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," Journal of Research of the National Bureau of Standards, Vol. 49, No. 6, pp. 409-436, December, 1952.

33. F. John, "Extreme Problems with Inequalities as Side Conditions," in Studies and Essays, Courant Anniversary Volume, K.O. Friedrichs, O.E. Neugebauer, and J.J. Stoker (Editors), Wiley, New York, pp. 187-204, 1948.

34. D.L. Keefer, and B.S. Gottfried, "Differential Constraint Scaling in Penalty Function Optimization," Abstract, Bulletin Operations Research Society of America, Vol. 18, Sup. 2, p. B-184, October, 1970.

35. R.P. King, "Necessary and Sufficient Conditions for Inequality Constrained Extreme Values," Industrial and Engineering Chemistry Fundamentals, Vol. 5, pp. 484-489, November, 1966.

36. B.W. Kort, and D.P. Bertsekas, "Multiplier Methods for Convex Programming," Proceedings of the IEEE Decision and Control Conference, San Diego, California, December, 1973.

37. H.W. Kuhn, and A.W. Tucker, "Nonlinear Programming," in Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, J. Neyman, (Editor), University of California Press, Berkeley, pp. 481-493, 1951.

38. F.A. Lootsma, "Hessian Matrices of Penalty Functions for Solving Constrained Optimization Problems," Philips Research Reports, Vol. 24, pp. 322-330, August, 1969.

39. _____, "Penalty-Function Performance of Several Unconstrained-Minimization Techniques," Philips Research Reports, Vol. 27, 1972.

40. M.J. Lowe, "Programming Instructions for Subroutine ALGØR," Electronics Research Laboratory Report No. 2773, Montana State University, Bozeman, Montana, December, 1973.

41. D.G. Luenberger, "Convergence Rate of a Penalty-Function Scheme," Journal of Optimization Theory and Applications, Vol. 7, pp. 39-51, January, 1971.

42. _____, Introduction to Linear and Nonlinear Programming, Addison-Wesley Publishing Company, Menlo Park, California, 1973.

43. G.P. McCormick, "Penalty Function Versus Non-Penalty Function Methods for Constrained Nonlinear Programming Problems," Mathematical Programming, Vol. 1 pp. 217-238, 1971.

44. O.L. Mangasarian, "Unconstrained Lagrangians in Nonlinear Programming," Computer Sciences Technical Report No. 174, University of Wisconsin, Madison, March, 1973.

45. A. Miele, E.E. Cragg, R.R. Iyer, and A.V. Levy, "Use of the Augmented Penalty Function in Mathematical Programming Problems, Part 1," Journal of Optimization Theory and Applications, Vol. 8, No. 2, pp. 115-130, 1971.

46. A. Miele, E.E. Cragg, and A.V. Levy, "Use of the Augmented Penalty Function in Mathematical Programming Problems, Part 2," Journal of Optimization Theory and Applications, Vol. 8, No. 2, pp. 131-153, 1971.

47. _____, H.Y. Huang, and J.W. Contrell, "Gradient Methods in Mathematical Programming, Part 1, Review of Previous Techniques" Aero - Astronautics Report No. 55, Rice University, Houston, Texas, 1969.

48. _____, P.E. Moseley, and E.E. Cragg, "Numerical Experiments on Hestenes' Method of Multipliers for Mathematical Programming Problems," Aero - Astronautics Report No. 85, Rice University, Houston, Texas, 1971.

49. _____, _____, and _____, "A Modification of the Method of Multipliers for Mathematical Programming Problems," Aero-Astronautics Report No. 86, Rice University, Houston, Texas, 1971.

50. _____, _____, A.V. Levy, and G.M. Coggins, "On the Method of Multipliers for Mathematical Programming Problems," Journal of Optimization Theory and Application, Vol. 10, pp. 1-33, 1972.

51. W. Murray, "Ill-Conditioning in Barrier and Penalty Functions Arising in Constrained Nonlinear Programming," Proc. of the Sixth International Symposium on Mathematical Programming, Princeton University, August 13-20, 1967.

52. J.D. Pearson, "Variable Metric Methods of Minimization," The Computer Journal, Vol. 12, pp. 171-178, May, 1969.

53. D.A. Pierre, Optimization Theory with Applications, Wiley, New York, 1969.

54. _____, "Nonlinear Programming: Research Goals and Preliminary Results," Research Memorandum S10, Electronics Research Laboratory, Montana State University, Bozeman, Montana, December, 1971.

55. D.A. Pierre, "Multiplier Algorithms for Nonlinear Programming," Presented at the 8th International Symposium on Mathematical Programming, August 27-31, 1973.

56. _____, "Two Multiplier Algorithms for Nonlinear Programming," September, 1973. (Revised version of a paper presented at the 8th International Symposium on Mathematical Programming, August 27-31, 1973.)

57. M.J.D. Powell, "An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives," The Computer Journal, Vol. 7, pp. 155-162, July, 1964.

58. _____, "A Method for Nonlinear Constraints in Minimization Problems," in Optimization, R. Fletcher (Editor), Academic Press, New York, 1969.

59. R.T. Rockafellar, "New Applications of Duality in Convex Programming," presented at the 7th International Symposium on Mathematical Programming (the Hague, 1970), published in the Proceedings of the 4th Conference on Probability (Brasov, Romania, 1971).

60. _____, "Penalty Methods and Augmented Lagrangians in Nonlinear Programming," Proc. 5th IFIP Conference on Optimization Techniques, Rome, May, 1973.

61. _____, "Augmented Lagrange Multiplier Functions and Duality in Nonconvex Programming," SIAM Journal on Control, to be published (January, 1974).

62. _____, "A Dual Approach to Solving Nonlinear Programming Problems by Unconstrained Optimization," Mathematical Programming, to be published.

63. _____, "The Multiplier Method of Hestenes and Powell Applied to Convex Programming," Journal of Optimization Theory and Applications, to be published.

64. R.T. Rockafellar, "A Counter Example," personal correspondence to D.A. Pierre, August 31, 1973.

65. H.H. Rosenbrock, "An Automatic Method for Finding the Greatest of the Least Value of a Function," The Computer Journal, Vol. 3, pp. 175-184, October, 1960.

66. B.V. Shah, R.J. Buehler, and O. Kempthorn, "Some Algorithms for Minimizing a Function of Several Variables," Journal of the Society for Industrial and Applied Mathematics, Vol. 12, pp. 74-92, March, 1964.

67. D. Tabak, and B.C. Kuo, Optimal Control by Mathematical Programming, Prentice Hall, Englewood Cliffs, New Jersey, 1971.

68. C. van de Panne, and W. Popp, "Minimum-Cost Cattle Feed under Probalistic Protein Constraints," Management Science, Vol. 9, No. 3, pp. 405-430, 1963.

69. S. Vajda, "Nonlinear Programming and Duality," in Nonlinear Programming, J. Abadie (Editor), North-Holland Publishing Company, Amsterdam, pp. 6-7, 1967.

70. P. Wolfe, "Methods of Nonlinear Programming," Chapter 6 of Nonlinear Programming, J. Abadie (Editor), North-Holland Publishing Company, Amsterdam, 1967.

71. W.I. Zangwill, "Minimizing a Function without Calculating Derivatives," The Computer Journal, Vol. 10, pp. 293-296, November, 1967.

72. _____, Nonlinear Programming: A Unified Approach, Prentice-Hall, Englewood Cliffs, New Jersey, 1969.

73. G. Zoutendijk, Methods of Feasible Directions, Elsevier Publishing Company, Amsterdam, 1960.