AD-A039 094 NEW JERSEY INST OF TECH NEWARK F/G 13/13
A FAST BOUNDARY TRACKING ALGORITHM FOR CONSTRAINED NONLINEAR MA--ETC(U)
MAR 77 J MORADI, M PAPPAS N00014-75-C-0987
UNCLASSIFIED NJIT-NV-12 NL

1 OF 2
ADA039094



AD A039094

Contract No. ONR N00014-75-C0987

A Fast Boundary Tracking Algorithm for Constrained Nonlinear Mathematical Programming Problems

by

Jacob Moradi

and

Michael Pappas



DDC
RECEIVED
MAY 9 1977
C

March 1977

NJIT Report No. NV 12

Reproduction in whole or in part is permitted for any purpose of the United States Government. Distribution of this document is unlimited.

New Jersey Institute of Technology
323 High Street
Newark, New Jersey 07102

AD No.
DDC FILE COPY

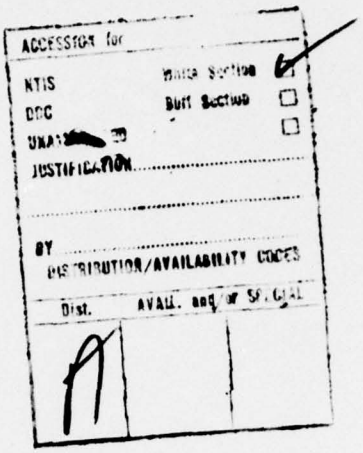
ABSTRACT

A fast search algorithm for the solution of nonlinear mathematical programming optimization problems is presented in this Report. A gradient search procedure is combined with a "Boundary Tracking"(BT) method using the feasible direction finding method of Zoutendijk for generating a feasible starting direction along the feasible-infeasible boundary.

The algorithm is applied to the minimum weight design of submersible, circular, cylindrical shells reinforced by equally spaced "T" type frames. This problem had produced algorithm failure in two earlier studies and was only recently solved by the Direct Search-Feasible Direction Algorithm (DSFD) which was shown by recent comparison studies to be among the fastest and most reliable mathematical programming methods available. The BT procedure was found to be substantially faster than DSFD, producing a solution with about one-eigth the effort required by DSFD.

In a general comparison study a code based on the BT algorithm was compared with twenty other codes representing most of the popular numerical optimization methods on ten test problems. These problems are such that majority of the codes tested failed to solve more than half of them. The new code proved superior to all others in overall generality and efficiency. It solved all problems and was the fastest code on the constrained problems.

TABLE OF CONTENTS



	Page
ABSTRACT	i
LIST OF FIGURES	iv
LIST OF TABLES	v
NOMENCLATURE	vi
CHAPTER 1 - INTRODUCTION	1
1.1 Optimization in Design	1
1.2 General Strategy of Mathematical Programming	4
1.3 Motivation and Structure	6
CHAPTER 2 - DESCRIPTION OF GENERAL STRATEGY OF THE *BOUNDARY TRACKING ALGORITHM*	10
2.1 Introduction	10
2.2 Location of the Behavior Boundary	10
2.3 Movement Along the Behavior Boundary	15
2.4 Search Termination Description	18
CHAPTER 3 MATHEMATICAL DEVELOPMENT	21
3.1 Boundary Location	21
3.1.1 Identification of Proper Constraints	21
3.1.2 The Boundary Locating Method	22
3.2 Mathematical Representation of General Strategy	26
3.2.1 Movement to the Boundary, Starting Point in the Feasible Region	28

TABLE OF CONTENTS (Cont'd)

		Page
3.2.2	Movement to the Boundary, Starting Point in the Infeasible Region	31
3.2.3	Initiation of Movement Along the Boundary	31
3.2.4	Movement Along the Boundary	34
CHAPTER 4 - DESIGN EXAMPLE		42
4.1	Problem Statement	42
4.2	Problem Formulation	43
4.2.1	Objective Function	43
4.2.2	Constraint Equations	44
4.3	Discussion of Results	50
4.3.1	Code Description	50
4.3.2	Code Application	51
CHAPTER 5 - COMPARISON STUDY		56
CHAPTER 6 - CONCLUSION		68
LIST OF REFERENCES		70
APPENDIX A - User Instructions		72
APPENDIX B - Program Listing		81

LIST OF FIGURES

		Page
Figure 1:	General Strategy of the Boundary Tracking Method	11
Figure 2:	Block Diagram of the General Strategy	12
Figure 3:	Block Diagram of the Termination Procedure	20
Figure 4:	Selection of a Proper Constraint	23
Figure 5:	The Boundary Locating Method	27
Figure 6:	Flow Chart of the Procedure for Movement to the Boundary	29
Figure 7:	Flow Chart of the Procedure for Movement Along the Boundary	35
Figure 8:	Flow Chart of the Termination Procedure	39
Figure 9:	Typical Shell Cross Section	45

LIST OF TABLES

		Page
Table 1:	Synthesis path for the shell design problem using DSFD method	53
Table 2:	Synthesis path for the shell design problem using the Boundary Tracking method	54
Table 3:	Code Descriptions	57
Table 4:	Performance of Optimization codes	58
Table 5:	Relative ranking of Optimization codes	60
Table 6:	Ranking of Optimization codes in Problems with Behavior Constraints	65

NOMENCLATURE

A	= cross sectional area of the frame
A_j	= number of active behavior constraints
$B_k(x)$	= behavior function
b	= web thickness
b'	= flange thickness
c	= distance from midplane of the shell to the surface of the frame
C_1	= an arbitrary large constant
C_2	= maximum flange to plating thickness ratio
d	= distance from the midplane of the hull plating to the neutral axis of the frame
E	= tensile modulus
EI_f	= bending stiffness of a frame
e	= eccentricity
e_1, e_2, e_3	= arbitrary small positive constraints
f, $f(x)$	= objective function
f_a, $\overline{f}_a$	= efficiency criteria
G	= shear modulus
GJ_f	= torsional stiffness of a frame
$g(x)$, $g_j(x)$	= constraint function and the jth constraint function
$g^*(x)$	= the smallest proper constraint
$g(t)$	= the constraint $g^*(x)$ as a function of a single variable t
H_E, H_M	= frame deflection parameters [see equations 3.11 - 3.12]
h	= web height
I	= number of variables

J = number of constraint equations

K = number of proper constraints

k_j = arbitrary small positive number

L' = distance between frames

L_k = lower limit on behavior function

L_s = length of the shell

m = axial wave number

N_a = numerical success rating

N_1, N_2 = arbitrary selected numbers

n = circumferential wave number

n_a = number of problems solved by code "a"

p = hydrostatic pressure

P_k = set of proper constraints

P = progress towards solution (section 4.2)

$p_{cg}(n,m)$ = gross collapse pressure

$p_{cs}(n,m)$ = collapse pressure of a shell panel

p^*_{cg}, p^*_{cs} = minimum collapse pressures

Q_p = total radial load

R = radius of the shell

R_{min}, R_{max} = minimum and maximum radius of the shell

R_i^+, R_i^- = active lower and upper regional constraints

S_j = factor of safety

t = skin thickness

t_{ap} = normalized cpu time

u = best movement direction vector

V_D = displacement volume of the cylinder

V_f = volume of the frame

V_s = volume of plating

V_w = volume of frame web

W = weight of the hull

W_j = deflection parameter

x, x_i = design variable vector and its components respectively

α = step size

α_{min} = minimum step size

Γ = frame deflection parameter

γ_s = specific weight of material

γ_w = specific weight of immersion fluid

ε_j = constraint activity limit

μ = poison's ratio

σ_b = frame bending stress

σ_c = compressive hoop stress

σ_{fa} = allowable frame stress

σ_{pa} = allowable plating stress

σ_r = axial stress

σ_t = maximum frame stress

σ_ϕ = hoop stress

$|\phi|$ = magnitude of arbitrary vector function ϕ

$\nabla\phi$ = the gradient of arbitrary vector function ϕ

$(\phi)^T$ = transpose of arbitrary vector function ϕ

SUPERSCRIPTS

b	= base number
ℓ	= comparison value
L	= Lower limit
n	= value at the n^{th} direction finding problem
r	= number of redesign cycles
s	= number of iterations for locating the IF boundary
U	= upper limit
0	= initial

CHAPTER 1

INTRODUCTION

1.1 Optimization in Design

Optimal design problems may be treated by many different methods, for example, ordinary and variational calculus, mathematical programming (MP), and some special techniques, such as the fully stressed concept used in structural design [1-10][1].

Of the above mentioned methods, MP procedures [11] seem to be the most flexible methods available for optimal design synthesis since they treat the broadest range of engineering problems and are easily adaptable.

The concept of design optimization requires that a quantity to be minimized or maximized be designated and that this quantity be expressed as a function of the design variables. This function is called the "merit" or "objective" function, and can be written in the form

$$f = f(x_i) \qquad i = 1,2,\ldots I \quad (1\text{-}1)$$

where x_i are the design variables and I is the number of design variables. Generally, the design variables are not free to take on any value, but are subject to constraints governing their range or the behavior of the design.

---

[1] Numbers in brackets designate references at the end of the thesis.

When limits on behavior are specified, a quantitative measure of the behavior as a function of the variables, called the "behavior function", $B(x_i)$ is required. Thus, if K behavior functions are specified, one can write K equations of the form

$$\begin{aligned} &L_k \leq B_k \leq U_k && k = 1, 2,\ldots m \\ &B_k \leq U_k && k = m + 1, m + 2,\ldots n \\ &L_k \leq B_k && k = n + 1, n + 2,\ldots p \end{aligned} \tag{1.2}$$

where L_k is the lower bound on B_k, and U_k the upper limit. L_k and U_k may be functions of x_i. Behavior constraints representing the above behavior function limits can be written as follows

$$\begin{aligned} &g_j = U_j - B_j \geq 0 && j = 1, 2,\ldots m \\ &g_j = B_{j-m} - L_{j-m} \geq 0 && j = m + 1, m + 2,\ldots 2m \\ &g_j = U_{j-2m} - B_{j-2m} \geq 0 && j = 2m + 1, 2m + 2,\ldots 2m + n \\ &g_j = B_{j-2m+n} - L_{j-2m+n} \geq 0 && j = 2m + n + 1,\ldots 2m + n + p \end{aligned} \tag{1.3}$$

All the above equations may be written as

$$\begin{aligned} &g_j \geq 0 && j = 1, 2,\ldots J \\ & && J = 2m + n + p \end{aligned} \tag{1.4}$$

Regional limits are often imposed by manufacturing or other considerations. These are of the form

$$x_i^L \leq x_i \leq x_i^U \tag{1.5}$$

x_i^L and x_i^U are constants and represent the minimum and maximum values of x_i respectively. Here regional constraints are distinguished from behavior constraints since any constraint of the form

$$A \leq x_i \leq B \tag{1.6}$$

where A and B are constants, may be treated more simply than the general form. Not all the x_i need be subject to such limits. A detailed discussion of regional constraints can be found in Reference [9].

Some variables may also be restricted to certain discrete values. In structural design these values can represent material properties, geometric available shapes and sizes, or thickness gages.

The concept of a merit surface is important to the understanding of the redesign process. The function $f(x_i)$ can be considered as a surface in I + 1 dimensional space where the coordinate axes are the x_i and y where the value of y is given by $f(x_i)$. The constraints delineate the region of interest within which the optimum is to be found. Points which satisfy the constraint equations are called "acceptable" or "feasible" points. All other points are called "unacceptable" or "infeasible". The set of all feasible points constitute the "feasible region" and all infeasible points define the "infeasible region". The surface between these regions is called the Infeasible-Feasible (IF) boundary. Those portions of the IF boundary where at least one behavior constraint is zero are called the "behavior" boundary. Points away from the constraint surface are called "free", and those on these surfaces are called "bound" points. The merit surface is explored to find the highest point in the acceptable region. The algorithms commonly employed in such problems usually start the exploration from a free acceptable point. The variables are generally treated as

continuous quantities. If discrete variables are encountered, they may be treated initially as continuous and, upon finding an optimum, a local exploration is made to find the optimal discrete value [12].

1.2 General Strategy of Mathematical Programming

Like all optimization methods the MP methods try to find those values x for which the merit function $f(x_i)$ will be minimized (or maximized). The problem may be stated as follows:

Find those x_i that produce the

$$\min. f(x_i) \tag{1.7}$$

such that all the constraints

$$g_j(x_i) \geq 0 \qquad j = 1, 2, \ldots J \tag{1.8}$$

and the equality constraints

$$g_k(x_i) = 0 \qquad k = J + 1, J + 2, \ldots K \tag{1.9}$$

are satisfied, where the x_i are the I real valued variables x_1, $x_2, \ldots x_I$ in I-dimentional Euclidian space, and constraints g_1, $g_2, \ldots g_K$ are real-valued real functions in that space.

Most MP methods do not treat the equality constraints directly. These constraints may, for example, be converted to inequality constraints of the form

$$\varepsilon \leq g_k(x_i) \leq \varepsilon \tag{1.10}$$

where ε is an arbitrary small number. One can also treat an equality constraint by solving for one variable in terms of the others, thereby eliminating a variable and constraint.

MP methods are based on searching strategies. The search usually starts from an arbitrarily selected point x^0 and, by means of some local search strategy, a set of points x^r is generated such that $f(x^r) < f(x^{r-1})$ while also satisfying all the constraint equations. In the majority of design problems, either f and/or some or all of the constraints g_j are nonlinear in x, and therefore one has a nonlinear, constrained optimization problem.

None of the available nonlinear optimization methods guarantees an optimal solution (when it exists). There are two major difficulties. First, many of the design problems are not unimodal that is, they have more than one local optimum. The nonlinear methods are, however, designed only to locate local optima and thus the global optimum may not be attained. Secondly the search algorithm may simply fail to find even a local optimum.

The prudent designer thus usually tries several different starting points. If all the search paths terminate at the same point, then optimality is assumed and the problem can be considered to be unimodal. However, if the termination point is different for different search paths, then either the best design is accepted or additional starting points are tried in an attempt to find a still better design. Such a decision draws on the designer's experience and judgment. The failure to converge to the same point may be due either to the presence of several local optima, or to algorithm failure. Failure is said to occur when the algorithm ter-

minates at a point significantly away from the nearest local optimum. An excellent discussion of the difficulties involved in the solution of unconstrained nonlinear programming problems is provided in Reference [12]. These difficulties are greatly compounded when constraints are added. For convenience, since this and other MP methods are capable only of locating a local optimum, in the discussion below the term optimum should be taken to mean local rather than global optimum.

1.3 Motivation and Structure

Many efficient algorithms exist for treating linear problems, that is, problems with linear objective functions and constraint equations, and certain simple nonlinear problems [5-7, 13]. Most design problems are, however, nonlinear problems that must be treated by relatively less efficient methods.

There are a number of methods available for solution of such problems [14]. However, there are several difficulties with these methods. For example, the computations required to produce improved designs in the neighborhood of the constraint boundaries are in some cases lengthy and relatively complex. The number of function evaluations is often very large, and, most methods are not reliable, that is, the best points produced by these methods may not be a local optima.

Recently, Eason and Fenton [15] compared a group of seventeen numerical optimization methods. This group of algorithms contains

most of the currently popular procedures. The results of this study indicate that the direct search algorithms are superior with respect to generality, efficiency, running cost, speed, preparation cost and reliability [15]. Unfortunately, none of the methods presented in this study are totally reliable. None of the codes solved all of the ten problems attempted.

In a recent paper, Pappas and Moradi compared a code based on the Direct Search-Feasible Direction Algorithm (DSFD) [16] with those studied by Eason and Fenton [17]. This code solved all the test problems treated in Reference [15]. In addition it also solved a difficult six-variable problem which the relatively reliable DSDA, and popular SUMT, procedures failed to solve [17]. In this study the DSFD based code proved superior to all others in overall generality and efficiency.

Even the relatively efficient DSFD however, required several hundred to several thousand function evaluations to achieve a solution to Eason's and Fenton's test problems. Therefore substantial computational effort and cost would be required by these procedures on problems with computationally demanding functions. Since many important engineering problems are of this type, there is a clear need for a new algorithm which requires a reduced number of function evaluations for solution.

This thesis presents an apparently fast and reliable algorithm which utilizes a search strategy designed to greatly reduce the number

of function evaluations required for solution. This is accomplished by developing a scheme which keeps the search confined to the neighborhood of the behavior boundary, on the premise that this is where the optimum is usually found in constrained problems.

The search moves efficiently along this boundary by making effective use of a relatively small amount of new local function information relying primarily on information generated earlier in the search. Thus, the number of new function evaluations required to sustain movement and thereby the total number of function evaluations required for solution are kept low.

Since the advent of high speed digital computers many accurate but computationally demanding methods have been developed for engineering analysis. Due to the relatively large execution time required for each reanalysis cycle (function evaluation) of many of these techniques and the large number of reanalysis cycles required by most MP procedures, the combination of such analytic methods and MP procedures may be very costly to utilize and are therefore often impractical. This difficulty can seemingly be minimized by use of the Boundary Tracking (BT) algorithm due to its apparent efficiency in reducing the number of function evaluations and therefore total execution time required for solution.

Where the BT algorithm is not available in a compiled form, a simpler algorithm may be preferable for use on small simple problems. In such circumstances the extra compilation time required due to the

greater complexity of the BT algorithm may overshadow the time saved by a reduction in the number of function evaluations required for solution.

CHAPTER 2

DESCRIPTION OF GENERAL STRATEGY OF THE BOUNDARY TRACKING ALGORITHM

2.1 Introduction

The most common strategy for constrained optimization techniques is first to move to the IF boundary from a starting point and then to move along this boundary to the optimum. A typical objective function surface is illustrated in Figure 1.

The general strategy of the Boundary Tracking (BT) algorithm is to first locate a point on the behavior boundary. Once the behavior boundary is located the best direction for movement along this boundary is determined. Movement along this boundary is then continued until the optimum point is obtained.

Figure 2 presents a general block diagram of the BT algorithm.

2.2 Location of the Behavior Boundary

The search starts from an arbitrary starting point x_i^0. If the problem has behavior constraints, these constraints are evaluated at this point. If the starting point proves to be in the feasible region, it is designated as a base point (point 1 Figure 1). A step is then taken in the direction of the gradient of the objective function. All the constraints are evaluated after the step. If the new point is in the feasible region the objective function is evaluated and compared to the last base point. If the function has improved, this new point is called a base point (points 2, 3, 4 in Figure 1) and the search

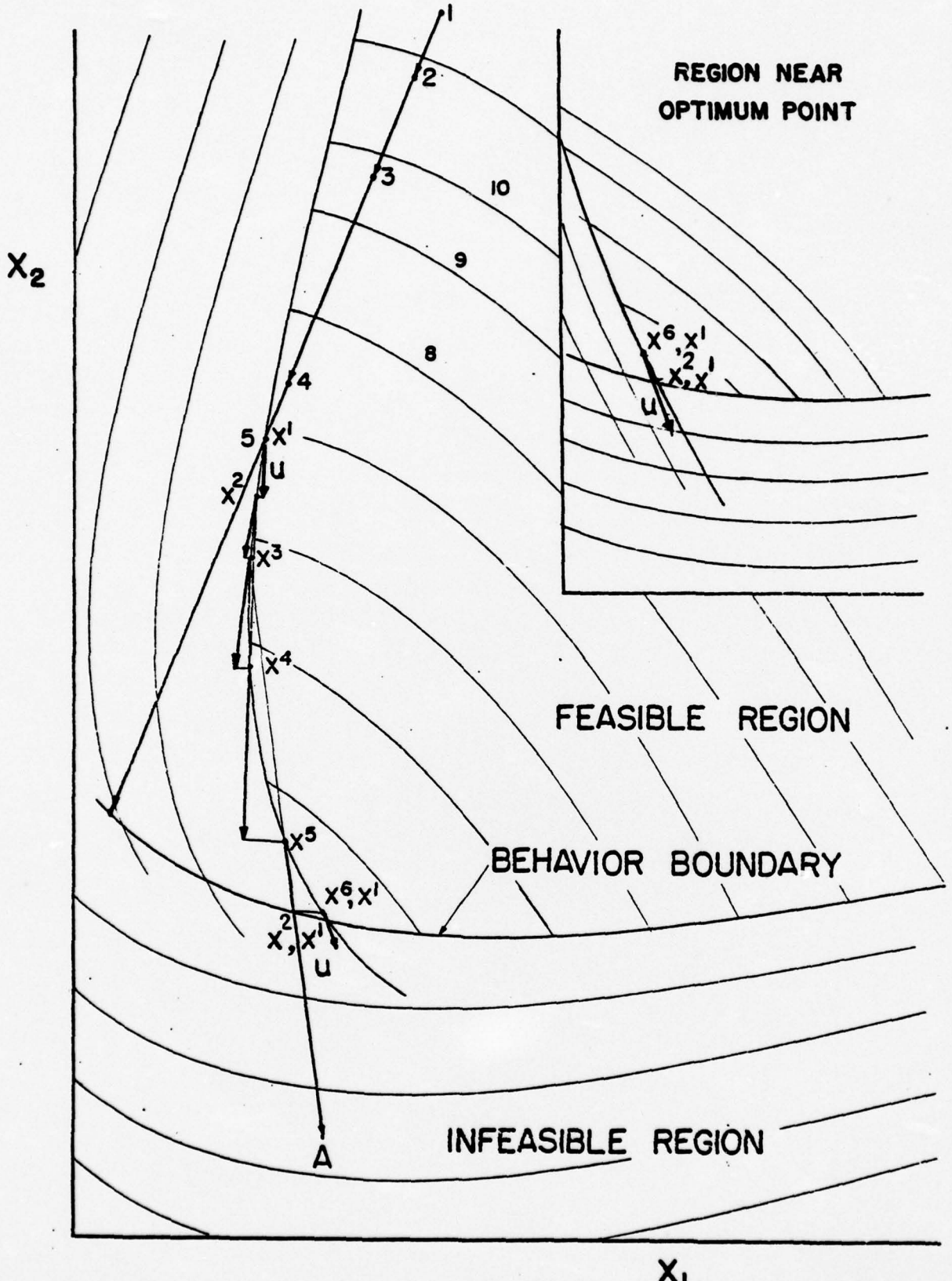


Figure 1. General Strategy of Boundary Tracking Method

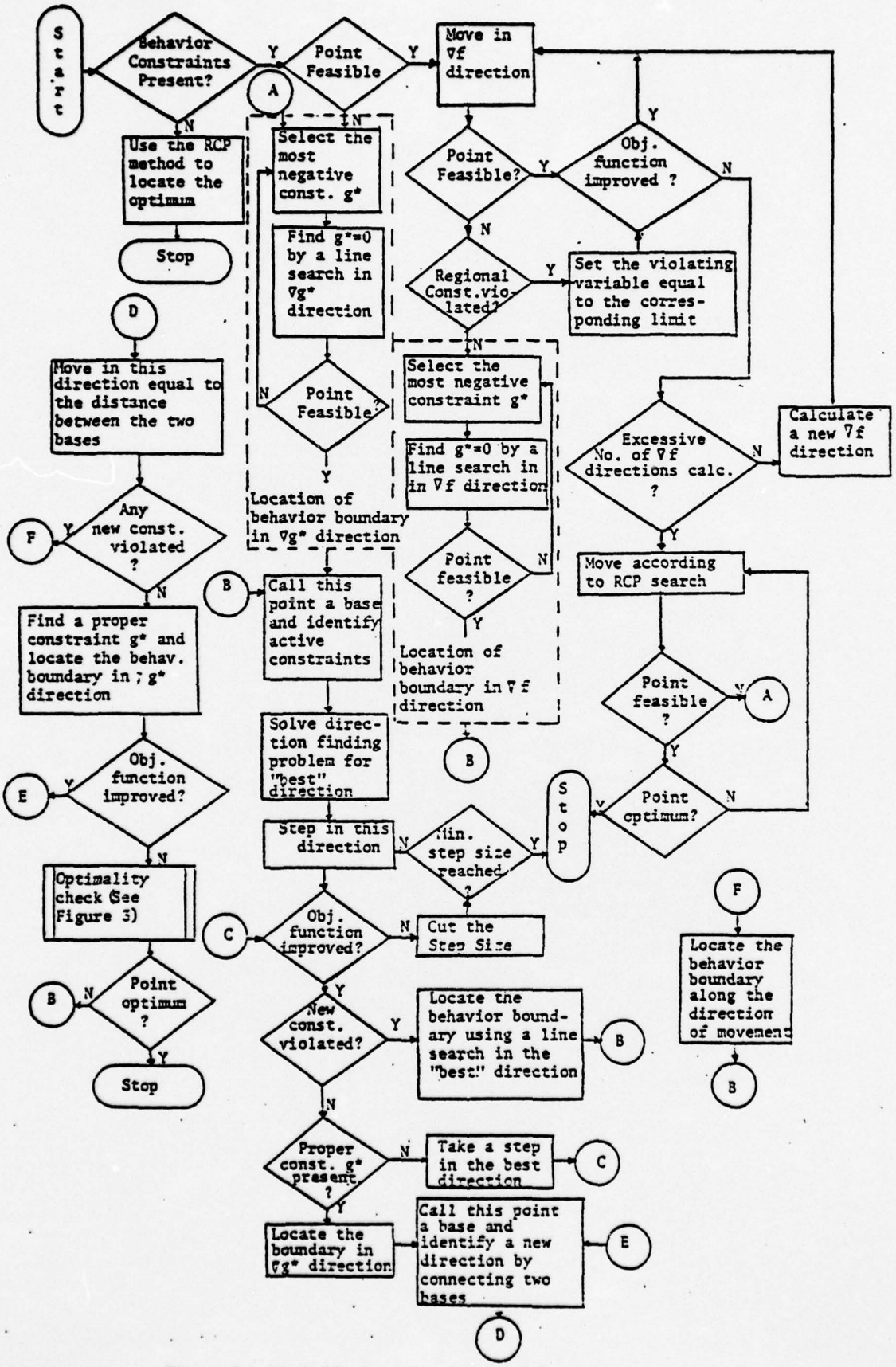


Figure 2 - Block Diagram of the General Strategy

continues, doubling the step size with each move.

Once the behavior boundary is crossed, a point on this boundary is located (point 5) using a line search in the direction of the last move.

The search is performed in the direction of movement, rather than say the direction of the gradient of the constraint function violated, since identification of the latter direction requires calculation of constraint function derivatives, while the former direction is available and thus requires no new information. The line search is used because of its simplicity.

If the search move crosses the IF boundary at a regional limit, it is possible to find the best direction to continue movement by solving Zoutendijk's direction finding problem (see section 3.2.3). Here for simplicity, however, the variables exceeding the limits are set equal to the corresponding limit and the search continued until a behavior constraint is violated. This strategy tends to deflect the direction of movement along the IF boundary.

If after a step, the point remains in the feasible region but the objective function has not improved, a new gradient direction is calculated at the last base point and the search restarted. If several direction changes fail to locate the constraint boundary, a more efficient, albeit more complex, unconstrained search method such as the modified Rotating Coordinate Pattern (RCP) search [17], is

used. This search procedure is applied here to avoid the zig-zagging problem associated with the gradient search method [12]. Thus the advantages of the gradient search method are exploited where possible, but the method is replaced by a more appropriate procedure whenever strategy dictates. If the behavior boundary is crossed in the modified RCP search, the nearby boundary may be located by any convenient method such as that already described.

The major contribution of this thesis is the procedure for movement along the behavior boundary. It is this movement which causes the difficulty associated with the solution of constrained nonlinear problems. The method used for locating the boundary, such as the gradient search or the RCP search, is of secondary importance. Since the bulk of the search is associated with movement along the boundary, the choice of the initial boundary locating procedure will not substantially effect overall performance. Thus, the procedures used for this purpose are not of great importance and need not be described in detail here. The interested reader is refered to references [12, 17] for a detailed description of these methods.

If the starting point is infeasible, the behavior boundary is located by first selecting the most negative constraint, that is, the constraint presenting the greatest violation. The point where this constraint vanishes is then located using a line search in the direction of the gradient of this constraint (see section 3.1.2). This direction is used here since it provides the shortest distance to the

surface where this constraint vanishes. The line search procedure used here is a modified Secant root finding procedure (see section 3.1.2).

All the constraints are then evaluated at the point where the selected constraint vanishes. If no violation exists the objective function is evaluated and the point is designated as a base point. However, if any other constraints are violated at this point, the smallest one (algebraically) is again selected and the point where it is zero is found. This process is continued until a point on the behavior boundary is located.

For problems without behavior constraints an unconstrained search method, such as the modified RC Pattern search [18], is used to locate the optimum point. This search method combines the well known RC Pattern search with Zoutendijk's feasible direction finding method (see section 3.2.3). The direction finding procedure is employed at the points of RC Pattern search failure.

2.3 Movement Along The Behavior Boundary

Once the point on the behavior boundary is located it is designated as a base point (point x^1 in Figure 1). The best direction to move is then determined by solving the direction finding problem of Zoutendijk (see section 3.2.3) and a step taken in this direction. The objective function is then evaluated at this point. If the function has improved, all the constraints are evaluated.

If none of the constraints that were inactive at the base are violated, an appropriate constraint is selected from among the active constraints (see section 3.1.1). The behavior boundary is located by finding a point where this constraint is zero by means of a line search in the direction of the gradient of the selected constraint (point x^2).

Upon locating the point on the boundary after the best direction move, the objective function is evaluated. If the function has improved the point is designated as a base point (point x^2). A move is then made from the last base in the direction of improvement along a line connecting two bases a distance equal to the distance between these bases.

The boundary is then located as before (see point x^3 of Figure 1) along the direction of the gradient of the appropriate constraint as evaluated at the last point where the best direction finding problem was formulated. It should be pointed out that this is an underlying feature of this algorithm, making multiple use of function evaluations in order to keep the number of new evaluations required as small as possible. Only if this direction fails to locate the boundary, are new gradient values computed.

When the point on the behavior boundary is found, it is compared to the last base point. If the function has improved, the point is designated as a new base point (points x^3, x^4, x^5) and the search is continued. However, if the function has not improved the direction

is abandoned and the search is re-started from the last base point by calculating a new best direction.

After each move all the constraints are evaluated, if a new constraint is violated (point A) the point where this constraint is zero is located and a new best direction calculated. However, if no new constraint is violated, an appropriate constraint is selected from among the previously active constraints (see section 3.1.1) and the boundary located, the process is continued until convergence is achieved.

If no appropriate constraint is active the move along the previous direction is continued, since it is the best direction available. If after a step in the above direction, the objective function has increased rather than decreased the step size is reduced. The process is repeated until a better point is found or the convergence criterion is met (see section 3.3).

If after any move a constraint inactive at the last base point is violated, the behavior boundary is located by finding the point where this constraint is zero. This point is found by a line search in the direction of the last move. This direction is used to eliminate the need to calculate the gradient of the new constraint. The search is then re-started by calculating the best movement direction at this point on the boundary.

2.4 Search termination description. For the few cases where the optimum occurs away from the behavior boundary, a sufficient condition for optimality is that all components of the gradient of the objective function be essentially equal to zero. For most cases, however, where the optimum is on the behavior boundary no such simple condition exists.

The search termination procedure used here is as follows. The search using a specified basic step size is performed until no further improvement can be obtained. When such a point is found it is designated as an optimality comparison base. Now the basic step size is reduced in an effort to achieve improvement and the search continued until a new optimality comparison base is obtained.

In order to justify a further reduction in the step size an optimality check is performed whenever the step size is to be reduced. If a convergence limit is satisfied, the step size is again reduced for further improvement and the search is continued until another optimality comparison base is obtained. A secondary convergence check is then performed in order to determine whether the latest improvement due to the step reduction is less than the previous improvement thereby indicating convergence has occurred. If both convergence tests are simultaneously satisfied, or the minimum step size is reached, the search is terminated. Otherwise, the search is re-started by calculating a new best movement direction.

A block diagram of the search termination procedure is given in Figure 3.

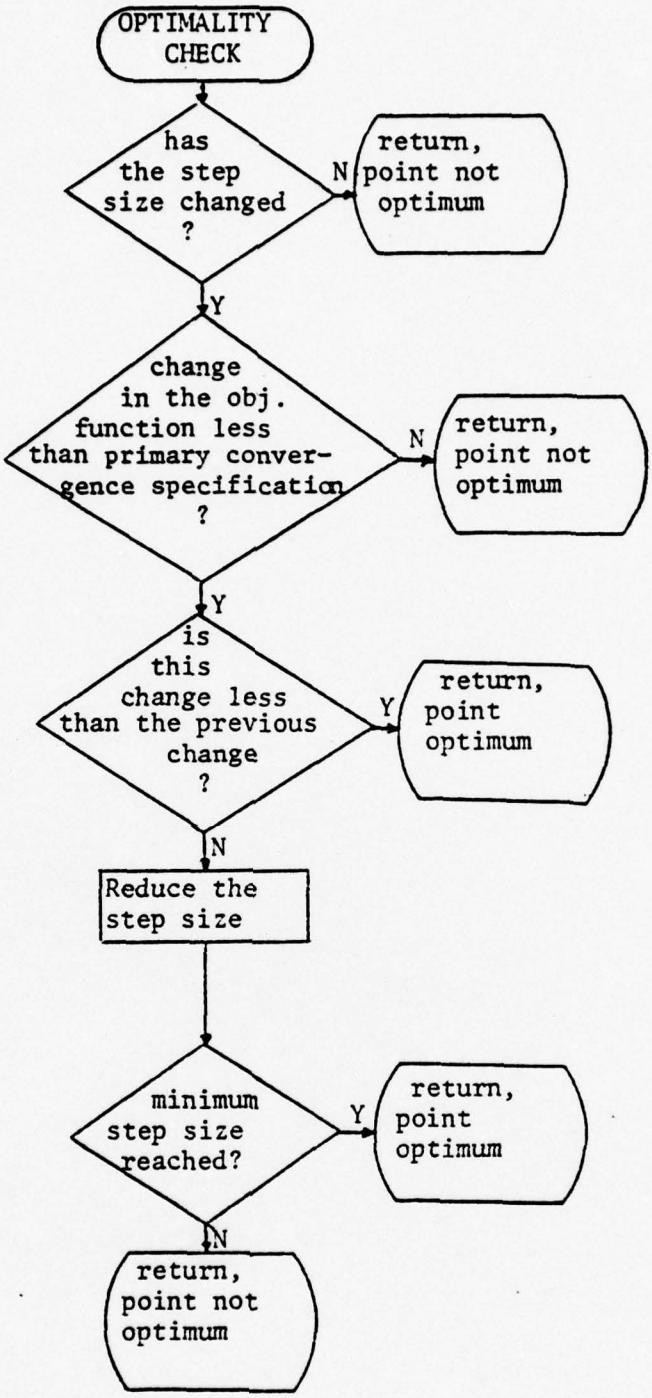


Figure 3 - Block diagram of the termination procedure

CHAPTER 3

MATHEMATICAL DEVELOPMENT

3.1 Boundary Location

It is assumed for the purpose of devising a search strategy that the solution to the optimization problem with behavior constraints lies on the behavior boundary. The main strategy is first to locate such a boundary and then to move along this boundary. For the purpose of locating the boundary a "proper" behavior constraint is identified and method for locating the behavior boundary selected or developed.

3.1.1 Identification of "proper" constraints. All negative and small positive constraints are designated as active constraints (for definition of activity see section 3.2.3). A constraint is considered "proper" if it is necessary or desirable to locate a point where the value of this constraint is zero after making a search move in an effort to establish a new base.

Since it is necessary to eliminate any constraint violation present at a point, all the negative constraints are considered "proper".

If all active constraints are positive, a positive active constraint is considered proper if

$$-\nabla f \cdot \nabla g_j < 0 \qquad (3.1)$$

This indicates that the projection of $-\nabla f$ on ∇g is of opposite sign to ∇g (point 1 Figure 4) and therefore, the value of the constraint

g_j will decrease as a result of a move in $-\nabla f$ direction (minimization problem).

The best direction for improving a function is its gradient direction. However, this direction is not always feasible in constrained problems (point 1 Figure 4).

A direction is desired such that after taking a step, the objective function will improve the maximum amount possible without violating the behavior constraints. A direction along the behavior boundary often has this property. Moving along the behavior boundary, however, may not always produce the best improvement in the objective function as shown by point 2 in Figure 4. Here projection of $-\nabla f$ on ∇g_j has the same sign as ∇g_j. Thus, the value of the constraint g_j will increase by moving in $-\nabla f$ direction. A criteria is therefore needed that can indicate which positive constraint (if any) is "proper". Equation (3.1) satisfies this purpose, since if satisfied it identifies the constraints that may be violated if a move in the best direction ($-\nabla f$) is made. Thus it identifies proper constraints, that is, those constraints along which it is desirable to move in order to achieve the best improvement in the objective function.

If no proper constraint can be identified, the movement along the previous direction is continued, since there is no advantage in locating and moving along the nearby boundary.

3.1.2 The boundary locating method. Call the set of proper constraints P_k where, $k = 1, 2, \ldots K$. If more than one "proper"

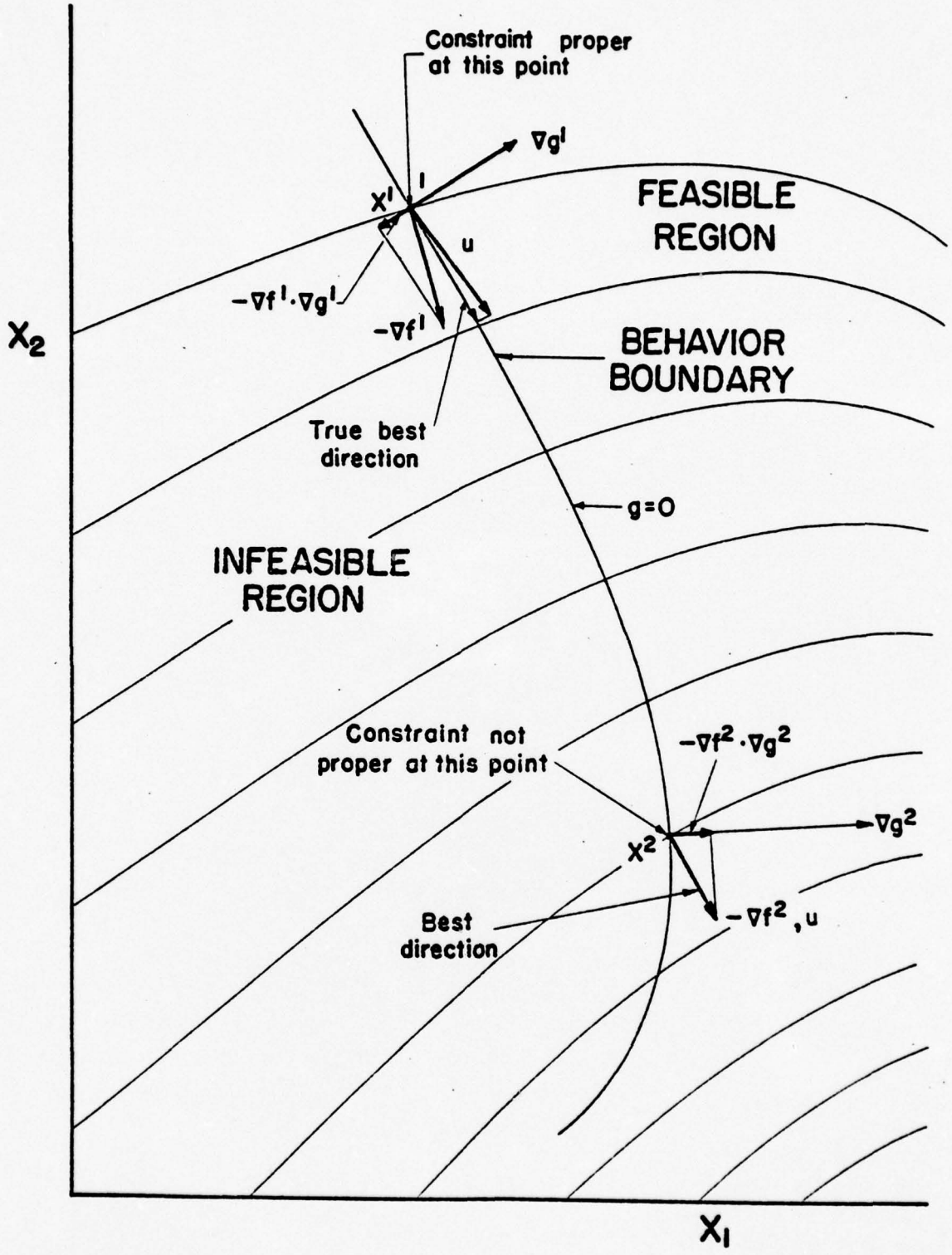


Figure 4 - Selction of a Proper Constraint

constraint is identified, that is if

$$K > 1 \tag{3.2}$$

the smallest (algebraically) is selected and called g*. The root of g* is then found in the direction of movement or the gradient of g*.

The root, if it exists, of the constraint g*(x) along a line may be found as closely as desired by means of the "secant root-finding" method [19] by iterating the equation

$$x^{s+1} = x^s - g^*(x^s)(x^s - x^{s-1})/[g^*(x^s) - g^*(x^{s-1})] \quad s = 1,2,\ldots N_1 \tag{3.3}$$

where N_1 is the maximum number of iterations permitted, until

$$|g^*(x^{s+1})| \leq e_1 \tag{3.4}$$

is satisfied. Here e_1 is an arbitrarily selected accuracy limit. Initially x^s, x^{s-1} are two points on the line defined by

Case A: *If the boundary is to be located in the direction of the last* search move (see section 2.3) let the initial points on the line (s = 1) be given by

$$x^{s-1} = x^b \tag{3.5}$$

$$x^s = x^r \tag{3.6}$$

where x^b and x^r are the end points of the last search step.

Case B: *If the boundary is to be located in the gradient direction* of the constraint g*(x) (see section 2.3) let

$$x^{s-1} = x^r \tag{3.7}$$

$$x^s = x^r \pm \alpha^n \nabla g^*(x^n) / (\nabla g^*(x^n)) \tag{3.8}$$

where x^r is the point from which the boundary is to be located and x^n is the last base where a direction finding problem was formulated

(see section 3.2.3.). Here the plus and minus signs are used when g* is negative or positive respectively. The quantity α^n is the step size used at point x^n.

Case C: For the case where this procedure is used to find the behavior boundary where a starting point of the optimal search is infeasible, replace x^r by the starting point x^0 and let,

$$\alpha^n = \alpha^0 \tag{3.9}$$

where α^0 is an arbitrarily specified initial step size, and let

$$\nabla g^* (x^n) = \nabla g^* (x^0) \tag{3.10}$$

in equations (3.7) and (3.8).

After equation (3.4) is satisfied, the remainder of constraints in the set P_K may also vanish. However, if after eliminating the the smallest violation

$$g_j (x^{s+1}) \geq - e_1 \qquad j = 1,2,\ldots J \tag{3.11}$$

is not satisfied, a new g* is selected and the procedure repeated until a point on the behavior boundary is located (equation 3.11 satisfied).

In case B, slow convergence may indicate that the gradient $\nabla g^*(x^n)$ may no longer be applicable at point x^n. This situation is shown in Figure (5). There is no point along the gradient $\nabla g^*(x^1)$ which falls on the boundary. A new gradient direction $\nabla g^*(x^r)$ is therefore calculated and used in place of $\nabla g^*(x^n)$. If after calculating a new gradient direction the boundary still cannot be located with a reasonable number of iterations, the difficulty may be due to too large a reach step (line $x^6 - x^r$ in Figure 5). Thus where recalculation of the

gradient fails, the step size is reduced and another effort is made to locate the boundary from a point somewhat closer to the last base.

Therefore, unless

$$s < N_1 \tag{3.12}$$

where N_1 is chosen empirically, let

$$\nabla g^* (x^n) = \nabla g^* (x^r) \tag{3.13}$$

in equation (3.8) and repeat the application of equation (3.3) If equation (3.12) is again violated let

$$\Delta x^{r+1} = \Delta x^r/2 \tag{3.14}$$

where

$$\Delta x^r = x^r - x^b \tag{3.15}$$

and x^b is the last base point. Then select the smallest proper constraint at point x^{r+1} where

$$x^{r+1} = x^b + \Delta x^{r+1} \tag{3.16}$$

(point $x^6 + \Delta x^{r+1}$ in Figure 5), replace x^r by x^{r+1} in equations (3.7) and (3.8) and repeat the iteration of equation (3.3). The process is continued until the boundary is located without violating equation (3.12) or until

$$|\Delta x^{r+1}| < \alpha^n. \tag{3.17}$$

If equation (3.17) is satisfied the attempt to locate the boundary is abandoned. The optimal search is then restarted by calculating a new starting direction (see section 3.2.3) at the last feasible point encountered.

3.2 Mathematical Representation of General Strategy

The general strategy of the algorithm can be represented mathematically as follows.

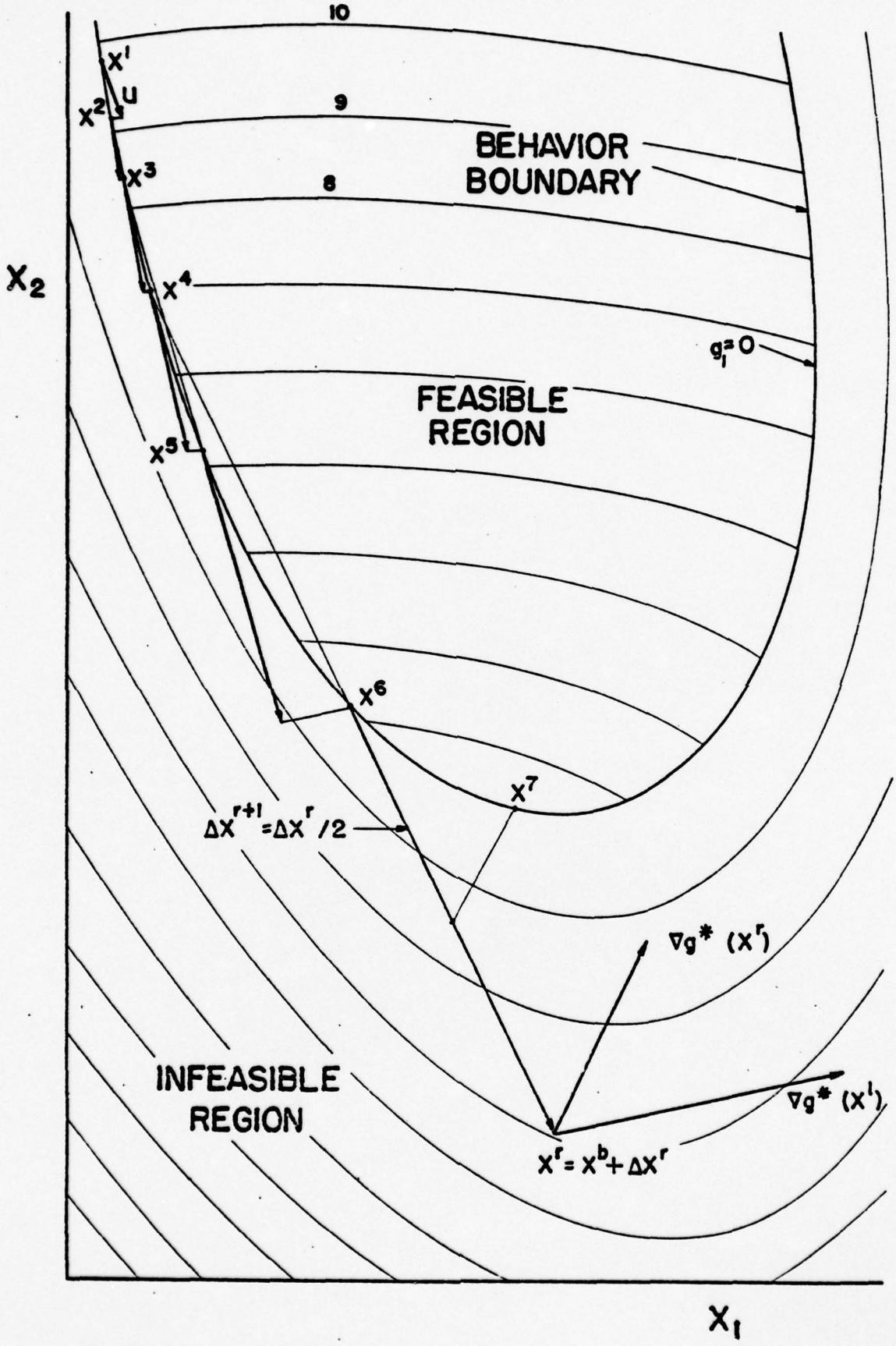


Figure 5 - The Boundary Locating Method

3.2.1 Movement to the boundary-starting point in the feasible region. A block diagram for the movement to the boundary procedure is given in Figure 6.

Given the starting point x_i^r, where initially r = 0, which satisfies regional constraints

$$x_i^L \leq x_i^r \leq x_i^U \qquad i = 1, 2, \ldots I \qquad (3.18)$$

and all

$$g_j(x_i^r) \geq 0 \qquad j = 1, 2, \ldots J \qquad (3.19)$$

the objective function $f(x^r)$ is evaluated. This point is called a base point x^b, where initially b = n = 1, and the first comparison base x^n is defined as,

$$x^n = x^b \qquad (3.20)$$

Then with

$$\alpha^r = \alpha^n \qquad (3.21)$$

let

$$x^{r+1} = x^r - \alpha^r \left[\nabla f(x^n)\right] / \left|\nabla f(x^n)\right| \qquad (3.22)$$

where ∇ is the gradient operator, $|\nabla f|$ the magnitude of ∇f and, α^r is the step size at iteration r.

If any

$$x_i^{r+1} > x_i^U$$

let

$$x_i^{r+1} = x_i^U \qquad (3.23)$$

or if any

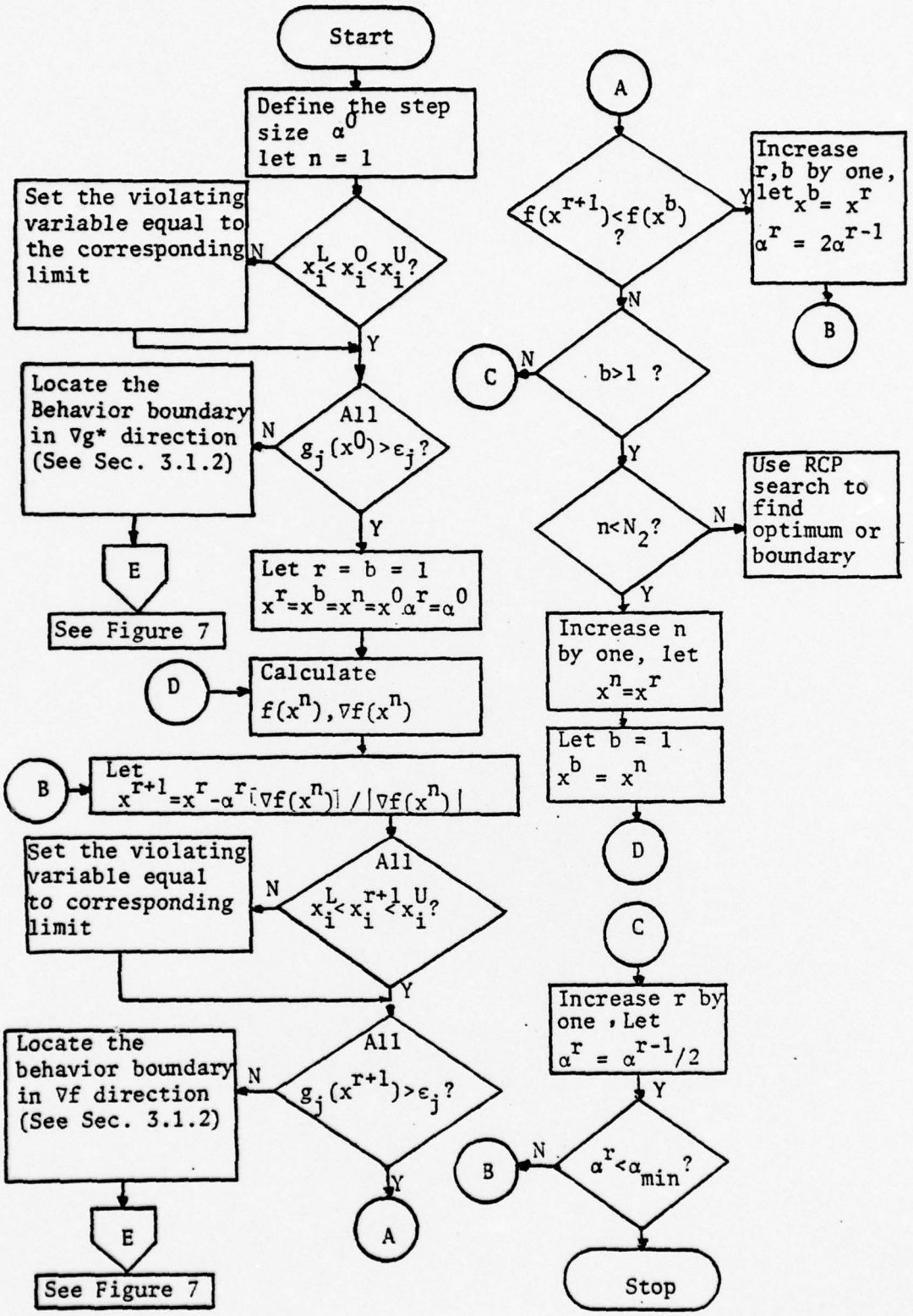


Figure 6 - Flow Chart of the Procedure for Movement to the Boundary

let

$$\left.\begin{aligned} x_i^{r+1} &< x_i^L \\ x_i^{r+1} &= x_i^L \end{aligned}\right\} \tag{3.24}$$

Now compute the $g_j(x^{r+1})$, if equations (3.19) are satisfied, evaluate $f(x^{r+1})$. If

$$f(x^{r+1}) < f(x^b) \tag{3.25}$$

call the point a new base point. Thus let

$$x^{b+1} = x^{r+1} \tag{3.26}$$

A new move is then made according to equation (3.22) with the step size redefined as

$$\alpha^{r+1} = 2\alpha^r \tag{3.27}$$

and application of equations (3.23-3.26) repeated. If b>1 and equation (3.25) is not satisfied increase n by one, let the next comparison base

$$x^n = x^b \tag{3.28}$$

and let

$$x^r = x^b \tag{3.29}$$

Now if

$$n < N_2 \tag{3.30}$$

where N_2 is an arbitrary constant signifying the number of direction changes, a new gradient direction ∇f is calculated. The application of equations (3.20-3.27) are then repeated until either a point in the infeasible region is found or equation (3.30) is not satisfied. If point x^{r+1} is in the infeasible region the behavior boundary is located in the direction of the last move (see section 3.1.2).

If n exceeds N_2 the gradient search is abandoned and the RCP search invoked. This search method is continued in the unconstrained

region until a point in the infeasible region is found or the optimum is achieved (see section 3.3).

If equation (3.25) is not satisfied when b=1, the step size α is redifined as

$$\alpha^{r+1} = \alpha^{r}/2 \tag{3.31}$$

and a new move is made according to equation (3.22). The application of equations (3.31) and (3.22) is repeated until equation (3.25) is satisfied or

$$\alpha^{r+1} \leqslant \alpha_{min} \tag{3.32}$$

where α_{min} is a minimum step size. In the latter case the point is assumed to be optimum and the search is terminated.

3.2.2 Movement to the boundary - starting point in the infeasible region. If the starting point x_i^o is in the infeasible region or the RCP search has located a point in this region the behavior boundary is located by a search in a constraint gradient direction (see section 3.1.2).

3.2.3 Initiation of movement along the boundary. After locating a point on the behavior boundary a method is needed which can initiate the move along this boundary toward the optimum if optimality has not yet been achieved. The Feasible direction finding algorithm of Zoutendijk [20] is suitable for this purpose since, it either provides a direction along which an improved point can be found, or indicates the presence of a local optimum.

Definition: a) A direction vector u is called "feasible" if after taking a sufficiently small step along this direction no constraint is violated [20]. This will be true if

$$(u)^T \nabla g_j (x) \geq 0 \tag{3.33}$$

where $(u)^T$ is the transpose of u, since a small step in this direction will produce no change or an increase in $g_j(x)$.

b) A feasible vector u is called "usable" if a move in the direction u can also provide an improvement in f(x). This will be the case if

$$(u)^T \nabla f(x) < 0 \tag{3.34}$$

since a sufficiently small step in the u direction will produce a decrease in the value of f(x).

c) A behavior constraint $g_j(x)$ is considered "active" if

$$g_j(x_i) \leq \varepsilon_j \tag{3.35}$$

where

$$\varepsilon_j = K_j (\alpha^n / \alpha^0) \tag{3.36}$$

is an array of positive, arbitrary small numbers. These numbers may approach zero during the search, but can not be identically zero [20]. The quantity α^0 is the step size at the beginning of the search and K_j is a small positive constant.

d) A lower regional constraint is considered "active" if

$$x_i - x_i^L \leq \alpha^n \tag{3.37}$$

and the upper constraints active if

$$x_i^U - x_i \leq \alpha^n \tag{3.38}$$

Denote the set of all active behavior constraints by A_j. Call R_i^- and R_i^+ the active lower and upper regional constraint set respectively.

The direction finding problem can be formulated in the following manner:

Given x, find u that results in the

$$\max \sigma \tag{3.39}$$

and for which

$$\sigma > 0 \tag{3.40}$$

$$(u)^T \nabla f(x) + \sigma \leq 0 \tag{3.41}$$

$$-(u)^T \nabla g_j(x) + W_j \sigma \leq 0 \qquad J \in A_j \tag{3.42}$$

$$u_i \leq 0 \qquad i \in R_i^- \tag{3.43}$$

$$u_i \geq 0 \qquad i \in R_i^+ \tag{3.44}$$

$$|u_i| \leq 1 \qquad i = 1, 2, \ldots I \tag{3.45}$$

Equations (3.45) bound the length of u to prevent the direction finding problem from producing an unbounded solution vector.

When suitable deflection parameters W_j are used, the solution of the problem provides a usable and feasible direction u since, if $\sigma > 0$ from equations (3.41) and (3.42) equations (3.33) and (3.34) will be satisfied. It also provides the best usable direction since, if σ is maximized, the left hand side of equation (3.34) will be a maximum and therefore the direction found will be the best direction for decreasing f(x) [20].

It may be seen that the direction finding problem of equations (3.39 - 3.45) is a linear programming problem. Thus, any linear programming method such as the simplex procedure [21] can be used to provide the solution.

Since the objective here is to move along the IF boundary, the direction u should be as close to this boundary as possible. The work associated with locating the boundary will therefore be minimized after the move. The deflection parameters W_j here are thus set to be equal to zero. This produces a direction u that tends to be tangent to the boundary.

3.2.4 Movement along the boundary. A flow chart for the movement along the boundary procedure is given in Figure 7.

Call the point on the behavior boundary x^b and define

$$\alpha^{\ell} = \alpha^n \tag{3.46}$$

where initially $b = \ell = 1$, x^b is a base point, and α^{ℓ} is a comparison step size. A direction u is then calculated (see section 3.2.3) and the point x^r defined as

$$x_i^r = x_i^b + \Delta x_i^r \tag{3.47}$$

where

$$\Delta x_i^r = \alpha^{\ell} u_i / |u| \tag{3.48}$$

if

$$f(x^r) < f(x^b) \tag{3.49}$$

and

$$g_j(x^r) > \varepsilon_j \qquad \begin{matrix} j = 1, 2, \ldots J \\ j \notin A_j \end{matrix} \tag{3.50}$$

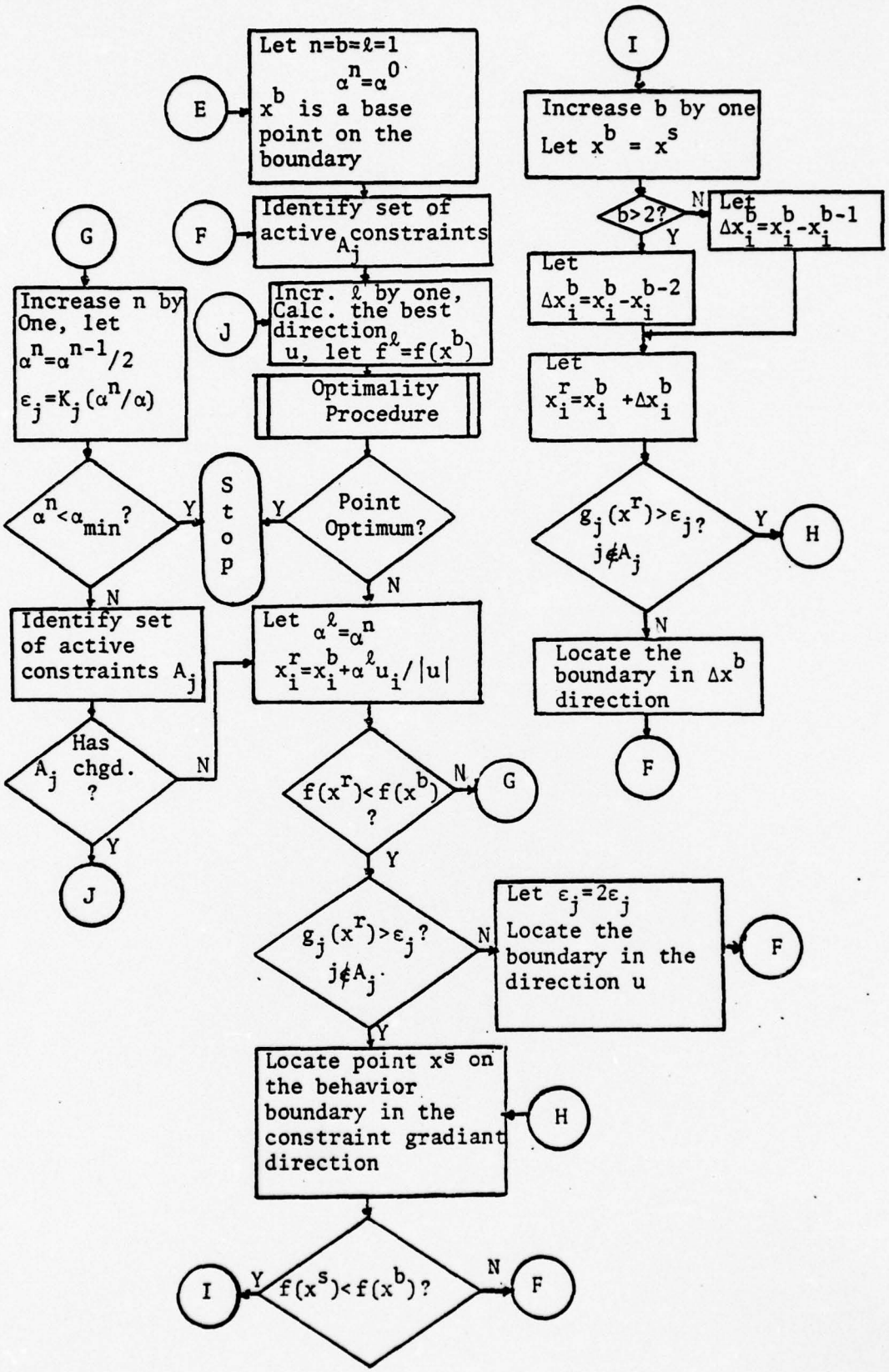


Figure 7 - Flow Chart of the Procedure for Movement along the Boundary

are satisfied, point x^s on the behavior boundary is located by a search in the constraint gradient direction and the objective function evaluated. Now, if equation (3.49) is satisfied at x^s, let

$$x^{b+1} = x^s \tag{3.51}$$

now index b and define the next step as

$$x_i^r = x_i^b + \Delta x_i^r \tag{3.52}$$

where

$$\Delta x_i^r = x_i^b - x_i^{b-2} \qquad b > 2 \tag{3.53}$$

or

$$\Delta x_i^r = x_i^b - x_i^{b-1} \qquad b = 2 \tag{3.54}$$

The constraints are now evaluated. If equation (3.50) is satisfied, the behavior boundary is located in the constraint gradient direction and the application of equations (3.51 - 3.54) repeated.

If when b=1, equation (3.50) is not satisfied, that is, if new constraints have become active, the value activity limit ε_j is doubled for all constraints in set A_j not satisfying equation (3.50), since its previously assigned value was not sufficient to properly define activity. The boundary is then located in the direction u and the application of equations (3.47 - 3.50) repeated.

If equation (3.50) is not satisfied when b > 2 the boundary is located in the direction of the last move Δx^r. Then a new set of active constraints is defined and the search re-started from the last base point (see section 3.2.3).

In application of equations (3.49 - 3.54) if equation (3.50) is satisfied, but equation (3.49) is not satisfied, this indicates that the last search move direction was unproductive, therefore the last direction of movement is abandoned and the search is re-started from the last base point x^b by calculating a new direction u, and setting b=1.

After employing equations (3.47) and (3.48), if equation (3.49) is not satisfied, this indicates that the step size is too large, therefore let

$$\alpha^{n+1} = \alpha^n/2 \tag{3.55}$$

$$\alpha^{\ell+1} = \alpha^{n+1} \tag{3.56}$$

Now, since new activity limits are defined [see equation (3.36)] the active constraints are again determined, if the set A_j has changed a new direction u is calculated, otherwise the previous direction is used and application of equations (3.47 - 3.49) repeated using the new step size. This procedure is continued until equation (3.49) is satisfied or convergence is achieved.

Application of equations (3.47 - 3.56) is invoked when applicable, as explained above, until the convergence or optimality criterion are met (see section 3.3).

3.3 Search Termination

A flow chart for the search termination procedure is given in Figure 8.

The question of optimality is considered at all points where the direction finding problem is formulated. For the few cases where a local optimum occurs away from the behavior boundary that is where all

$$g_j(x) > \varepsilon_j \tag{3.57}$$

a sufficient condition for optimality is

$$|\nabla f| \leq e_2 \tag{3.58}$$

where e_2 is an arbitrary small number.

For most cases, however, where optimum is constrained, the solution to Zoutendijk's direction finding problem [20] provides a test for optimality. A null solution vector u indicates a local optimum where all $\varepsilon_j = 0$. However, where some $\varepsilon_j \neq 0$ and the active constraints are also not zero, the point may be merely near, rather than at the optimum, since in this type of problems the optimum point is usually on the behavior boundary. Therefore, when $u = 0$ attempts are made to re-start the search and the optimality procedure is invoked only when a non-zero direction vector u is obtained. In order to re-start the search the step size is redefined as

$$\alpha^{n+1} = \alpha^n/2 \tag{3.59}$$

and therefore the activity limits ε_j are redefined according to equation (3.36). A new direction u is then calculated if A_j has changed. This procedure is repeated until a non-zero direction is found or the minimum step size is encountered, that is

$$\alpha^n \leq \alpha_{min} \tag{3.60}$$

in which case the search is terminated and the point is assumed to be

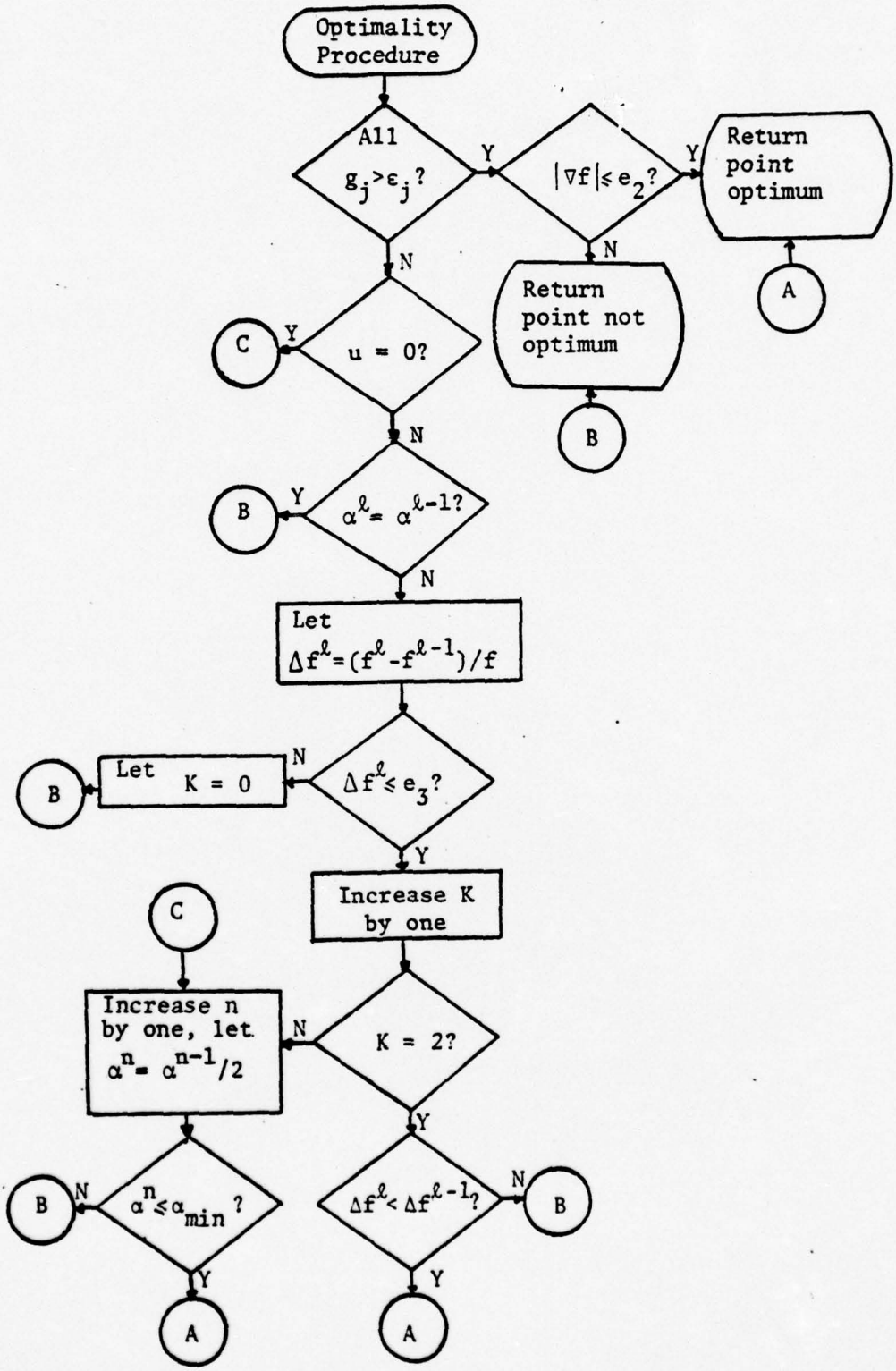


Figure 8 - Flow Chart of the Termination Procedure

an optimum.

For the cases where $u \neq 0$, increase ℓ by one and let

$$\alpha^{\ell} = \alpha^{n} \tag{3.61}$$

$$f^{\ell} = f(x^{b}) \tag{3.62}$$

where f^{ℓ} is a comparison value with initially $\ell = 0$, and $f^{0} = C_1$, where C_1 is an arbitrary large number. Now if

$$\alpha^{\ell} = \alpha^{\ell-1} \tag{3.63}$$

no optimality check is made since ℓ has increased because the search was re-started due to the fact that the previous movement direction was unproductive. However, if the step size has changed, this indicates that the maximum improvement in the function has been achieved using the previous step size and the step size must be reduced for additional improvement. A convergence check is now made in order to determine if a search using the new step size is justified. Thus, where equation (3.63) is not satisfied, define

$$\Delta f^{\ell} = |(f^{\ell} - f^{\ell-1}) / f^{\ell}|. \tag{3.64}$$

Unless

$$\Delta f^{\ell} < e_3 \tag{3.65}$$

where e_3 is the primary convergence criteria, the search is continued.

A secondary convergence check is initiated whenever the primary convergence criteria is met, [equation (3.65) satisfied] in order to confirm optimality. For this purpose the step size α is reduced according to equation (3.59) and the search is re-started. When further movement with the new step size terminates the primary convergence check is invoked. Therefore, whenever equation (3.65) is not

satisfied at point x^b, let

$$K = 0 \tag{3.66}$$

and where equation (3.65) is satisfied index K and let

$$\alpha^{n+1} = \alpha^n/2 \tag{3.67}$$

$$\alpha^{\ell+1} = \alpha^{n+1} \tag{3.68}$$

If equation (3.60) is satisfied, the point is assumed to be a local optimum and the search is terminated. Otherwise a new direction u is computed and the search continued. When the primary convergence check is satisfied in two consecutive tries, that is if

$$K = 2 \tag{3.69}$$

the secondary convergence check is invoked. Thus if

$$\Delta f^{\ell} < \Delta f^{\ell-1} \tag{3.70}$$

the search is terminated. Otherwise a new direction u is computed, K is set equal to 1 and the search continued. When equation (3.70) is satisfied, the change in the value of the objective function (Δf) has decreased from the previous convergence check, even though a reduced step size was used, the point is thus assumed to be optimum and the search is terminated.

CHAPTER 4

DESIGN EXAMPLE

4.1 Problem Statement

Consider the minimum weight design of submersible, circular, cylindrical shells reinforced by equally spaced "T" type frames. The design variables with respect to which the optimization is to be carried out are: Plate thickness, frame web and frame flange thickness, frame flange width, web height, and frame spacing. The objective function to be minimized is the ratio of shell weight to the weight of fluid displaced. All the standard design equations used in submersible shell design practice are to be satisfied. All variables will be treated as continuous. The fixed design parameters are: The operating depth, shell diameter, shell segment length, shell eccentricity, construction material properties, factors of safety to be used in design, maximum and minimum values permitted for the design variables, and, when required, a maximum (when external frames are used) or minimum (when internal frames are used) frame diameter.

This problem is selected here since it is a difficult and computationally demanding engineering problem which the relatively reliable optimization code, DSDA and the popular SUMT procedure, could not solve [17].

In a previous study [22] it has been shown that the merit surface in this problem is fairly flat, therefore a wide range of variables

with similar objective function values may be generated during the search. This study also demonstrates that movement along the behavior boundary is very difficult in this problem.

A modified version of this problem with only four variables was treated in Reference [22], using the DSDA procedure. The formulation of the more difficult six variable problem presented here is similar to that used in Reference [22].

Only CADOP3 [23], a modified version of CADOP2 code, [16] reached an optimal solution to this problem. This code is three times faster than CADOP2 code in problems with behavior constraints. However, due to the difficulties involved in moving along the behavior boundary, the execution time required by CADOP3 to achieve the optimum is quite long (see section 4.3). The problem, therefore, was treated using the BT algorithm in order to demonstrate its superior ability in moving along a difficult behavior boundary.

4.2 Problem Formulation

4.2.1 Objective function. For cylindrical vessels with periodic "T" type reinforcing frames the objective function may be written as

$$f(x) = \begin{cases} W/\gamma_w V_D & \text{internal frames} \\ W/[\gamma_w(V_D + V_w + V_f)] & \text{external frames} \end{cases} \qquad (4.1)$$

where W is the weight of the hull segment, given by

$$W = \gamma_s (V_s + V_w + V_f) \qquad (4.2)$$

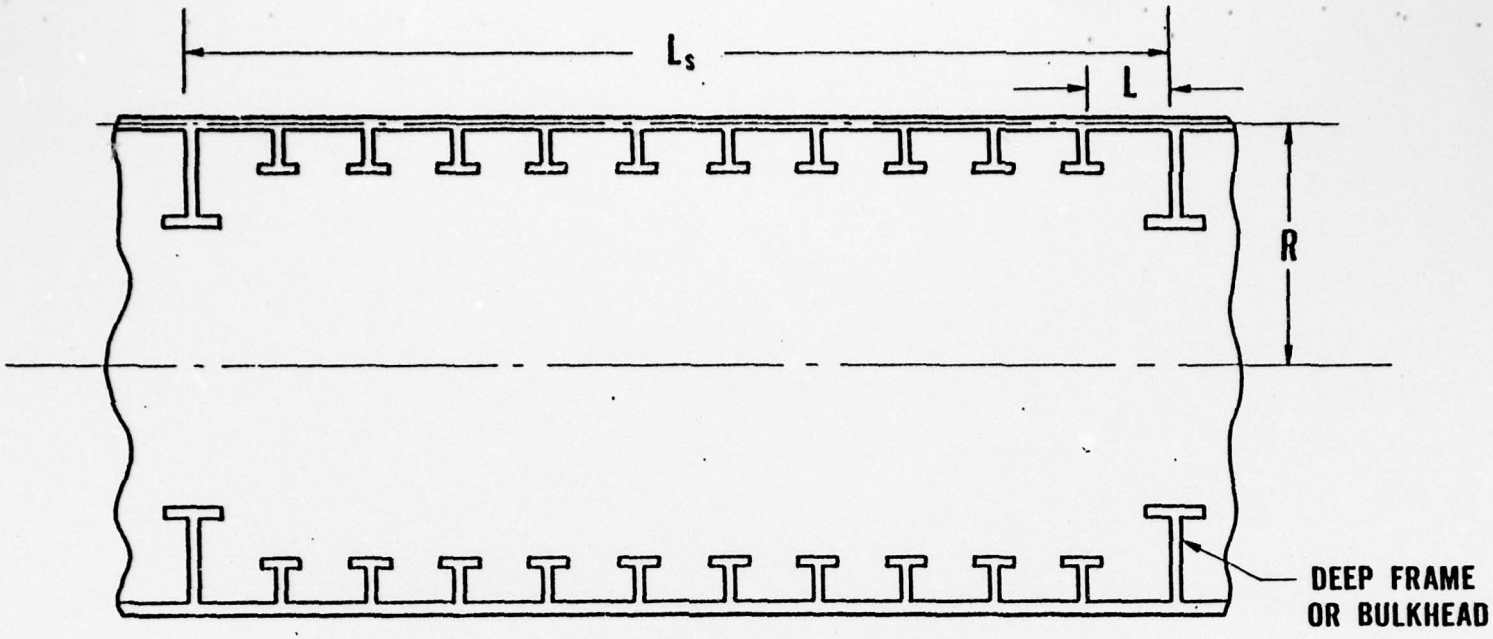
The γ_s and γ_w are the specific weight of the material and immersion fluid, respectively. V_s, V_w, V_f are the volume of the plating, frame webs, and frame flanges, respectively, and V_D is the displacement volume of the cylinder enclosed by the plating envelope. The variables x_1, x_2,...,x_6, which in turn represent the quantities t, b, b', w, L', and h, respectively, are defined in Figure 9. The objective function f(x), therefore represents the ratio of the shell weight to the weight of the fluid displaced for the shell segment of length L_s. The weight of the bulkheads is omitted.

4.2.2 Constraint equations. The constraint equations $g_j(x)$ are divided into two groups: a) Behavior constraints which control the failure modes or impose limitations on the space relationships among the variables, and b) regional constraints which specify ranges of the variables. The basic behavior constraint equations are formulated using a modified version of the design equations given in References [24] ans [25].

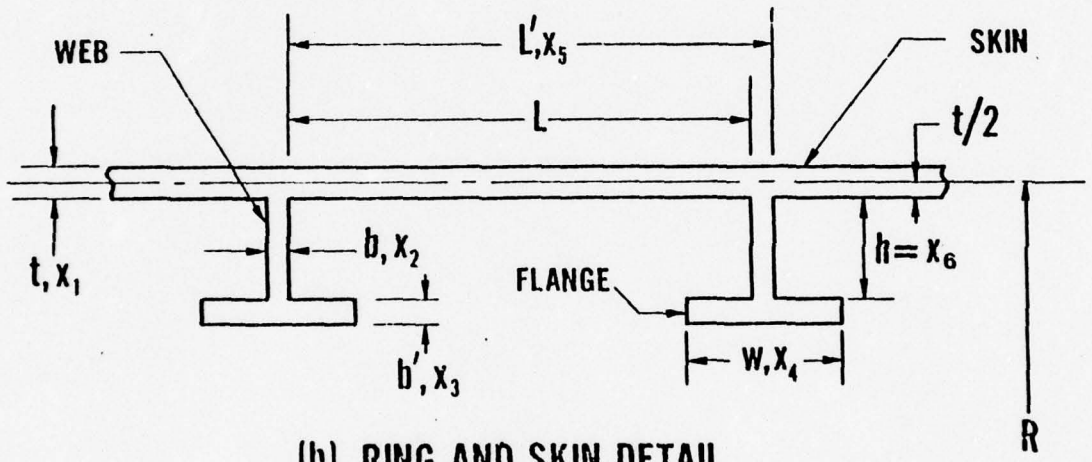
The general instability constraint is given by

$$g_1 = (p^*_{cg} - S_1 p) / p^*_{cg} \geq 0 \qquad (4.3)$$

where p is the applied hydrostatic pressure; p^*_{cg} is the minimum of the collapse pressure $p_{cg}(n,m)$ due to general instability [26], and S_1 is the factor of safety for this failure mode. The collapse pressure is given by



(a) SHELL SEGMENT CROSS-SECTION



(b) RING AND SKIN DETAIL

Figure 9 - Typical Shell Cross Section

$$p_{cg}(n,m) = [Dm^2(1 + \beta^2)^2 + \frac{\beta^2 m^2}{L'}(GJ_f + EI_f\beta^2) + \frac{12DZ^2}{\pi^4 m^2}\frac{(1 + F\Lambda_f)}{\Lambda}]\frac{\pi^2}{L_s^2 R(1/2 + \beta^2)} \tag{4.4}$$

where $p_{cg}(n,m)$ is the buckling pressure for a specified n and m and

$$\Lambda_f = 1 + 2n^2\bar{d}(1 - \beta^2\mu) + n^4\bar{d}^2(1 + \beta^2)^2$$

$$\Lambda = (1 + \beta^2)^2 + 2\beta^2 F(1 + \mu) + \beta^4 F(1 - \mu^2) \tag{4.5}$$

$$Z^2 = L_s(1 - \mu^2) / R^2t^2$$

$$\beta = nL_s / m\pi R$$

$$F = (b'w + bh) / tL' \tag{4.6}$$

$$D = Et^3 / 12(1 - \mu^2)$$

$$\bar{d} = d / R$$

where d is the (algebraic) distance to the neutral axis of the frame from the midplane of the hull plating, taken as positive when it is outward from the central axis of the hull [25]. The quantities GJ_f and EI_f are the torsional and bending stiffness of the frame, respectively. Symbols representing hull and frame dimensions are defined in Figure 9. The quantities G, E, and μ are the shear modulus $G = E / 2(1 + \mu)$, tensile modulus, and poisson's ratio, respectively.

Buckling of the shell between frames is controlled by the constraint

$$g_2 = (p^*_{cs} - S_2 p) / p^*_{cs} \geq 0 \tag{4.7}$$

where

$$p_{cs}(n,m) = (\pi^2/L'^2R)\,[Dm^2(1+\beta^2)^2 + 1/\Lambda(12DZ^2/\pi^2m^2)]1/(1/2+\beta^2) \quad (4.8)$$

For a given set of design parameters and pressures the buckling pressure functions $p_{cg}(n,m)$ and $p_{cs}(n,m)$ depend on the two discrete independent variables, the axial and circumferential wave number m and n respectively. To obtain the critical buckling pressure one must find the minimum of $p_{cg}(n,m)$ and $p_{cs}(n,m)$ with respect to n = 0, 1,2., and m = 1, 2,.... For this purpose the procedure of Reference [26] is used here.

The necessity to avoid plating yield produces the constraint

$$g_3 = (\sigma_{pa} - \sigma_p) / \sigma_{pa} \geq 0 \quad (4.9)$$

where σ_{pa} is the allowable plating stress and

$$\sigma_p = (\sigma_r^2 + \sigma_\phi^2 - \sigma_r\sigma_\phi)^{1/2} \quad (4.10)$$

The quantities σ_r and σ_ϕ are found from

$$\sigma_\phi = -pR/_t[1 + \Gamma(H_M + \mu H_E)] \quad (4.11)$$

$$\sigma_r = -pR/_{2t}[1 + 2\Gamma H_E] \quad (4.12)$$

The terms -pR/t and -pR/2t are the hoop and axial stresses, respectively; Γ is a frame deflection parameter, and H_M and H_E are the functions that define the bending effect on the shell due to local frame reinforcing [24]. Equations for H_M, and H_E are given in Reference [24].

The constraint against the frame yielding failure mode is given by

$$g_4 = (\sigma_{fa} - \sigma_T) / \sigma_{fa} \geq 0 \quad (4.13)$$

where σ_{fa} is the allowable frame stress and σ_T the maximum frame stress. The quantity σ_T is given by

$$\sigma_T = \sigma_b + \sigma_c \tag{4.14}$$

with the compressive hoop stress σ_c given by

$$\sigma_c = Q_p \, pR/(A + bt) \tag{4.15}$$

Here A is the cross-sectional area of the frame and Q_p is the total radial load per inch of circumference. Q_p is given by equation (23) of Reference [24]. The frame bending stress σ_b results from slight out-of-roundness of the hull. It is calculated using

$$\sigma_b = [Ece \, (n^2 - 1) \, / \, R^2] \, [p/ \, (p_{cs} - p)] \tag{4.16}$$

where c is the distance from the midplane of the shell to the surface of the frame, e is the eccentricity from the true circle radius, and n the wave number that minimizes p_{cg}.

It should be noted that σ_b is discontinuous at points near $p = p_{cg}$ and is discontinuous with respect to n, since n is an integer.

Flange buckling is controlled by letting

$$g_5 = \sigma_b - [.5\pi^2 E/12 \, (1 - \nu^2)] \, [x_3/ \, (x_4 - x_2)]^2 \geq 0 \tag{4.17}$$

Two geometric constraint equations can be used. A maximum flange thickness is specified to prevent the flange from becoming excessively thick. It is desirable to specify a maximum flange thickness as a fraction of the plating thickness. Thus, the constraint

$$g_6 = (C_2 t - b') \, / \, C_2 t \geq 0 \tag{4.18}$$

is used, where C_2 is the maximum flange to plating thickness ratio.

The constant C_2 is made arbitrarily large if the designer does not wish to apply this constraint. Minimum flange width is controlled by the inequality

$$g_7 = (w - C_3 h) / w \geq 0 \tag{4.19}$$

to insure that the flange width is sufficient so that the design rule for the control of the web cribbing is not invalidated, C_3 is an arbitrary constant controlling the minimum ratio of flange width to web height.

It is desirable to limit the minimum or maximum (depending on whether internal or external frames are used) radius of the frame because of space consideration. One can then use the constraint

$$g_8 = (R + h + b' + t/2 - R_{min}) / R_{min} \geq 0 \tag{4.20}$$

where R_{min} is the minimum specified radius for internal frames or

$$g_8 = (R_{max} - R - t/2 - h - b') / R_{max} \geq 0 \tag{4.21}$$

for external frames where R_{max} is the maximum specified radius.

To insure against web buckling the following design constraint is invoked.

$$g_9 = \sigma_b - [4\pi^2 E/12\,(1 - \nu^2)]\,(x_2 / x_6)^2 \geq 0 \tag{4.22}$$

Side constraints of the form

$$\left.\begin{aligned} (x_i^U - x_i) / x_i^U \geq 0 \\ (x_i - x_i^L) / x_i \geq 0 \end{aligned}\right\} \tag{4.23}$$

are used to limit the range of the variables x_i for manufacturing, space, or other practical reasons. The flange width clearly cannot

exceed the frame spacing, imposing an upper limit on x_4. Sometimes x_4 is limited by design consideration, thus if $x_4^{U'}$ is the designer specified maximum

$$x_4^U = \begin{cases} x_4^{U'} & , \quad x_4^{U'} \leq x_5 = L' \\ x_5 & , \quad x_4^{U'} > x_5 \end{cases} \tag{4.24}$$

The formulation of the example treated here is now complete, but other forms of constraints and objective functions, which do not arise in this example are clearly possible.

4.3 Discussion of Results

4.3.1 Code description. A FORTRAN IV code called CADOP4 based on this algorithm was developed and used in this study. The code is operational with IBM FORTRAN levels G and H and the UNIVAC TDOS systems. The user is required to supply the objective and constraint functions, the initial values of the variables, and side constraint values.

The initial step size is internally generated so that at the starting point a step of size α^0 in the ∇f direction produces a one percent change in the value of the objective function. This step is constrained so that $\alpha^0 > .01$. The minimum step size is defined as $\alpha_{min} = \alpha^0/1000$. The program uses $e_1 = 10^{-5}$, $e_2 = e_3 = 10^{-6}$, $N_1 = 6$, $N_2 = 10$, $C_1 = 10^5$ and the activity limit constant is initially specified as $K_j = 0.01$ for all constraints.

Nondimensional constraint equations of the form

$$g_j(x_i) = (U_k - B_k) / U_k \geq 0 \quad \text{when } U_k \neq 0 \text{ and } B_k \neq 0$$

or (4.25)

$$g_j(x_i) = (B_k - L_k) / L_k \geq 0 \quad \text{when } L_k \neq 0 \text{ and } B_k \neq 0$$

are used. Otherwise, a dimensional form of equations is used.

A constraint given as $0 \leq b(x_i) \leq A$ would be written as two constraint equations. For example, one could be of the form of the first of equations (4.25) where $B_1 = b(x_i)$ and $U_1 = A$. The second would be of the form $g_2 = -b(x_i)$ since $L_k = 0$.

4.3.2 Code application. The above problem was treated using CADOP3 and CADOP4 codes. Except that the initial step size is specified for this problem as $\alpha = 4/R$, where R is the radius of the shell. The minimum step size is defined as $\alpha_{min} = \alpha^0/200$. Runs were made on an IBM 370 model 168 using FORTRAN level G. The same starting points were used in these runs. The execution time in seconds along with other necessary information were printed at each base point in order to closely track the synthesis path of each procedure.

Tables 1 and 2 present the results obtained from CADOP3 and CADOP4 codes, respectively. A total of 72810 function evaluations were performed using CADOP3 code prior to termination. Due to the great execution time required by CADOP3 for this problem, the run was terminated prior to reaching the optimum. The execution time for this run was 56.2 seconds. CADOP4 code required only about 4000 function evaluations with 4.8 seconds execution time to reach a similar

level of convergence.

The constraint presenting the web buckling (g_9) was active from near the beginning of the search, while the plate yielding (g_3) became active after a relatively short time and remained active through the rest of the search. The general buckling constraint was also active at the latter part of the search. Thus, the optimal design is constrained by general instability, local web buckling and plating yield.

The CADOP3 code came within 1% of the optimum point after 23 seconds and approximately 43000 function evaluations, while CADOP4 required only 1.4 seconds and about 1000 function evaluations to reach this level of convergence. It should be pointed out that for most practical engineering purposes, a 1% level of accuracy is quite acceptable. It is the refining process that is responsible for the greatest portion of the total execution time.

It must be pointed out that due to the presence of intermediate print statements, the execution time required for solution without these statements is appreciably less than the above mentioned time. A run made without any intermediate print statements using CADOP4 code terminated after 2.1 seconds at the optimum point while the original run required 8.7 seconds.

Comparison of the above results demonstrate the apparent superiority of the Boundary Tracking algorithm to the DSFD procedure with respect to speed. This is the result of great reduction in the number

TABLE 1

SYNTHESIS PATH FOR THE SHELL DESIGN PROBLEM USING DSFD METHOD

No. of Bases	Obj. Function Value	x_1	x_2	x_3	x_4	x_5	x_6	No. of Obj Function Evaluations	No. of Constraint Function Evaluations	Constraints Active	CPU Time in Sec.
0	.191149	1.	.5	.5	10	10	20	3	22		.02
50	.131655	.91745	.39056	.21840	6.5253	12.9584	19.83179	1510	3209	3, 9	2.13
100	.1128944	1.19171	.35006	.13070	4.20835	27.60905	16.13659	2657	6332	3	4.21
150	.10910318	1.19397	.29135	.11307	3.2443	24.03268	13.11856	4044	9974	3	6.52
200	.108462	1.19684	.28109	.10376	2.92241	23.7068	12.60682	6782	14494	3	9.96
250	.1081338	1.1984	.27466	.10326	2.6263	23.3650	12.27031	8519	19026	3	13.37
300	.107774	1.19696	.2675	.10326	2.26339	22.437	11.887	10529	23145	3	16.67
350	.107417	1.20908	.25968	.10326	1.08243	23.6244	11.48603	12621	27486	3	20.29
400	.107051*	1.20184	.25190	.10326	1.44585	21.4827	11.0595	14621	32447	3	26.99
450	.106670	1.16294	.22833	.20389	1.02888	16.8998	10.1515	15925	35804	3, 9	31.79
500	.106545	1.1822	.22409	.25339	1.03081	16.62096	9.9441	17511	39112	3	37.51
550	.106497	1.18498	.21779	.29590	1.00365	16.785	9.7805	19730	43882	1, 3, 9	46.46
600	.106483	1.18528	.21826	.34346	.93264	16.7932	9.70967	21578	47987	1, 3, 9	52.5
623	.106477**	1.18425	.21752	.35553	.89695	16.57193	9.67398	22580	50230	1, 3, 9	56.21

*Converged within 1% of the optimum point after approximately 375 bases.

**Within .2% of the optimum point.

TABLE 2

SYNTHESIS PATH FOR THE SHELL DESIGN PROBLEM USING THE BOUNDARY TRACKING METHOD

No. of Bases	Obj. Function Value	x_1	x_2	x_3	x_4	x_5	x_6	No. of Obj Function Evaluations	No. of Constraint Function Evaluations	Constraints Active	CPU Time in Sec.
Starting Point	.191149	1.0	.5	.5	10.0	10.0	20.0	1	10		
1	.146414	.95870	.37494	.37567	9.93493	11.918	19.93510	19	95	9	.11
10	.118556	1.3393	.37116	.27278	8.52995	56.78	18.52975	54	401	2, 9	.52
20	.11011	1.26089	.26746	.12878	2.20746	30.388	12.20792	81	664	9	.98
30	.10726*	1.20859	.25749	.10027	1.45954	23.173	11.3766	134	1085	3	1.63
40	.106886	1.17902	.24437	.10019	.74477	17.08	10.6618	192	1475	3	2.29
50	.106721	1.18417	.23362	.38917	.67781	17.49	10.0889	242	1822	3	2.98
60	.106652	1.18309	.22614	.471329	.75997	17.08	9.7476	316	2288	3	3.89
70	.106596	1.1837	.21881	.55404	.8405	16.99	9.4114	390	2766	3	4.81
80	.106535	1.1777	.21182	.57461	.85095	15.7	9.0787	456	3204	3	5.61
90	.106400	1.18178	.20257	.94152	.62714	15.868	8.6507	514	3603	1, 3	6.37
100	.106286	1.18286	.19107	1.0497	.60277	15.572	.503	614	4318	1, 3, 9	7.72
110	.106283	1.18246	.190911	1.05799	.59457	15.485	8.4948	632	4477	1, 3, 9	8.01
120	.1062828	1.18199	.19072	1.0673	.58527	15.384	8.4855	650	4636	1, 3, 9	8.27
130	.106275**	1.18188	.19068	1.0693	.58319	15.361	8.4834	684	4869	1, 3, 9	8.72

*Converged within 1% of the optimum point after approximately 25 bases.

**Optimum point.

of function evaluations required for convergence. It demonstrates the ability of the BT procedure to move efficiently along a difficult behavior boundary.

CHAPTER 5

COMPARISON STUDY

The CADOP4 code as described in section 4.3.1 was used in this study along with all the constants and parameters presented earlier.

This comparison study is based on the works of Eason and Fenton given in References [14] and [15]. All ten problems treated in these references were run, using the new code with the starting points given in Reference [14]. All the control constants and the step sizes are internally generated so that there was no special tuning for individual problems. Computations were performed in double precision on an IBM 370 model 168 system, using a level G compiler, thus closely simulating the Eason and Fenton study.

A brief description of the codes studied are given in Table 3, while Table 4 contains the data for rating the success of the various codes in solving the ten problems to which they were applied. A detailed description of the problems is given in Reference [14]. A normalized time required for the solution of each problem successfully solved is given in Table 4. Normalized time is defined here as the execution CPU time divided by the CPU time required to execute a timing standardization program [14]. The symbol "P" denotes progress toward a solution, and a blank denotes failure. The criteria used in References [14] and [15] for defining successful solution or progress were also applied here.

TABLE 3

CODE DESCRIPTIONS

Name	Description	Algor Class*
ADRANS	Random search followed by pattern moves	DS
CLIMB	Rosenbrock search	DS
DAVID	Davidon-Fletcher-Powell with numerical derivatives	GF
DFMCG	Fletcher-Reeves conjugate gradient method with secant approximation derivatives	G
DFMFP	Davidon-Fletcher-Powell with secant approsimation derivatives	G
FMIND	Hook & Jeeves pattern search	DS
GRAD4	Steepest descent method	G
GRID1	Grid and star network search, with shrinkage	AR
MEMGRD	Davidon-Fletcher-Powell with retained step length information	GF
NMSERS	Simplex search	DS
PATSH	Modified pattern search, dome strategy	DS
PATRNI	Modified pattern search, ridge strategy	DS
RANDOM	Random search with shrinkage	AR
SEEK1	Pattern search followed by random search	DS
SEEK3	Modified pattern search	DSF
SIMPLX	Modified simplex search	DSF
DSDA	Modified pattern search followed by Mugele's search	DS
CADOP2	Modified pattern search followed by Zoutendijk feasible direction method	DS
CADOP3	Modified CADOP2	DS
CADOP4	Boundary Tracking method	BT

*DS = direct search, DSF = direct search employing SUMT strategy and penalty function, G = gradient procedure, GF = gradient procedure using SUMT strategy and penalty function, AR = area reduction method, BT = Boundary Tracking.

TABLE 4

PERFORMANCE OF OPTIMIZATION CODES*

Problem No.	1	2	3	4	5	6	7	8	9	10
Variables	5	3	5	4	2	2	3	2	2	4
Constraints**	10	2	6	0	0	1	2	0	6	0
Code Names										
ADRANS	2.64	0.458	1.65	P	0.100	0.654	0.159	0.069	P	P
CLIMB				0.015	0.005			0.007		
DAVID		0.188	0.84	1.132	0.075	0.046				
DFMCG			P	0.015	P			0.004		
DFMFP			P	0.038	0.250		P	0.387		
FMIND		0.004	P		0.140		0.003	0.003	P	3.74
GRAD4		P	P	0.283	0.393		P	0.004		P
GRID1	P	P		0.033	P	P		0.037	P	P
MEMGRD		0.143	0.91	0.059	0.066	0.067	P			
NMSERS		0.019	0.045	0.060	0.002	0.012	P	0.007	0.39	P
PATSH	1.78	0.220	1.00	0.013	0.018	0.020	0.010	0.009	P	3.54
PATRN0	P	P	P	0.021	P			0.002	P	1.95
PATRN1	P	P	P	0.008	0.002	P		0.001	1.02	1.20
RANDOM		P		P		0.024	1.20	0.013		P
SEEK1	P	P	P		0.010		0.010	0.007	P	1.53
SEEK3		0.102	0.14	0.212	0.035	0.191	0.141	0.013	4.20	P
SIMPLX	2.97	0.297	1.31	0.196	0.035	0.191	0.262	0.050	P	
DSDA	P	0.021	0.047	0.104	0.003	0.008	0.145	0.004	1.94	1.35
CADOP2	0.307	0.016	0.090	0.075	0.008	0.011	0.069	0.003	1.55	1.73
CADOP3	0.119	0.006	0.030	0.080	0.004	0.011	0.025	0.005	1.00	1.60
CADOP4	0.044	0.003	0.0125	0.080	0.004	0.005	0.003	0.005	0.39	1.60

*Numerical entry indicates normalized time required for solution, and P indicates progress toward a solution [14, 15].

**Excluding regional constraints.

The important variables effecting execution time are; the problem, the algorithm, the code, the input and output requirements, the architecture of the machine and the compiler system. In order to minimize the effects of those variables not involved in the comparison, similar computer machines and compiler systems along with the same test problems and output requirements are used here. Unfortunately the effects of the code structure can not accurately be determined. Thus, data of Table 4 and ratings shown in Table 5 can not be considered to accurately reflect the effectiveness of the basic optimization algorithms.

The data and rating procedures used here however, can be directly applied, with reasonable accuracy, for relative comparison of the codes tested. The results obtained are thus useful to the user of such codes. The relative effectiveness of basic optimization algorithms can only be infered from this data if one assumes that the codes for each procedure have been prepared with essentially comparable effectiveness.

This study uses Eason and Fenton's comparison procedures since they are the best available and the most current. Furthermore, use of another investigator's comparison methods rather than one proposed by the developer of a new method reduces the tendency toward developer bias in reporting on the effectiveness of his method.

The data from Table 4 may be applied to a number of rating methods for comparing the codes tested. Table 5 presents relative ranking of

TABLE 5

RELATIVE RANKING OF OPTIMIZATION CODES

Generality				Efficiency				Generality and Efficiency	
	n_a		N_a		f_a		$\bar{f}_a$		T_a
10	CADOP4 CADOP3 CADOP2	10	CADOP4 CADOP3 CADOP2	1.3	PATRN1	0.18 0.19	NMSERS CADOP4	2.2 2.9 4.1	CADOP4 CADOP3 CADOP2
9	DSDA PATSH	9.5	DSDA PATSH	2.5	CADOP4	0.26 0.28	CADOP3 PATRN1	9	DSDA
8	SEEK3 SIMPLX	8.5	SEEK3 SIMPLX ADRANS	3.8 4.1	CADOP3 NMSERS	0.40 0.54	CADOP2 DSDA	15 16	PATSH NMSERS PATRN1
7	ADRANS NMSERS	8.0	NMSERS	6.6	CADOP2	0.82	MEMGRD	18	SEEK3
5	PATRN1 FMIND MEMGRD DAVID	7.0 6.0	PATRN1 FMIND SEEK1	9.5 15.7	DSDA FMIND	0.85 0.97	PATSH FMIND	21 24	SIMPLX ADRANS
4	SEEK1	5.5	MEMGRD PATRN0	23.8	PATSH	1.2	SEEK3	25	FMIND
3	CLIMB DFMFP GRAD4 PATRN0 RANDOM	5.0	DAVID GRAD4 GRID1 RANDOM	24.7 34.8 61.1	SEEK3 MEMGRD SIMPLX	1.9 2.4 2.9	SIMPLX DAVID ADRANS	26 27	MEMGRD DAVID
2	DFMCG GRID1	4.0 3.0	DFMFP CLIMB DFMCG	63.6 92.5	DAVID ADRANS				

the codes using the criteria of Reference [15]. The rating equations are as follows:

Let the number of problems solved by code "a" be n_a, and let n'_a be the number of problems with a "P" rating. Then the numerical success rating N_a is given by

$$N_a = n_a + n'_a/2 \tag{5.1}$$

One of two computational efficiency ratings is given as

$$f_a = [\Sigma_{p=1}^{10} b_{ap} t_{ap} / \min (t_{ap})] / n_a \tag{5.2}$$

where $b_{ap} = 1$ if code "a" solved problem "p" and zero otherwise, t_{ap} is the normalized time required for solution and $\min(t_{ap})$ is the shortest time required by any of the codes studied to solve problem "p". The other efficiency criterion is

$$\overline{f}_a = [\Sigma_{p=1}^{10} b_{ap} t_{ap} / \text{mean } (t_{ap})] / n_a \tag{5.3}$$

where $\text{mean}(t_{ap}$ is the average time required by the codes studied to solve problem "p". An overall rating number which can be considered a composite measure of generality (reliability) and efficiency (speed) is given by

$$T_a = \Sigma_{p=1}^{10} t'_{ap} \tag{5.4}$$

where t'_{ap} is set equal to t_{ap} if algorithm "a" solves a problem "p", and to twice the time used by the slowest code solving a problem "p" if code "a" could not solve it. This penalty time is used to penalize code unreliability. Only codes that solved half or more of the problems are rated for efficiency.

The tables presented here are similar to those given in References [15] and [17], except that the CADOP3 and CADOP4 codes are included.

The CADOP3 code is a modified version of CADOP2 code presented in Reference [17]. Thus, the rating methods and presentation of results used are essentially those of Reference [15] and [17].

From Tables 4 and 5 it can be concluded that CADOP4 is the fastest code in overall generality and efficiency rating. This Boundary Tracking method presented here appears comparable in speed to the fastest methods presented in Reference [17], (NMSERS, PATRNI, DSDA, and CADOP2). CADOP4 is faster than average in all problems. It is the fastest code solving problems 1, 2, 3, 6, 7, and 9, which are problems with behavior constraints. On the other hand, problems 4, 5, 8, and 10, where the CADOP3 and CADOP4 codes performed identically, but less efficiently than some other procedures, are problems without such constraints. This is expected since the two codes are identical in treating the unconstrained problems.

Of the problems tested, problem 1 which has the largest number of behavior constraints (10) was apparently also the most difficult, since it was solved by only six of the twenty-one codes presented in Table 4. The second most difficult problem was problem 9, for which only 7 of the 21 codes led to the optimum point. This problem has only 6 behavior constraints and two variables. However, there is a rapid change in the slope of the behavior boundary in the region near the optimum point. Because of this rapid variation in slope, a major part of the total execution time was spent on refining the location of the point so as to meet the convergence criterion. Problem 10, which required the longest execution time, has no behavior constraints. The long execution time

required by this problem is due to the complexity of the objective function and the number of the function evaluations needed to reach the optimum.

It can be seen from Table 4 that the CPU time required for treating problems 9 and 10 in most cases is much greater than the sum of the CPU times required for the solution of the rest of the problems tested. Thus, the speed of a code in solving problems 9 and 10 plays an overwhelming role in determining the code's overall generality and efficiency rating, T_a. For example, a code that is very efficient on the majority of problems but requires long solution times for problems 9 and 10 would have a relatively low overall generality and efficiency rating using equation (5.4). Therefore, this rating method is not very useful for comparing codes solving all the test problems. For such codes the efficiency ratings f_a and $\overline{f}_a$ represent the ability of the method more clearly. Furthermore, the overall generality and efficiency rating T_a does not sufficiently penalize those codes which failed to solve any of the test problems 1 through 8, but performed well on problems 9 and 10. The penalty time used here fails to accurately take into account the weaknesses of such codes in solving problems 1 through 8.

The superior performance of CADOP4 code in problems with behavior constraints compared to CADOP3 code is due to the fact that the number of function evaluations required in CADOP3 is proportional to the number of bases needed for solution multiplied by the number of variables. On the other hand, the number of function evaluations in CADOP4 is

proportional only to the number of base points, and does not have to be multiplied by the number of variables. Therefore, in CADOP3 the number of function evaluations increases directly with the number of variables, while in the CADOP4 code it does not.

Compared to the relatively reliable codes ($n_a \geq 8$), CADOP4 is faster than CADOP3 on six of the ten problems solved by CADOP3, faster than DSDA on six of nine, and faster than PATSH on eight of nine, and faster than SEEK3 in all problems solved by these schemes. As may be seen, the CADOP2 and CADOP3 codes are the only reliable codes comparable to CADOP4. CADOP4 is significantly faster than CADOP3, CADOP2, PATSH, SEEK3, and SIMPLX. The straight pattern (PATRN1) or simple (NMSERS) codes are ranked relatively high in efficiency primarily due to their superior performance in unconstrained problems (Table 5). Of this group, only NMSERS appears to be sufficiently reliable to merit consideration for use. The minor difference in efficiency between NMSERS and CADOP4 is, however, overshadowed by the superior reliability of the latter. Thus, in the overall speed, generality, and efficiency rating, CADOP4 stands out. Viewed on the basis of these comparisons, CADOP4 appears to be a superior nonlinear mathematical programing code.

The new algorithm is intended to be a constrained optimization algorithm. Therefore, the performance of CADOP4 is compared to the codes of References [15] and [17] on problems having behavior constraints. This comparison is shown in Table 6.

Table 6 is similar to Table 5 except that the problems without

TABLE 6

RANKING OF OPTIMIZATION CODES IN PROBLEMS WITH BEHAVIOR CONSTRAINTS

Generality				Efficiency				Generality and Efficiency	
	n_a		N_a		f_a		$\bar{f}_a$		T_a
6	CADOP4 CADOP3 CADOP2	6	CADOP4 CADOP3 CADOP2	1.0	CADOP4	0.07	CADOP4	0.46	CADOP4
								1.2	CADOP3
	DSDA		DSDA	3.2	CADOP3	0.15	NMSERS		
5	PATSH SEEK3 SIMPLX ADRANS	5.5	PATSH SIMPLX ADRANS	3.3	NMSERS	0.19	CADOP3	2.0	CADOP2
				8	CADOP2	0.43	CADOP2	8.1	DSDA
		5	SEEK3	13	DSDA	0.83	DSDA	10.7	SEEK3
4	NMSERS	4.5	NMSERS	28	SEEK3	1.0	PATSH	11.4	PATSH
3	MEMGRD DAVID	3.5	MEMGRD	39	PATSH	1.1	MEMGRD	13.4	SIMPLX
2	FMIND RANDOM		PATRN1 FMIND	43	MEMGRD	1.2	DAVID	14	ADRANS
		3	SEEK1	63	DAVID	1.3	SEEK3	17	NMSERS
1	PATRN1 SEEK1		DAVID	78	SIMPLX	2.1	SIMPLX	18	MEMGRD
		2.5	RANDOM	10.4	ADRANS	3.2	ADRANS	18	DAVID
	CLIMB DFMFP GRAD4		PATRNO						
		2	GRID1						
0	PATRNO DFMCG GRID1	1.5	GRAD4						
		1	DFMFP						
		0.5	DFMCG						
		0	CLIMB						

behavior constraints are excluded. The rating equations used for these values are the same as those used for Table 5. The results of Table 6 indicate the superiority of CADOP4 with respect to speed, generality, and efficiency. Only codes that solved half or more ($n_a \geq 3$) of the problems are rated for efficiency.

The superior performance of the CADOP4 code in all the above ratings strongly suggests the superiority of the BT method for treating problems with behavior constraints. CADOP4 proved to be substantially more efficient than the CADOP3 and NMSERS codes on such problems. In the overall generality and efficiency rating CADOP4 code again stands alone with CADOP3 code in the second place being approximately three times slower. However, as explained earlier, due to the relatively long execution time required for problem 9 this rating method does not reflect accurately the relative effectiveness of the codes which solved all the test problems. The efficiency ratings f_a and $\overline{f}_a$ may again be more useful in comparing such codes. In these ratings CADOP4 code is two to three times faster than other relatively fast codes (CADOP3, NMSERS).

Thus, the Boundary Tracking algorithm presented here appears to be superior to DSFD with regard to efficiency and to all other optimization procedures tested with respect to generality and efficiency.

Based on its performance on the ten problems of Reference [15] and on the other comparison studies, the Boundary Tracking method appears to be a fast and reliable nonlinear mathematical programing optimization procedure.

In conclusion, it must be pointed out that the test problems used in the above comparison study are relatively small and simple, while many practical engineering problems are large and contain complex and computationally demanding functions. Therefore, although it is reasonable to assume that the above comparison study is representative of the performance of the new code on most practical problems of size and complexity similar to that of the study, this performance may not be representative for large complex problems. Unfortunately, no comparison study utilizing large complex problems is available, and is not likely to be available soon due to the difficulty and cost associated with such a project. Furthermore, one cannot guarantee that the performances noted in the present study are typical for all relatively small simple problems. Nevertheless, this is the most complete of the available general comparison studies and should be useful as a guide in selecting a suitable algorithm for a particular problem. The designer must, however, proceed with care in comparing his problem to the above test problems, taking into account such things as the number of variables, constraints, nature of the function to be evaluated, etc. in selecting a desirable algorithm.

CHAPTER 6

CONCLUSION

The successful application of the CADOP4 code to the relatively complex problem of shell synthesis and its performance in the general comparison study presented in section 4.1, imply that the Boundary Tracking method developed here is a superior nonlinear mathematical programming procedure. Although the code proved to be the best in the general comparison study, the real potentials of the algorithm are demonstrated in the shell design problem. Since many engineering problems are of such a class, the new algorithm shows the promise of adding a major contribution to the field of automated design.

In the above studies the BT method proved to have great potential for use in computationally demanding problems. However, as in the case of most new methods, additional studies may lead to improvement, particularly in the speed of the algorithm.

The only apparent disadvantage of CADOP4 is its relative complexity compared to some of the other reasonably reliable methods. It contains 1320 FORTRAN statements, while, for example, the CADOP3 contains 790, DSDA contains 372, and PATSH only 75 FORTRAN statements. Thus, if CADOP4 code is not available in a compiled form, for simple problems of relatively low complexity and dimensionality, one of the simpler codes may be preferable since the time required for compilation of the program may exceed the time saved by using CADOP4.

In addition, its performance on problems without behavior constraints, although quite good, is not as outstanding as on problems with such constraints.

In conclusion, the user of this or any other algorithm for automated design, should be aware that the general nonlinear constrained optimization problem is quite difficult to handle. Also, none of the available techniques will guarantee an optimum solution. One should, therefore, be careful in the application of such an algorithm and analyze the results thoroughly. One also, should make use of several synthesis runs, using different starting points where possible, before assuming the value is an optimum solution.

LIST OF REFERENCES

1. Shen, C. Y., and Prager, W., "Recent Developments in Optimal Structural Design," Appl. Mech. Rev., Oct. 1968, pp. 985-992.

2. Wasintynski, Z., and Brandt, A., "The Present State of Knowledge in the Field of Optimum Design of Structures," AMR, 16, 1963, pp. 341-350.

3. Parewonesky, P., "Optimal Control; A Review of Theory and Practice," AIAA J., Nov. 1965, pp. 1985.

4. Gerard, G., "Optimum Structural Design Concepts for Aerospace Vehicles," J. Spacecraft and Rockets, Jan. 1966, pp. 5-18.

5. Hadley, G., Linear Programming, Addison-Wesley, 1964.

6. Hadley, G., Nonlinear and Dynamic Programming, Addison-Wesley, 1964.

7. Graves, R. L., and Wolf, P., Recent Advances in Mathematical Programming, McGraw-Hill, 1963.

8. Schmit, L. A., and Mallet, R. H., "Structural Synthesis and Design Parameter Hierarchy," J. Struct. Div. Am. Soc. Civil Engrs., Aug. 1963, pp. 269-299.

9. Schmit, L. A., Kicher, T. P., and Morrow, W. M., "Structural Synthesis Capability for Integrally Stiffened Waffle Plates," AIAA J., Vol. 1, No. 12, Dec. 1963, pp. 2820-2836.

10. Schmit, L. A., "Automated Design," International Science and Technology, No. 54, June 1966, pp. 63-78 and 115-117.

11. Mangasarian, O. L., "Techniques of Optimization," J. of Engineering for Industry, Trans, ASME, Series B, Vol. 94, No. 2, May, 1972, pp. 365-372.

12. Wilde, D. J., Optimum Seeking Methods, Prentice Hall, 1964.

13. Abaide, J. Nonlinear Programming, North Holland Pub. Co. 1967.

14. Eason, E. D., and Fenton, R. G., "Testing and Evaluation of Numerical Methods for Design Optimization," UTIME-TP 7204, University of Toronto, Canada, 1972, 1972.

15. Eason, E. D., and Fenton, R. G., "A Comparison of Numerical Optimization Methods for Engineering," J. of Engineering for Industry, TRANS. ASME, Series B, Vol. 96, No. 1, Feb. 1974, pp. 196-201.

16. Moradi, J. Y., "A Direct Search Mathematical Programming Alforithm," Master's Thesis, Newark College of Engineering, 1974.

17. Pappas, M., and Moradi, J. Y., "An Improved Direct Search Mathematical Programming Algorithm," J. of Engineering for Industry, Vol. 97, No. 4, Nov. 1975, pp. 1305-1310.

18. Pappas, M., "Performance of the 'Direct Search Design Algorithm' as a Numerical Optimization Method," AIAA Journal, Vol. 13, No. 6, June 1975, pp. 827-829.

19. Shampine, L. F., and Allen, R. C., Numerical Computing; an Introduction, Saunders Co. 1973.

20. Zoutendijk, G., Methods of Feasible Directions, Elsevier, Amsterdam, 1960.

21. Wilde, O. L., and Beightler, C. S., Foundation of Optimization, Prentice-Hall, Englewood Cliffs, N. J. 1967.

22. Pappas, M. and Allentuch, A., "Automated Optimal Design of Frame Reinforced, Submersible, Circular, Cylindrical Shells," J. of Ship Research, Vol. 17, No. 4, Dec. 1973, pp. 206-216.

23. Pappas, M., "An Improved Direct Search-Feasible Direction Optimization Algorithm," N.J.I.T. Report No. NV 10, 1977.

24. MacNaught, D. F., "Strength of Ships," Principles of Naval Architecture, SNAME, 1967, Chap. IV, Section 8.

25. Pappas, M. and Allentuch, A., "Pressure Hull Optimization Using General Instability Equation Admiting More than One Longitudinal Buckling Half-Wave," J. of Ship Research, Vol. 19, March 1975, pp. 18-22.

26. Pappas, M. and Amba-Rao, C., "A Discrete Search Procedure for the Minimization of Stiffened Cylindrical Shell Stability Equations," AIAA Journal, Vol. 8, No. 11, Nov. 1970, pp. 2093-2094.

USER INSTRUCTIONS FOR CADOP4

I. INSTRUCTION

This code treats inequality constrained or unconstrained linear or nonlinear minimization problems by means of a direct search mathematical programming method. The method starts from a user specified starting or initial point and generates a sequence of better points until, hopefully, an optimal, or near optimal, point is reached. The code is intended primarily for constrained nonlinear problems. Linear problems are much more effectively handled by one of many linear programming methods. Unconstrained problems are better treated by UNCDP, a simplified version of CADOP2, or one of several other unconstrained nonlinear problem codes. CADOP4 will, however, treat both linear and unconstrained problems.

It should be noted that nonlinear mathematical programming methods generally do not guarantee an optimal solution because of the possibility of multiple local optima, algorithm failure or numerical difficulties. Although CADOP4 is among the most reliable of the nonlinear mathematical programming codes it cannot be considered absolutely reliable. Thus, it is desirable to use several optimization runs with widely separated starting points to confirm the convergence has been achieved or to determine if local optima are present.

II. PROBLEM FORMULATION

This code treats the problem:

Find those values $\bar{x}_i$ of the set of "variables" x_i, $i = 1,2---IP$ and the constant "parameters" p_k, $k = 1,2---KP$, that result in the minimum value of the objective function $f(p_k,x_i)$, that is

$$f(p_k,\bar{x}_i) = \min [f_k(p,x_i)] \tag{1}$$

while also satisfying the "behavior" constraints

$$g_j(p_k x_i) \geq 0 \qquad j = 1,2,---JP \tag{2}$$

and the "regional" constraints

$$x_i^l \leq x_i \leq x_i^u \tag{3}$$

The program in its present form treats problems with

$2 \leq IP \leq 10$, $0 \leq JP \leq 10$ and $0 \leq KP \leq 100$

III. SPECIFICATION OF OBJECTIVE AND CONSTRAINT FUNCTIONS

The subprogram FUNCTION OBJ(J) is essentially a dummy subprogram created to accept the users FORTRAN IV program statements defining the objective and behavior constraint functions. The normal form of the behavior constraints is

$$B_j\ (p_k, x_i) \quad \leq \quad U_j\ (p_k, x_i) \qquad (4)$$

where B_j can be thought of as the behavior and U_j the upper limit of behavior.

The FUNCTION OBJ(J) is modified so as to define the objective and constraint functions by inserting the following statements immediately after the COMMON statement

```
      GO TO (1,2,---jp), J     (omit if no constraints are used)
      OBJ = expression defining f (p_k,x_i) using arrays P(k), X(i)
      RETURN
    j B = expression defining B_j
      U = expression defining U_j
      GO TO 101
```

A constraint statement group is used for every constraint. The last constraint group need not use the final GO TO statement.

Example

For the problem:

minimize

$$2x_1 x_2 - p_1 x_3 x_4^2$$

such that

$$2x_3^3 - x_1 \leq p_2$$

and

$$-6 \leq x_2^2 - x_4 \leq 0$$

one could define three constraints and insert the statements

```
      GO TO (1,2,3), J
      OBJ = 2. * X (1) * X (2) - P (1) * X (3) * X (4) **2
      RETURN
1     B = 2.* X (3) ** 3 - X (1)
      U = P(2)
      GO TO 101
2     B = X(2)** 2 - X(4)
      U = 0.
      GO TO 101
3     U = (X (2) ** 2 - X (4) )
      B = - 6.
```

Alternately, noting that both upper and lower limits on the second constraint equation cannot be active simultaneously one could use two constraint equations and insert the program segment:

```
      GO TO (1,2),J
      ORJ = 2. * X (1)* X(2) - P (1) * X (3) * X (4) ** 2
      RETURN
    1 B = 2. * X (3) ** 3 - X (1)
      U = P(2)
      GO TO 101
    2 TB = X (2) ** 2 - X (4)
      LF(TB.LT.-3.)  GO TO 3
      B = TB
      U = 0
      TO TO 101
    3 B = -6.
      U = TB
```

The contraints values at the optimum are printed in the form of equation (2) where

$$g_j = \begin{cases} -(B_j - U_j)/|U_j| & U_j \neq 0 \text{ and } B_j \neq 0 \\ -(Bj - U_j) & U_j = 0 \text{ or } B_j = 0 \end{cases}$$

thus a positive value of g_j indicates the constraint is satisfied.

IV DATA CARDS

The program allows multiple optimization runs of a particular problem within a single computer run using different parameters, for parametric studies, different starting points, for optimality confirmation or search for local optima, and different sets of regional constraint limits.

The user must, by use of the data card set, specify the number of parameters, variables and behavior constraints used. specify for the first run if regional constraints are to be used, and for subsequent runs whether the parameters, initial point or regional limits are to be changed. If parameters are used they must be entered. The initial point must be entered. The values for the regional constraints must be entered if such limits are imposed.

The first data set is entered on the "Problem Constraints" card which specifies, in order, the number of parameters (KP), variables (IP) and behavior constraints (JP), used (see line 8 of program listing). The data is entered in 3I10 format in the first three fields in order and must be right justified.

The 2nd data set is entered on the "Data Control" card and is used to specify whether: 1) new parameters are to be used (ICNTR2), 2) a new initial point is to be used (ICNTR3) and, 3) regional limits or new regional limits are to be used (ICNTR4) (see lines 12-23). The entries are made in 3I10 format in the order given above. For the first run no entries in

the first two fields are needed. If regional limits are used an entry, any nonzero digit, is made in the third field (col. 21-30).

If, and only if, the number of parameters specified is greater than zero (KP > 0) a 3rd data set is used to enter the problem parameters p_k. The entries are made 5 to a card in F15.5 format in order i = 1,2 ---- KP, (see lines 13 and 14).

The 4th data set specifies the starting point and entries are made, in order, 8 to a card in F10.5 format (see lines 7 and 15).

If, and only if, an entry is made in the third field of the Data Control card (ICNTR4 $\neq$ 0) a 5th data set defining the lower limits is entered, 8 to a card, in F10.5 format followed by an upper limit set (starting with a new card) with similar format. Both complete limit sets must be entered (see lines 22-25).

For every additional run an additional Data Control card is added followed by parameter and/or initial point and/or regional limit sets, where and only where the need for a new set is indicated by an appropriate entry (ICNTR2 $\neq$ 0 or (ICNTR3 $\neq$ 0 or ICNTRL4 $\neq$ 0) on the Data Control card (see lines 18-22).

A blank Data Control card is used to terminate the run(s) (see lines 18 and 19).

V. CONVERSION TO SINGLE PRECISION

The program as supplied is a double precision form. Experience has shown, however, that the great majority of problems can be treated adequately in single precision and thus it is recommended that single precision be tried first. To convert to single precision, remove the IMPLICIT statement at the beginning of MAIN and all subprograms. Insert function conversion statements such as DSQRT(B) = SQRT(B), etc. after the COMMON block in the subprograms where such functions are used. Furthermore, the REAL FUNCTION OBJ*8(J) should be put in single precision form.

Storage requirements for the basic program in double precision is about 150K and in single precision about 100K.

VI. CHANGE OF PROBLEM SIZE

To change the maximum number of variables (IP) or the maximum number of constraints (JP) the program can treat, change the arrays in MAIN and all subprograms as follows:

1. Change arrays of the COMMON statements of the main and all subprograms as follows;

   D, DL, DU, and DZ become D(IP), DL(IP), DU(IP) and DZ(IP).

   XB becomes XB(IP,5).

   V(A in OPT2, CONMOV and LINPRO) becomes V(M,N). Where

   $$M = JP + 4IP + 5$$

   $$N = JP + 5IP + 6$$

   GN (GO in OPT2, CONMOV, ZERO, and G in PATTRN and FIND) becomes GN(JP). IA becomes IA(JP).

   where IP is the new maximum number of variables and JP the new number of constraints

2. In COMMON/DOL/ change BL array to BL(JP) wherever this statement is used.

3. In COMMON/ZE/ change DT and DTT to DT(IP) and DTT(IP).

   change SE to SE(JP).

   change DK to DK(JP + 1, IP).

4. In COMMON/POL/ change GB to GB(JP).

5. In COMMON/CON/ change SS to SS(IP).

   CHANGE IJ and JI to IJ(JP) and JI(JP).

6. In COMMON/FIN/ change x to x(IP).

   change IA, CK and IC to IA(JP), CK(JP) and IC(JP)

7. In the SUBROUTINE DELF Dimension statement change DT and DTT to DT(IP), DTT(IP) and GO to GO(JP).

8. In the SUBROUTINE DB DIMENSION statement change DT to DT(IP).

9. In the OPT2 SUBROUTINE DIMENSION statement change IB, IC, and X to IB(IP) etc.. TN and GG to TN(JP), GG(JP). change SSS to SSS(M), where M is defined in item 1 as DB1 to DB1 (JP + 1, IP).

10. In the SUBROUTINE CONMOV DIMENSION statement change GG to GG (JP) and X to X (IP)

11. In the SUBROUTINE ZERO DIMENSION statement change GG to GG (JP).

12. In the SUBROUTINE FIND DIMENSION statement change DT(10) to DT(IP).

13. In the SUBROUTINE PATTERN DIMENSION statement change all 10's to IP.

14. Change all dummy arrays such as DUM, IDUM, IDUN, etc. in all COMMON statements to equalize COMMON size.

```
FORTRAN IV G LEVEL  21                  MAIN              DATE = 77068          18/00/22              PAGE 0001

0001                IMPLICIT REAL*8(A-H,O-Z)
0002                COMMON D(10),P(100),DL(10),DU(10),DZ(10),A,XB(10,5),GN(10),FB(5),
                   CV(55,66),Q,IP,KP,JP,IDUN(22)
0003                 COMMON IXJ
0004          1     FORMAT(5F15.5)
0005           2    FORMAT(4I10)
0006          3     FORMAT(2F10.5)
0007          101   FORMAT(8F10.5)
0008                READ2,KP,IP,JP
0009             45 DO 21 I=1,IP
0010                DL(I)=-1.E+48
0011           21   DU(I)=1.E+48
0012                READ2,ICNTR2,ICNTR3,ICNTR4
0013                IF(KP.EQ.0) GO TO 12
0014                READ1,(P(I),I=1,KP)
0015           12   READ101,(D(I),I=1,IP)
0016           18   FORMAT(1HO,' STARTING DESIGN VARIABLE VALUES' )
0017                GO TO 23
0018          24    READ(5,2,END=33)ICNTR2,ICNTR3,ICNTR4
0019                 IF(ICNTR2+ICNTR3+ICNTR4.EQ.0)STOP
0020                 IF(ICNTR2.NE.0)READ1,(P(I),I=1,KP)
0021                IF(ICNTR3.NE.0)READ101,(D(I),I=1,IP)
0022           23   IF(ICNTR4.EQ.0)GO TO 22
0023                READ 101,(DL(I),I=1,IP)
0024           17   FORMAT(1HO,' LOWER LIMITS OF DESIGN VARIABLES')
0025                 READ 101,(DU(I),I=1,IP)
0026           19   FORMAT(1HO,' UPPER LIMITS OF DESIGN VARIABLES')
0027          22    IF(KP.LE.0) GO TO 104
0028                PRINT 6
0029                PRINT7,(K,P(K),K=1,KP)
0030          104    PRINT 18
0031                PRINT101,(D(I),I=1,IP)
0032                 PRINT 17
0033                PRINT101,(DL(I),I=1,IP)
0034                 PRINT 19
0035                PRINT101,(DU(I),I=1,IP)
0036                 CALL DELF(FOPT)
0037           198   PRINT 180,FOPT
0038           180   FORMAT( ' OPTIMUM DESIGN=',F15.9)
0039          6     FORMAT( '0DESIGN PARAMETERS'/)
0040          7     FORMAT(' P',I2,'=',F14.4)
0041                PRINT8
0042          8     FORMAT( '0DESIGN VARIABLE VALUES'/)
0043                PRINT9,(K,D(K),K=1,IP)
0044          9     FORMAT(' D',I1,'=',F15.8)
0045                IF(JP.EQ.0) GO TO 24
0046                PRINT10
0047          10    FORMAT( '0NEARNESS TO CONSTRAINTS'/)
0048                 PRINT 11,(GN(K),K=1,JP)
0049           11    FORMAT(' G',I2,'=',F15.5)
0050           100   GO TO 24
0051          33     STOP
0052                 END
```

```
0001                SUBROUTINE DELF(FOPT)
0002               IMPLICIT REAL*8(A-H,O-Z)
0003               COMMON D(10),P(100),DL(10),DU(10),DZ(10),A,XB(10,5),GN(10),FB(5),
                  CV(55,66),Q,IP,KP,JP,IDUN(22)
0004                COMMON IXJ
0005                COMMON/DDL/STEST,TDIF,AX1,AMIN,BL(10),AO,IN
0006                COMMON/PAT/OBJ,JE,IXYZ
0007               DIMENSION GO(10),DT(10),DTT(10)
0008                 IXYZ=1
0009                KT=0
0010                STEST=0.
0011                IN=0
0012                AO=1.
0013                K=1
0014                A=1.
0015                AX1=1.
0016                FTEST=0.
0017                TDIF=0
0018                JK=0
0019                LI=0
0020                IXJ=0
0021                IF(JP.EQ.0) GO TO 51
0022                DO 3 I=1,JP
0023                BL(I)=1.
0024          3     GO(I)=10.
0025        510     K=1
0026                IF(LI.GT.0) GO TO 31
0027                DO 5 J=1,JP
0028                GN(J)=OBJ(J)
0029          5     CONTINUE
0030          520   FB(K)=OBJ(0)
0031          31    IF(LI.EQ.1) AX1=10.*AX1
0032          51    CALL DB(0,LI)
0033                IF(JP.EQ.0) GO TO 1000
0034                PRINT 514,(D(I),I=1,IP),FB(K)
0035        514     FORMAT(12F10.5)
0036                IF(KT.EQ.0) GO TO 45
0037               DIF=DABS((FTEST-FB(1))/FB(1))
0038                IF(DIF.GT. 1.E-15) GO TO 45
0039                IF(DIF.GT.TDIF) GO TO 45
0040                FOPT=FB(K)
0041                RETURN
0042          45    IF(A.LT.AMIN) GO TO 1771
0043        450     KT=KT+1
0044                IF(KT.EQ.10) GO TO 1000
0045                TDIF=DIF
0046        511     FTEST=FB(1)
0047                SUN=0.
0048                DO 81 I=1,IP
0049          81    SUN=SUN+DZ(I)**2
0050                SUN=DSQRT(SUN)
0051                IF(SUN.EQ.0) GO TO 1771
0052        5121    CONTINUE
0053                IF(A.GT.AMIN) GO TO 5120
0054        1771    FOPT=FB(K)
0055                RETURN
```

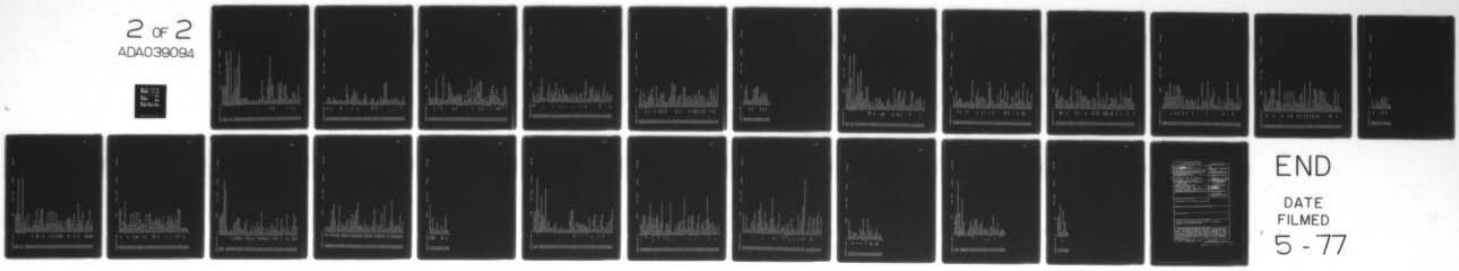
```
0056        5120    AL=A
0057        14      SUMM=0.
0058                DO 82 I=1,IP
0059                DT(I)=D(I)
0060                XB(I,K)=D(I)
0061         8      D(I)=D(I)-DZ(I)*AL/SUN
0062                IF(D(I).GT.DU(I)) D(I)=DU(I)
0063                IF(D(I).LT.DL(I)) D(I)=DL(I)
0064         82     SUMM=SUMM+(D(I)-DT(I))**2
0065                SUMM=DSQRT(SUMM)
0066                IF(SUMM.EQ.0) GO TO 1771
0067                DO 83 I=1,IP
0068                D(I)=DT(I)+(D(I)-DT(I))*AL/SUMM
0069                IF(D(I).GT.DU(I)) D(I)=DU(I)
0070                IF(D(I).LT.DL(I)) D(I)=DL(I)
0071        83      CONTINUE
0072                IXJ=0
0073        630     IF(JP.EQ.0) GO TO 12
0074                I=0
0075                DO 9 J=1,JP
0076                GO(J)=GN(J)
0077                GN(J)=DBJ(J)
0078                IF(GN(J).GT.0.) GO TO 9
0079                I=I+1
0080         9      CONTINUE
0081                IF(I) 12,12,13
0082         12     K=K+1
0083                FB(K)=DBJ(0)
0084                DO 1818 L=1,IP
0085        1818    XB(L,K)=D(L)
0086                IF(FB(K).GT.FB(K-1)) GO TO 19
0087                IF(LI.GT.0) GO TO 1401
0088         18     AL=2.*AL
0089        1401    IF(K.LT.5) GO TO 14
0090                FB(1)=FB(K-2)
0091                FB(2)=FB(K-1)
0092                FB(3)=FB(K)
0093                DO 141 I=1,IP
0094                XB(I,1)=XB(I,K-2)
0095                XB(I,2)=XB(I,K-1)
0096                XB(I,3)=XB(I,K)
0097        141     CONTINUE
0098                K=3
0099         140    GO TO 14
0100         13     K=K+1
0101                DO 15 N=1,IP
0102         15     XB(N,K)=D(N)
0103                KU=0
0104                KS=1
0105                GSS=GN(1)
0106                DO 17 J=2,JP
0107                IF(GSS.LT.GN(J)) GO TO 17
0108                GSS=GN(J)
0109                KS=J
0110         17     CONTINUE
0111                J=KS
```

```
0112                DO 23 L=1,IP
0113                DT(L)=XB(L,K-1)
0114          23    DTT(L)=XB(L,K)
0115                IF(DABS(GN(J)).LE..0000001) GO TO 1701
0116                GS=GO(J)
0117          1031  T=GS/(GS-GSS)
0118          1773  CONTINUE
0119                DO 24 L=1,IP
0120                D(L)=DT(L)+T*(DTT(L)-DT(L))
0121                IF(D(L).LT.DL(L)) D(L)=DL(L)
0122          24    IF(D(L).GT.DU(L)) D(L)=DU(L)
0123                IXJ=0
0124                GN(J)=OBJ(J)
0125                IF(DABS(GN(J)).GT..000001) GO TO 100
0126                GO TO 170
0127          100   GS=GSS
0128                GSS=GN(J)
0129                DO 1021L=1,IP
0130                DT(L)=DTT(L)
0131          1021  DTT(L)=D(L)
0132                GO TO 1031
0133          170   CONTINUE
0134                IXJ=0
0135                DO 716 J=1,JP
0136          716   GN(J)=OBJ(J)
0137          1701  OBJ=OBJ(C)
0138                IF(K.LE.2) GO TO 1775
0139                IF(OBJ.GT.FB(K-1)) GO TO 1774
0140                GO TO 1775
0141          1774  DO 1776 I=1,IP
0142          1776  D(I)=XB(I,K-1)
0143                FOPT=FB(K-1)
0144                DO 16 J=1,JP
0145          16    GN(J)=GO(J)
0146          1775  LI=0
0147                FOPT=OBJ
0148          177   CALL OPT2(FOPT,LI)
0149                IF(LI.GT.0) GO TO 5101
0150                RETURN
0151          5101  FB(1)=FOPT
0152                GO TO 510
0153          19    FB(I)=FB(K-1)
0154                DO 191 I=1,IP
0155          191   D(I)=XB(I,K-1)
0156                LI=2
0157                K=1
0158                GO TO 51
0159          1000  A=AO
0160                CALL PATTRN(KA,SB)
0161                IF(KA.EQ.0) GO TO 510
0162                FOPT=SB
0163                LI=0
0164                RETURN
0165                END
```

```
0001                SUBROUTINE DB(J,LI)
0002               IMPLICIT REAL*8(A-H,O-Z)
0003               COMMON D(10),P(100),DL(10),DU(10),DZ(10),A,XB(10,5),GN(10),FB(5),
                  CV(55,66),Q,IP,KP,JP,IDUN(22)
0004                COMMON IXJ
0005                COMMON/DBL/STEST,TDIF,AX1,AMIN,BL(10),AD,IN
0006                DIMENSION DT(10)
0007                IXJ=0
0008                TN=OBJ(J)
0009                IXJ=0
0010                DEL=.001
0011                IF(A.LT.DEL) DEL=A
0012                DO 1 I=1,IP
0013                DT(I)=D(I)
0014                D(I)=D(I)+DEL
0015                DZ(I)=(OBJ(J)-TN)/DEL
0016                IXJ=0
0017          1     D(I)=DT(I)
0018                DTEMP=0.
0019                ITEMP=0
0020                DO 2 I=1,IP
0021               IF(DABS(DZ(I)).LT.DTEMP) GO TO 2
0022                DTEMP=DABS(DZ(I))
0023                ITEMP=I
0024          2     CONTINUE
0025                IF(LI.EQ.2) RETURN
0026                A=AD/AX1
0027                IF(JP.EQ.0) GO TO 11
0028                DO 3 I=1,JP
0029          3     BL(I)=.1/AX1
0030          11    IF(LI.GT.0) RETURN
0031                A=DABS((0.01*TN)/DZ(ITEMP))
0032                IF(A.LT..01) A=.01
0033                AD=A
0034                AMIN=A/1000.
0035                PRINT 10,A,AMIN
0036          10    FORMAT(2F10.5)
0037                RETURN
0038                END
```

AD-A039 094 NEW JERSEY INST OF TECH NEWARK F/G 13/13
A FAST BOUNDARY TRACKING ALGORITHM FOR CONSTRAINED NONLINEAR MA--ETC(U)
MAR 77 J MORADI, M PAPPAS N00014-75-C-0987
UNCLASSIFIED NJIT-NV-12 NL

2 OF 2
ADA039094



END
DATE FILMED
5-77

```
0001                SUBROUTINE OPT2(FOPT,LI)
0002               IMPLICIT REAL*8(A-H,O-Z)
0003                INTEGER Z,E,G,BB,W,B,H
0004               COMMON D(10),P(100),DL(10),DU(10),DZ(10),V,XB(10,5),GG(10),FB(5),
                  CA(55,66),Q,IP,KP,JP,IA(10),KL,KW,N,M,L,B,BB,E,G,Z,H,W
0005                COMMON IXJ
0006               COMMON/ZE/GS,GSS,SE(10),DK(11,10),DT(10),DTT(10),DI,SMAL,JIN,K,MU,
                  C1M,JJ,J12,L11
0007                COMMON/POL/GB(10)
0008                COMMON/DOL/STEST,TDIF,AX1,AMIN,BL(10),AD,IN
0009                COMMON/PAT/DBJ,JK,IXYZ
0010               COMMON/CON/SS(10),BASE,IJ(10),JI(10),J1Q,IZJUMP
0011               DIMENSION IB(10),IC(10),SSS(55),TN(10),GG(10),X(10),DB1(11,10)
0012                BASE=1.
0013                SBASE=0.
0014                SB=FOPT
0015                JXN=1000
0016                J1N=1000
0017                J1X=0
0018                AL=V
0019                MU=0
0020                JIM=0
0021                JIN=0
0022                LI=0
0023                L11=0
0024           1    JJ=1
0025                VT=V/10.
0026                ICC=0
0027                IF(SBASE.EQ.BASE) GO TO 3001
0028                SBASE=BASE
0029                JIM=0
0030                JIN=0
0031                IF(AL.EQ.V) GO TO 101
0032                PRINT 1500,V,FOPT,(D(I),I=1,IP)
0033          1500  FORMAT(' CHECK POINT A=',F10.5,'FPT=',E16.8,'D=',5F10.5)
0034                PRINT 1501,(GG(J),J=1,JP)
0035          1501  FORMAT(' G=',10F10.5)
0036                SMAL=0
0037                SB=FOPT
0038                AL=V
0039                IF(SB.EQ.STEST) GO TO 150
0040               DIF=DABS((SB-STEST)/SB)
0041                IF(DIF.GT. 1.E-05) GO TO 45
0042                IN=IN+1
0043                IF(IN.LT.2) GO TO 150
0044                IF(DIF.GT.TDIF) GO TO 45
0045           46   FOPT=FB(I)
0046                LI=0
0047                RETURN
0048           150  V=V/2.
0049                DO 151 I=1,JP
0050          151   BL(I)=BL(I)/2.
0051                AX1=AX1/2.
0052                L11=1
0053                IF(V.LE.AMIN) GO TO 46
0054                GO TO 47
```

```
0055          45    IN=0
0056          47    STEST=SB
0057                TDIF=DIF
0058         101    JJ=1
0059                JK=1
0060                IZJUMP=0
0061                J2Q=0
0062                J3Q=0
0063                JU=1
0064                DI=10.
0065        3000    J12=1
0066                MU=0
0067                NU=0
0068                MP=0
0069                IF(V.GT.AMIN) GO TO 11
0070                FOPT=FB(JJ)
0071                LI=0
0072                RETURN
0073          11    DEL=.00001
0074               IF(V.LT.DEL) DEL=V
0075                IP1=JP+4*IP+4
0076                DO 103 I=1,IP1
0077         103    SSS(I)=0
0078               JK=1
0079               N=IP+1
0080               Z=-1
0081               MM=JP+1
0082               M=MM+2*N+2*IP
0083               L=MM+N+IP
0084               E=0
0085                G=0
0086               NN=6
0087          90   Q=99999
0088         190   B=M+N+G+1
0089               W=M
0090         320   H=1
0091               BB=B+1
0092               J=0
0093               DO 4 I=1,IP
0094               IB(I)=0
0095               IC(I)=0
0096               IF(D(I)-DL(I).LT.V)GO TO 5
0097                IF(DU(I)-D(I).LT.V)GO TO 6
0098               GO TO 4
0099         5     J=J+1
0100               IB(I)=J
0101                GO TO 4
0102         6      J=J+1
0103               IC(I)=J
0104         4      CONTINUE
0105               JX=J
0106               MM=JP+1
0107               IL=0
0108           156 J=0
0109               DO 22 I=1,MM
0110               II=I-1
```

```
0111              IF(I.NE.1) GO TO 78
0112               TM=FOPT
0113               FB(JJ)=TM
0114              GO TO 77
0115         78    CONTINUE
0116               IF(GO(II).GT.BL(II)) GO TO 22
0117               TM=GO(II)
0118            77 J=J+1
0119              TN(J)=TM
0120            22 CONTINUE
0121               IF(JX.EQ.JXN.AND.J.EQ.JIN) GO TO 113
0122               JIN=J
0123               JIM=J
0124               JXN=JX
0125               IF(JIM.EQ.JIN) GO TO 2000
0126               IF(JP.EQ.0) GO TO 221
0127               DO 220 J=1,JP
0128         220   GB(J)=GO(J)
0129               DBJ=FB(JJ)
0130            221 CONTINUE
0131              DO 23 K=1,IP
0132               XB(K,JJ)=D(K)
0133              DT(K)=D(K)
0134              D(K)=D(K)+DEL
0135              J=0
0136               IXJ=0
0137               DO 24 I=1,MM
0138              II=I-1
0139              IF(I.EQ.1) GO TO 777
0140               IJ(II)=0
0141               IF(GO(II).GT. BL(II)) GO TO 24
0142            777 J=J+1
0143               IF(II.NE.0) IJ(II)=J
0144               JI(J)=II
0145               IF(ICC .NE.0) GO TO 1000
0146               DBI(I,K)=DBJ(III
0147         1000  DK(J,K)=(DBI(I,K)-TN(J))/DEL
0148               IF(J.GT.1) DK(J,K)=-DK(J,K)
0149            24 CONTINUE
0150            23 D(K)=DT(K)
0151               JIN=J
0152              DO 25 I=1,J
0153              SUM=0.
0154              DO 26 K=1,IP
0155            26   SUM=SUM+DK(I,K)**2
0156               IF(SUM.EQ.0.) GO TO 250
0157               SUM=DSQRT(SUM)
0158              DO 27 K=1,IP
0159            27   DK(I,K)=DK(I,K)/SUM
0160           250   SE(I)=SUM
0161            25 CONTINUE
0162              JQ=0
0163           330 LX=W+2
0164               IF(J.GT.1) GO TO 775
0165               IF(JX.EQ.0) GO TO 152
0166           775 DO 350 I=1,LX
```

```
0167          340  DO 350 J=1,B
0168          350  A(I,J)=0
0169               J=0
0170               DO 2 I=1,MM
0171               II=I-1
0172               IF(I.EQ.1) GO TO 7
0173                IF(GO(II).GT.BL(II)) GO TO 2
0174          7    J=J+1
0175                ICC=1
0176                L=0
0177                DO 3 K=1,IP
0178               IF(IC(K).NE.0) GO TO 3
0179               L=L+1
0180               A(J,L)=DK(J,K)
0181          3    CONTINUE
0182               DO 10 K=1,IP
0183               IF(IB(K).NE.0) GO TO 10
0184               L=L+1
0185               A(J,L)=-DK(J,K)
0186          10    CONTINUE
0187               N=L+1
0188               IF(J.EQ.1)GO TO 2
0189                A(J,N)=0.
0190          2     CONTINUE
0191               A(1,N)=1.
0192               JA=J+1
0193               DO 8 I=1,L
0194               J=J+1
0195          8    A(J,I)=1.0
0196               W=J
0197               A(W+1,N)=1.0
0198               B=J+N+1
0199               BB=B+1
0200               DO 9 I=JA,W
0201           9   A(I,B)=1.0
0202               L=W
0203               DO 510 I=1,N
0204          510  A(W+1,I)=FLOAT(ZI)*A(W+1,I)
0205                M=J
0206               NN=6
0207               CALL LINPRO(K2)
0208               IF(K2.EQ.0) V=V/2
0209                IF(K2.EQ.0) GO TO 1
0210               LL=M+1
0211               DO 1460 I=1,LL
0212               J=A(I,BB)
0213                SSS(J)=A(I,B)
0214          1460  CONTINUE
0215               L=0
0216               DO 19 I=1,IP
0217               IF(IC(I).NE.0)GO TO 12
0218               L=L+1
0219                SS(I)=SSS(L)
0220               GO TO 19
0221          12    SS(I)=0
0222          19    CONTINUE
```

```
0223            DO 13 I=1,IP
0224            IF(IB(I).NE.0)GO TO 13
0225            L=L+1
0226             SS(I)=SS(I)-SSS(L)
0227      13    CONTINUE
0228       113  SUM=0.0
0229            DO 115I=1,IP
0230      115   SUM=SUM+SS(I)*SS(I)
0231             SUM=DSQRT(SUM)
0232            IF(SUM.EQ.0.)GO TO 1101
0233             GO TO 1150
0234      1101   LI=1
0235             IF(IXYZ.EQ.4) GO TO 3001
0236             RETURN
0237      1150  DO 16 I=1,IP
0238      16    SS(I)=V*SS(I)/SUM
0239      1001   DO 161 I=1,IP
0240             D(I)=XB(I,1)+SS(I)
0241      161   CONTINUE
0242       118   J10=0
0243             IXJ=0
0244             OBJ=OBJ(0)
0245             DFB=OBJ-FB(JJ)
0246             IF(OBJ.LT.FB(JJ)) GO TO 1180
0247             IF(IXYZ.EQ.4) GO TO 3001
0248             IF(IZJUMP.EQ.1) GO TO 1123
0249      3001   V=V/2.
0250             AXI=2.*AX1
0251             DO 108 I=1,JP
0252      108    BL(I)=BL(I)/2.
0253             LII=1
0254             IF(V.LT.VT) GO TO 1003
0255             IF(V.LT.AMIN) GO TO 1003
0256             FOPT=FB(1)
0257             DO 106 I=1,IP
0258      106    D(I)=XB(I,1)
0259             DO 107 J=1,JP
0260      107    GO(J)=GB(J)
0261             GO TO 101
0262      2000   DO 1002 I=1,IP
0263      1002   SS(I)=0.5*SS(I)
0264             GO TO 1001
0265      1003   IF(IXYZ.EQ.4) RETURN
0266             DO 1182 I=1,IP
0267      1182   D(I)=XB(I,1)
0268             DO 1004 J=1,JP
0269      1004   GO(J)=GB(J)
0270             LI=0
0271             FOPT=FB(1)
0272             RETURN
0273       152   IF(SE(1).EQ.0.) JK=2
0274             IF(SE(1).EQ.0.) RETURN
0275       153   DO 154 I=1,IP
0276      154    D(I)=DT(I)-DK(1,I)*V
0277             OBJ=OBJ(0)
0278             IF(OBJ.LT.FB(JJ)) GO TO 102
```

```
0279                V=V/2.
0280                IF(V.LT.VT) RETURN
0281                IF(V.GT.AMIN) GO TO 153
0282                RETURN
0283          102   JK=0
0284                RETURN
0285         1180   IXJ=0
0286                IF(IXYZ.EQ.4) JK=0
0287                IF(IXYZ.EQ.4) RETURN
0288                CALL CONMOV(IDU,FOPT)
0289                IF(IDU.EQ.1) GO TO 118
0290                GO TO 1
0291         1123   FOPT=FB(JJ)
0292                DO 1124 J=1,JP
0293         1124   GO(J)=CB(J)
0294                DO 1125 I=1,IP
0295         1125   D(I)=XB(I,JJ)
0296                GO TO 1
0297                END
```

```
0001                SUBROUTINE COMMOV(IDU,FOPT)
0002               IMPLICIT REAL*8(A-H,O-Z)
0003               COMMON D(10),P(100),DL(10),DU(10),DZ(10),V,XB(10,5),GO(10),FB(5),
                  CA(55,66),Q,IP,KP,JP,IA(10),KL, IDIM(11)
0004                COMMON IXJ
0005               COMMON/ZE/GS,GSS,SE(10),DK(11,10),DT(10),DTT(10),DI,SKAL,JIN,K,MU,
                  CIM,JJ,J12,LI1
0006                COMMON/POL/GB(10)
0007                COMMON/DOL/STEST,TDIF,AXI,AMIN,BL(10),AO,IN
0008                COMMON/PAT/OBJ,JK,IAYZ
0009               COMMON/CON/SS(10),BASE,IJ(10),JI(10),J1Q,IZJUMP
0010               DIMENSION GG(10),X(10)
0011                DO 16701=1,JP
0012                GG(I)=OBJ(I)
0013                IF(GG(I).GT.0.) GO TO 1660
0014                IF(GO(I).GT.BL(I)) GO TO 168
0015          1660  GO(I)=GG(I)
0016          1670  CONTINUE
0017                NO=0
0018                GO TO 1731
0019          1698  IXJ=0
0020                DO 169 I=2,JIN
0021                K1=JI(I)
0022                GO(K1)=OBJ(K1)
0023                GG(K1)=GO(K1)
0024          169   CONTINUE
0025          1731   CONTINUE
0026           130  JOB=0
0027                J30=0
0028                GPP=1000000.
0029                DO 120 M=2,JIN
0030                KK=JI(M)
0031                J=IJ(KK)
0032          1730  SUM=0
0033                IF(GO(KK).GT.0) GO TO 1732
0034                GO TO 1735
0035          1732  DO 1734 I=1,IP
0036          1734  SUM=SUM+DK(I,I)*DK(J,I)
0037                IF(SUM.GT.0) GO TO 120
0038          1735  JOB=JOB+1
0039                GP=GO(KK)/SE(J)
0040                IF(GP.GT.GPP) GO TO 120
0041                GPP=GP
0042                KPP=KK
0043                JPP=J
0044           120  CONTINUE
0045                IF(JOB.GT.0) GO TO 1737
0046                IF(JJ.GE.2) GO TO 1767
0047                JJ=JJ+1
0048                DO 31 I=1,IP
0049                SS(I)=2.*SS(I)
0050                XB(I,JJ)=D(I)
0051           31    D(I)=D(I)+SS(I)
0052                FB(JJ)=OBJ
0053                IXJ=0
0054                DO 313 I=1,JP
```

```
0055                DO 314 J=2,JIN
0056                K=JI(J)
0057                IF(I.EQ.K) GO TO 313
0058          314   CONTINUE
0059                GO(I)=OBJ(I)
0060          313   GB(I)=GO(I)
0061                IZJUMP=1
0062                IOU=1
0063                RETURN
0064          1737  CONTINUE
0065                K=JPP
0066                NU=KPP
0067                GS=GO(NU)
0068                IF(DABS(GS).LE..000001) GO TO 1767
0069                JUMP=0
0070                MU=NU
0071                CALL ZERO(JUMP)
0072          172   IF(SMAL.EQ.0) GO TO 310
0073                FOPT=FB(JJ)
0074                DO 311 I=1,IP
0075          311   D(I)=XB(I,JJ)
0076                DO 312 J=1,JP
0077          312   GO(J)=GB(J)
0078                IOU=2
0079                RETURN
0080          310   IF(J2Q.EQ.1.AND.JUMP.NE.2) J1Q=1
0081                IF(JUMP.NE.2) GO TO 1632
0082                GO TO 1698
0083          1632  CONTINUE
0084                J1=0
0085                IXJ=0
0086                DO 1634 I=2,JIN
0087                K=JI(I)
0088                IF(J3Q.GT.0) GO TO 1633
0089                GG(K)=OBJ(K)
0090                GO(K)=GG(K)
0091          1633  IF(GO(K).LT.(-.000001)) J1=J1+1
0092          1634  CONTINUE
0093                IF(J1.GT.0) GO TO 130
0094          1767  J2=0
0095                IXJ=0
0096                DO 1630 I=1,JP
0097                DO 1636 J=2,JIN
0098                K=JI(J)
0099                IF(I.EQ.K) GO TO 1630
0100          1636  CONTINUE
0101                IF(J3Q.GT.0) GO TO 1631
0102                GG(I)=OBJ(I)
0103                GO(I)=GG(I)
0104          1631  IF(GO(I).LT.(-.000001)) J2=J2+1
0105          1630  CONTINUE
0106                J3Q=0
0107                IF(J2.NE.0) GO TO 1773
0108                OBJ=OBJ(0)
0109                FOPT=OBJ
0110                IF(J1Q.EQ.1) IOU=2
```

```
0111                IF(J1Q.EQ.1) RETURN
0112                IF(JUMP.EQ.2) GO TO 1800
0113                GO TO 1766
0114          1800  DO 1801 I=1,IP
0115          1801  XB(I,1)=XB(I,JJ)
0116                FB(1)=FB(JJ)
0117                JJ=1
0118                GO TO 1766
0119          1773  GS=GG(NU)
0120                MU=NU
0121                DO 1765 I=1,JP
0122                IF(GG(I).GE.GS) GO TO 1765
0123                GS=GG(I)
0124                MU=I
0125          1765  CONTINUE
0126                IF(MU.NE.NU) GO TO 1764
0127          1766  JJ=JJ+1
0128                IF(JJ.LE.5) GO TO 1641
0129                DO 1642 I=1,IP
0130                XB(I,1)=XB(I,3)
0131                XB(I,2)=XB(I,4)
0132          1642  XB(I,3)=XB(I,5)
0133                FB(1)=FB(3)
0134                FB(2)=FB(4)
0135                FB(3)=FB(5)
0136                JJ=4
0137          1641  DO 164 I=1,IP
0138           164  XB(I,JJ)=D(I)
0139                FB(JJ)=DBJ
0140          1759  JN=JJ-1
0141          1635  IF(FB(JJ).GT.FB(JN)) GO TO 1760
0142                BASE=BASE+1
0143                ICC=0
0144                DO 1640 J=1,JP
0145          1640  GB(J)=GG(J)
0146          1763  CONTINUE
0147          1762  IF(J1Q) 165,165,119
0148          119   IOU=1
0149                RETURN
0150          1760  DO 1771 I=1,IP
0151          1771  D(I)=XB(I,JN)
0152                FOPT=FB(JN)
0153                DO 1772 J=1,JP
0154          1772  GG(J)=GB(J)
0155                IOU=2
0156                RETURN
0157           165  IF(JJ.GT.2) GO TO 112
0158                SUM=0.
0159                DO 1131 I=1,IP
0160                DC=XB(I,2)-XB(I,1)
0161                SUM=SUM+DC**2
0162          1131  D(I)=XB(I,2)+DC
0163                SUM=DSQRT(SUM)
0164                IF(SUM.LT.V/10.) GO TO 1123
0165                GO TO 1181
0166           112  JM=JJ-2
```

```
0167                SUM=0.
0168                DO 114 I=1,IP
0169                DC=XB(I,JJ)-XB(I,JM)
0170                IF(DC.EQ.0) GO TO 114
0171                SUM=SUM+DC**2
0172                IF(J2Q.EQ.1) DC=(DC*V)/DABS(DC)
0173          114   D(I)=XB(I,JJ)+DC
0174                SUM=DSQRT(SUM)
0175                IF(SUM.LT.V/10.) GO TO 1123
0176          1181  DO 1040 I=1,IP
0177                IF(D(I).LT.DL(I)) D(I)=DL(I)
0178          1040  IF(D(I).GT.DU(I)) D(I)=DU(I)
0179                IF(J2Q.NE.1) GO TO 1698
0180                DO 1612 I=1,IP
0181          1612  XB(I,1)=XB(I,JJ)
0182                JJ=1
0183                GO TO 1698
0184           168  DO 173 M=1,IP
0185                DTT(M)=D(M)
0186                X(M)=D(M)
0187                D(M)=XB(M,JJ)
0188           173  DT(M)=D(M)
0189                GS=GG(I)
0190                GG(I)=GG(I)
0191                NU=1
0192                BL(I)=2.*BL(I)
0193                J10=1
0194                LI=2
0195               IF(DABS(GS).LE..000001)GO TO 1632
0196                GSS=GG(I)
0197                DO 174 M=1,IP
0198           174  D(M)=X(M)
0199               IF(DABS(GSS).LE..000001)GO TO 1632
0200                IU=0
0201                X1=DABS(GSS-GS)
0202                X2=DLOG(X1)+1
0203                IM=5+IDINT(X2/.69314718)
0204                JUMP=1
0205                MU=NU
0206                LII=LI
0207                CALL ZERO(JUMP)
0208                MU=0
0209                GO TO 172
0210          1764  IU=0
0211                GSS=GG(MU)
0212                JACK=1
0213               IF(DABS(GSS).LE..000001)GO TO 1679
0214                DO  1768 I=1,IP
0215                DTT(I)=D(I)
0216                D(I)=XB(I,JJ)
0217          1768  DT(I)=D(I)
0218                GS=GB(MU)
0219                JACK=2
0220               IF(DABS(GS).LE..000001)GO TO 1679
0221                JACK=0
0222                X1=DABS(GSS-GS)
```

```
0223                X2=DLOG(X1)+1.
0224                IM=5+IDINT(X2/.69314718)
0225          1861  CONTINUE
0226                T=GS/(GS-GSS)
0227                DO 1769 I=1,IP
0228                D(I)=DT(I)+T*(DTT(I)-DT(I))
0229                IF(D(I).LT.DL(I)) D(I)=DL(I)
0230          1769  IF(D(I).GT.DU(I)) D(I)=DU(I)
0231                IXJ=0
0232                GO(MU)=OBJ(MU)
0233                GG(MU)=GO(MU)
0234               IF(DABS(GO(MU)).GT..000001) GO TO 1689
0235                GO TO 1679
0236          1689  GS=GSS
0237                GSS=GO(MU)
0238                IF(GS.LT.0) GO TO 1503
0239                GO TO 1504
0240          1503  IF(GSS.LT.GS) GO TO 1610
0241                GO TO 1505
0242          1504  IF(GSS.GT.GS) GO TO 1610
0243          1505  IU=IU+1
0244                IF(IU.GT.IM) GO TO 1610
0245                DO 1676 I=1,IP
0246                DT(I)=DTT(I)
0247          1676  DTT(I)=D(I)
0248                GO TO 1861
0249          1610  IF(JJ.EQ.JU) DI=DI*2.
0250                IF(JJ.GT.2) GO TO 1615
0251                DO 1619 I=1,IP
0252          1619  D(I)=XB(I,2)+(XB(I,2)-XB(I,1))/DI
0253                GO TO  1618
0254          1615  JM=JJ-2
0255                DO 162  I=1,IP
0256           162  D(I)=XB(I,JJ)+(XB(I,JJ)-XB(I,JM))/DI
0257                JU=JJ
0258          1618  DO 1616 I=1,IP
0259                IF(D(I).LT.DL(I)) D(I)=DL(I)
0260          1616  IF(D(I).GT.DU(I)) D(I)=DU(I)
0261                IF(LI.EQ.3) GO TO 1698
0262          1679   DBJ=OBJ(0)
0263                IF(DBJ.GT.FB(JJ))GO TO 1123
0264                IF(JACK.EQ.1) GO TO 1707
0265                IF(JACK.EQ.2) GO TO 1123
0266                DO 1677 I=1,JP
0267                IF(I.EQ.MJ) GO TO 1677
0268                GO(I)=OBJ(I)
0269                GG(I)=GO(I)
0270          1677  CONTINUE
0271          1707  CONTINUE
0272                JO=0
0273                DO 1121 J=1,JP
0274                IF(GO(J).LT.-.000001)JO=JO+1
0275          1121  CONTINUE
0276                IF(JO.LE.0) GO TO 1122
0277                J3Q=1
0278                J2Q=1
```

```
0279                GO TO 1632
0280          1122  LI=2
0281                IXJ=0
0282                FOPT=OBJ(0)
0283                IGU=2
0284                RETURN
0285          1123  FOPT=FB(JJ)
0286                DO 1124 J=1,JP
0287          1124  GO(J)=GB(J)
0288                DO 1125 I=1,IP
0289          1125  D(I)=XB(I,JJ)
0290                IGU=2
0291                RETURN
0292                END
```

```
0001              SUBROUTINE ZERO(JUMP)
0002             IMPLICIT REAL*8(A-H,O-Z)
0003             COMMON D(10),P(100),DL(10),DU(10),DZ(10),V,XB(10,5),GO(10),FB(5),
                CA(55,66),O,IP,IDUM(24)
0004               COMMON IXJ
0005             COMMON/ZE/GS,GSS,SE(10),DK(11,10),DT(10),DTT(10),DI,SMAL,JIN,K,NU,
                CIM,JJ,J12,LI
0006               DIMENSION GG(10)
0007               IU=0
0008               IF(JUMP.EQ.1) GO TO 175
0009               SMAL=0
0010              KU=0
0011        1738   CONTINUE
0012               DO 1740 I=1,IP
0013               DT(I)=D(I)
0014        1740   D(I)=DT(I)+DK(K,I)*GS/SE(K)
0015        1700   CONTINUE
0016               DO 1032 I=1,IP
0017               IF(D(I).LT.DL(I)) D(I)=DL(I)
0018               IF(D(I).GT.DU(I)) D(I)=DU(I)
0019        1032   DTT(I)=D(I)
0020               IXJ=0
0021               GSS=OBJ(NU)
0022               IF(GS.LT.0.) GO TO 1701
0023               GO TO 1702
0024        1701   IF(GSS.LT.GS) GO TO 1703
0025               GO TO 1704
0026        1702   IF(GSS.GT.GS) GO TO 1703
0027               GO TO 1704
0028        1703   IF(KU.LE.0) GO TO 1713
0029               GO TO 1710
0030        1713   DO 1705 I=1,IP
0031        1705   D(I)=DT(I)
0032        1750   CALL DB(NU,LI)
0033               SUMM=0.
0034               J=K
0035               DO 1706 I=1,IP
0036               DK(J,I)=DZ(I)
0037        1706   SUMM=SUMM+DZ(I)**2
0038               SUMM=DSQRT(SUMM)
0039               SE(J)=SUMM
0040               DO 1707 I=1,IP
0041        1707   DK(J,I)=-DK(J,I)/SUMM
0042               KU=1
0043               GO TO 1738
0044        1704  IF(DABS(GSS).LE..000001) GO TO 1632
0045               X1=DABS(GSS-GS)
0046               X2=DLOG(X1)+1.
0047               IM=5+IDINT(X2/.69314718)
0048               IU=0
0049        1800   CONTINUE
0050        175    T=GS/(GS-GSS)
0051        1751   DO 163 I=1,IP
0052        1631   D(I)=DT(I)+T*(DTT(I)-DT(I))
0053        163    CONTINUE
0054        1600   DO 160 I=1,IP
```

```
0055                IF(D(I).LT.DL(I)) D(I)=DL(I)
0056          160   IF(D(I).GT.DU(I)) D(I)=DU(I)
0057                IXJ=0
0058                GG(NU)=OBJ(NU)
0059               IF(DABS(GG(NU)).GT..000001) GO TO 1601
0060                GO TO 1632
0061          1601  GS=GSS
0062                GSS=GG(NU)
0063                IF(GS.LT.0) GO TO 1603
0064                GO TO 1604
0065          1603  IF(GSS.LT.GS) GO TO 1710
0066                GO TO 1605
0067          1604  IF(GSS.GT.GS) GO TO 1710
0068          1605  IU=IU+1
0069                IF(IU.GT.IM) GO TO 1710
0070                DO 1602  L=1,IP
0071                DT(L)=DTT(L)
0072          1602  DTT(L)=D(L)
0073                GO TO 1600
0074          1710  IF(JJ.EQ.J12) DI=2.*DI
0075                SAN=0.
0076                IF(JJ.GT.2) GO TO 1715
0077                IF(JJ.EQ.2) GO TO 1100
0078                DO 1200 I=1,IP
0079          1200  XB(I,2)=D(I)
0080          1100  DO 1719 I=1,IP
0081                DX=XB(I,2)-XB(I,1)
0082                SAN=SAN+DX**2
0083          1719  D(I)=XB(I,2)+DX/DI
0084                SAN=DSQRT(SAN)
0085                SAN=SAN/DI
0086                IF(SAN.LT.V) GO TO 1
0087                GO TO 1716
0088          1715  JM=JJ-2
0089                DO 172 I=1,IP
0090                DX=XB(I,JJ)-XB(I,JM)
0091                SAN=SAN+DX**2
0092          172   D(I)=XB(I,JJ)+DX/DI
0093                SAN=DSQRT(SAN)
0094                SAN=SAN/DI
0095                IF(SAN.LT.V) GO TO 1
0096          1716  DO 1717 I=1,IP
0097                IF(D(I).LT.DL(I)) D(I)=DL(I)
0098           1717 IF(D(I).GT.DU(I)) D(I)=DU(I)
0099                J12=JJ
0100                JUMP=2
0101          1632  RETURN
0102           1    SMAL=1
0103                RETURN
0104                END
```

```
0001              SUBROUTINE LINPRO(K2)
0002              IMPLICIT REAL*8(A-H,O-Z)
0003              INTEGER Z,E,G,BB,W,B,R,C,H
0004              COMMON D(10),U(100),DL(10),DU(10),DZ(10),V,XB(10,5),GN(10),FB(5),
                 CA(55,66),Q,IP,KP,JP,IA(10),KL,KW,N,M,L,B,BB,E,G,Z,H,W
0005               COMMON IXJ
0006              K2=1
0007              NN=6
0008              M=M-1.0
0009         560  LL=M+2
0010              DO 560 K=2,LL
0011          570 A(K-1,N+G+K-1)=1
0012          580 A(K-1,BB)=K+N+G-1
0013          600 IF(G.NE.0) GO TO 620
0014          610 IF(E.EQ.0) GO TO 780
0015          611 GO TO 650
0016         620  KK=L+E+2
0017              LL=M+2
0018              DO 630 K=KK,LL
0019          630 A(K-1,K+N-L-E-1)=-1
0020          650 W=W+1
0021          660 Q=0
0022         670  LL=N+G
0023              DO 760 J=1,LL
0024          680 S=0
0025         690  LL1=M-G-E+2
0026              KK1=M+1
0027              DO 700 I=LL1,KK1
0028          700 S=S+A(I,J)
0029          720 A(W+1,J)=-S
0030          730 IF(A(W+1,J).GT.Q) GO TO 760
0031          740 Q=A(W+1,J)
0032          750 C=J
0033         760  CONTINUE
0034          761 S=0
0035         762  LL=M-G-E+2
0036              KK=M+1
0037              DO 763 J=LL,KK
0038          763 S=S+A(J,B)
0039          765 A(W+1,B)=-S
0040           780 CONTINUE
0041          790 IF(G.EQ.0) GO TO 810
0042              LL=N+1
0043              KK=N+G
0044          810 IF(L.EQ.0) GO TO 830
0045               LL=N+G+1
0046               KK=N+G+L
0047               IQ=I
0048          830  IF(G+E.EQ.0) GO TO 2000
0049         831  LL=N+G+L+1
0050              KK=B-1
0051              IQ=1
0052          860 GO TO 2000
0053          895 IF(Q.EQ.99999) GO TO 1230
0054          900 IF(Q.EQ.0) GO TO 1330
0055          910 GO TO 1400
```

```
0056          920  M=M+1
0057          930  Q=.1E39
0058          940  R=-1
0059               LL=M+1
0060          950  DO 1000 I=1,LL
0061          960  IF(A(I,C).LE.0) GO TO 1000
0062          970  IF(A(I,B)/A(I,C).GT.Q) GO TO 1000
0063          980  Q=A(I,B)/A(I,C)
0064          990  R=I
0065         1000  CONTINUE
0066         1010  IF(DFLOAT(R).GE.-.5) GO TO 1050
0067               IQ=2
0068         1030  GO TO 2000
0069         1050  P=A(R,C)
0070         1060  A(R,BB)=C
0071         1070  DO 1080 J=1,B
0072         1080  A(R,J)=A(R,J)/P
0073               LL=M+1
0074         1100  DO 1180 I=1,LL
0075         1110  IF(I.EQ.R) GO TO 1180
0076         1120  DO 1170 J=1,B
0077         1130  IF(J.EQ.C) GO TO 1170
0078         1140  A(I,J)=A(I,J)-A(R,J)*A(I,C)
0079         1150  IF(DABS(A(I,J)).GT..1E-4)GO TO 1170
0080         1160  A(I,J)=0
0081         1170  CONTINUE
0082         1180  CONTINUE
0083               LL=M+1
0084         1190  DO 1200 I=1,LL
0085         1200  A(I,C)=0
0086         1210  CONTINUE
0087         1220  A(R,C)=1
0088         1230  G=0
0089         1240  LL=N+G+L
0090               DO 1260 J=1,LL
0091         1250  IF(A(M+1,J).GT.Q)GO TO 1280
0092         1260  Q=A(M+1,J)
0093         1270  C=J
0094         1280  CONTINUE
0095         1290  GO TO 900
0096         1330  IF(M.EQ.M+1) GO TO 1360
0097         1340  M=M-1
0098         1350  IF(A(M+2,B).LT..1E-5) GO TO 1353
0099               K2=C
0100         1352  RETURN
0101         1353  LL=M+1
0102               DO 1358 I=1,LL
0103         1354  IF(IDINT(A(I,BB)).LE.N+G+L) GO TO 1358
0104         1355  DO 1356 J=1,B
0105         1356  A(I,J)=0
0106         1358  CONTINUE
0107         1359  GO TO 1230
0108         1360   CONTINUE
0109         1400  IF(C.EQ.0) GO TO 1420
0110         1420  LL=M+1
0111         1430  DO 1460 I=1,LL
```

```
0112          1440  IF(IDINT(A(I,BB)).EQ.0) GO TO 1460
0113          1460  CONTINUE
0114          1470  IF(Q.NE.0) GO TO 920
0115          1520  KK=N+1
0116                LL=B-G-1
0117                XJJ=-2*A(N+1,B)
0118                LL=H-1
0119                IQ=3
0120          1550  GO TO 2000
0121          2000  LL=H-1
0122                LL=N+1
0123                GO TO(895,1050,999),IQ
0124          999   RETURN
0125                END
```

```
0001                SUBROUTINE PATTRN(KN,SB)
0002                 IMPLICIT REAL*8(A-H,O-Z)
0003               COMMON D(10),P(100),DL(10),DU(10),DZ(10),A,XB(10,5),G (10),FB(5),
                  CV(55,66),Q,IP,KP,JP,IDUN(22)
0004                COMMON IXJ
0005               COMMON/FIN/X(10),CK(10),IA(10),IK,IL,IC(10)
0006                COMMON/PAT/ST,JK,IXYZ
0007                COMMON/BAS/IK1,IK2
0008               DIMENSION AD(10,10),EI(10,10),ET(10,10),AZ(10),IX(10)
0009                IXYZ=4
0010                AMIN=.0000001
0011                IK1=0
0012                IK2=0
0013                LI=2
0014                DI=10.*A
0015               IDIF=10.E+10
0016               IPP=IP
0017               AL=A
0018               IM=0
0019               STEST=0.
0020               KA=0
0021               KW=0
0022                KT=0
0023               KL=0
0024               KF=0
0025               DO 111 K=1,IPP
0026               IF(D(K).LT.DL(K))D(K)=DL(K)
0027               IF(D(K).GT.DU(K))D(K)=DU(K)
0028          111 X(K)=D(K)
0029           25  KK=1
0030               DO 1 K=1,IPP
0031               DO 22 I=1,IPP
0032               EI(I,K)=0.
0033                ET(I,K)=0.
0034            22 AD(I,K)=0.
0035               ET(K,K)=1.
0036               EI(K,K)=1.
0037             1 CONTINUE
0038               ST=SB
0039               IF(KA.EQ.1)GO TO 9
0040                KA=1
0041          190   IK=0
0042                CALL FIND(SB,I)
0043                IF(I.NE.0) GO TO 1000
0044           9    ST=SB
0045               GO TO 100
0046           19   IK=0
0047                CALL FIND(ST,I)
0048                IF(I.NE.0) GO TO 1000
0049          100  KL=KL+1
0050                IK=1
0051               KF=KF+1
0052               DO 2 J=1,IPP
0053               DO 17 K=1,IPP
0054               AZ(K)= EI(J,K)*A
0055                IF(KK.NE.0) AZ(K)=ET(J,K)*A
```

```
0056         17     D(K)=D(K)+AZ(K)
0057                DO 35 K=1,IPP
0058                IF(D(K).GT.DU(K))GO TO 4
0059                IF(D(K).LT.DL(K))GO TO 4
0060            35 CONTINUE
0061                CALL FIND(SA,I)
0062                IF(I.NE.0) GO TO 1000
0063                IF((ST-SA)/DABS(SB).GT.2.E-16) GO TO 5
0064             4 DO 18 K=1,IPP
0065         18     D(K)=D(K)-AZ(K)-AZ(K)
0066                DO 36 K=1,IPP
0067                IF(D(K).LT.DL(K))GO TO 3
0068                IF(D(K).GT.DU(K))GO TO 3
0069            36 CONTINUE
0070                CALL FIND(SA,I)
0071                IF((ST-SA)/DABS(SB).LT.2.E-16) GO TO 3
0072             5 ST=SA
0073               GO TO 2
0074           3   DO 21 K=1,IPP
0075         21     D(K)=D(K)+AZ(K)
0076           2   CONTINUE
0077                IF((SB-ST)/DABS(SB) .LT.2.E-16) GO TO 7
0078         26     SUM=0.
0079                IM=0
0080                KX=0
0081                DO 8 K=1,IPP
0082                IX(K)=0
0083                D1=2.*D(K)-X(K)
0084               AD(1,K)=D(K)-X(K)
0085                DX=X(K)
0086                IF(DX.EQ.0.) DX=1.
0087                IF(DABS(AD(1 ,K)/DX  ).LT.1.E-15) GO TO 88
0088                IX(K)=1
0089                KX=KX+1
0090               SUM=SUM+AD(1,K)*AD(1,K)
0091          88    X(K)=D(K)
0092               IF(D1.LT.DL(K))D1=DL(K)
0093               IF(D1.GT.DU(K))D1=DU(K)
0094          8     D(K)=D1
0095                SB=ST
0096               SUM=DSQRT(SUM)
0097                IF(SUM.LT.A*.5) GO TO 7
0098                KK=0
0099                DO 11 K=1,IP
0100                ET(1,K)=EI(1,K)
0101                DX=X(K)
0102                IF(DX.EQ.0.) DX=1.
0103               IF(DABS(AD(1,K)/DX  ).LT. 1.E-15) AD(1,K)=0.
0104          11    EI(1,K)= AD(1,K)/SUM
0105                DO 12 I=2,IP
0106                DO 12  J=1,IP
0107                ET(I,J)=EI(I,J)
0108                EI(I,J)=0.
0109          12    AD(I,J)=0.
0110                K=1
0111                DO 13 I=1,IP
```

```
0112                IF(IX(I).EQ.0) GO TO 13
0113                M=I+1
0114                IF(M.GT.IP) GO TO 89
0115                 K=K+1
0116                L=K-1
0117                SUMA=0
0118                DO 14 J=M,IP
0119                AD(K,J)=AD(1,J)
0120          14    SUMA=SUMA+EI(L,J)*AD(K,J)
0121                SUM=0
0122                DO 15 J=1,IP
0123                EI(K,J)=AD(K,J)-EI(L,J)*SUMA
0124                IF(DABS(EI(K,J)).LE.1.E-35) GO TO 15
0125                SUM=SUM+EI(K,J)**2
0126          15    CONTINUE
0127                IF(SUM.LT.1.E-32) GO TO 13
0128                SUM=DSQRT(SUM)
0129                DO 16 J=1,IP
0130                DX=X(K)
0131                IF(DX.EQ.0.) DX=1.
0132                IF(DABS(EI(K,J)/DX  ).LT.1.E-15) EI(K,J)=0.
0133          16    EI(K,J)=EI(K,J)/SUM
0134          13     CONTINUE
0135          89    J=KX
0136                DO 180I=1,IP
0137                IF(IX(I).EQ.I) GO TO 180
0138                J=J+1
0139                 EI(J,I)=1.
0140          180   CONTINUE
0141               GO TO 19
0142             7 DO 20 K=1,IPP
0143            20 D(K)=X(K)
0144               IF(KK)25,24,23
0145          24    KK=-1
0146                IF(KF.GT.200)GO TO 23
0147                 GO TO 9
0148          23    CONTINUE
0149                 LI=2
0150                 TN=SB
0151          32     IF(KL.GT.KT+4) GO TO 33
0152                 A=A/2.
0153          33     CALL OPT2(SB,LI)
0154                 PRINT 1500,A,SB,ST,(D(I),I=1,IP)
0155        1500     FORMAT( ' CHECK POINT A=',F10.5,'SB=',E16.8,'ST=',E16.8,'D=',5F12
                   1.7)
0156                 KT=KL
0157                 IF(JK.EQ.2) KN=1
0158                 IF(JK.EQ.2) GO TO 1001
0159          311    KF=0
0160          301    IF(JK.NE.0) GO TO 46
0161                 IF(A.EQ.AL) GO TO 26
0162                 AL=A
0163                DIF=DABS((SB-STEST)/SB)
0164                 IF(DIF.GT. 1.E-05) GO TO 45
0165                 IM=IM+1
0166                 IF(IM.LT.2) GO TO 150
```

```
0167                IF(DIF.GT.TDIF) GO TO 45
0168                 GO TO 1001
0169          45     IM=0
0170          149    STEST=SB
0171                 TDIF=DIF
0172                 GO TO 26
0173          148    IM=0
0174                 A=A/2.
0175          46     DO 47 K=1,IPP
0176          47     D(K)=X(K)
0177                IF(A.LT.AMIN)  GO TO 1001
0178                GO TO 190
0179          150   A=A/2
0180                 IF(A.LT.AMIN) GO TO 1001
0181                 GO TO 149
0182          1001   DO 1003 I=1,IP
0183          1003   D(I)=X(I)
0184                 KN=1
0185                 RETURN
0186          1000   DO 1002 I=1,IP
0187          1002   D(I)=X(I)
0188                 KN=0
0189                 RETURN
0190                END
```

```
0001                  SUBROUTINE FIND(TD,I)
0002                   IMPLICIT REAL*8(A-H,O-Z)
0003                  COMMON D(10),P(100),DL(10),DU(10),D2(10),A,XB(10,5),G (10),FB(5),
                     CV(55,66),Q,IP,KP,JP,IDUN(22)
0004                   COMMON IXJ
0005                  COMMON/FIN/X(10),CK(10),IA(10),IK,IL,IC(10)
0006                   DIMENSION DT(10)
0007                   LI=2
0008                  I=0
0009                   IXJ=0
0010                  TD=OBJ(0)
0011                  IF(JP.EQ.0) RETURN
0012                  DO 25 K=1,IP
0013                25 DT(K)=D(K)
0014                   IF(IK.EQ.0) GO TO 51
0015                   IF(IL.EQ.0) RETURN
0016                   DO 52 J=1,IL
0017                   IB=IC(J)
0018                   G(IB)=OBJ(IB)
0019                   IF(G(IB)) 5,5,52
0020           5       I=I+1
0021                   IA(I)=IB
0022           52      CONTINUE
0023                   RETURN
0024           51      IL=0
0025                  DO 3 J=1,JP
0026                  G(J)=OBJ(J)
0027                   IF(G(J).GT. 0) GO TO 3
0028           6       IL=IL+1
0029                   IC(IL)=J
0030                   IF(G(J)) 4,4,3
0031                4 I=I+1
0032                  IA(I)=J
0033                3 CONTINUE
0034                   RETURN
0035                   END
```

```
0001                REAL FUNCTION OBJ*8(KLX)
0002                IMPLICIT REAL*8(A-H,O-Z)
0003                COMMON D(10),P(100),DUM(3727),IDUM(25)
0004          101   OBJ=B-U
0005                 IF(B.EQ.0. .OR.U.EQ.0.) RETURN
0006                OBJ=(B-U)/DABS(U)
0007                RETURN
0008                 END
```

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NJIT [redacted]- NV-12	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): A FAST BOUNDARY TRACKING ALGORITHM FOR CONSTRAINED NONLINEAR MATHEMATICAL PROGRAMMING PROBLEMS,	5. TYPE OF REPORT & PERIOD COVERED: INTERIM rept.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): JACOB MORADI AND MICHAEL PAPPAS	8. CONTRACT OR GRANT NUMBER(s): ONR-N 00014-75-C-0987
9. PERFORMING ORGANIZATION NAME AND ADDRESS: NEW JERSEY INSTITUTE OF TECHNOLOGY, 323 HIGH STREET, NEWARK, NEW JERSEY 07102	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 119 p.
11. CONTROLLING OFFICE NAME AND ADDRESS: DR. NICHOLAS PERRONE, (Code 474), OFFICE OF NAVAL RES., 800NO. QUINCY ST. ARLINGTON, VIRGINIA 22203	12. REPORT DATE: MAR [redacted]77
	13. NUMBER OF PAGES: 119
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

MATHEMATICAL PROGRAMMING; DESIGN, OPTIMAL, OPTIMIZATION, NUMERICAL, DESIGN, COMPUTER-AIDED

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A FAST SEARCH ALGORITHM FOR THE SOLUTION OF NONLINEAR MATHEMATICAL PROGRAMMING OPTIMIZATION PROBLEMS IS PRESENTED IN THIS REPORT. THE PROCEDURE COMBINES A "BOUNDARY TRACKING" (BT) STRATEGY WITH THE FEASIBLE DIRECTION FINDING METHOD OF ZOUTENDIJK. THIS ALGORITHM WAS COMPARED WITH TWENTY OTHER CODES REPRESENTING MOST OF THE POPULAR NUMERICAL OPTIMIZATION METHODS ON TEN TEST PROBLEMS. THE NEW CODE PROVED SUPERIOR TO ALL OTHERS IN OVERALL GENERALITY AND EFFICIENCY.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601 | UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

392 024 mt